
NETWORK FORENSICS



RIC MESSIER

Network Forensics

Network Forensics

Ric Messier

WILEY

<https://t.me/learningnets>

Network Forensics

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2017 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-119-32828-5
ISBN: 978-1-119-32917-6 (ebk)
ISBN: 978-1-119-32918-3 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or website may provide or recommendations it may make. Further, readers should be aware that Internet websites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2017941046

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

This book is dedicated to Atticus and Zoey, who got me through many years.

About the Author

Ric Messier, MS, GCIH, GSEC, CEH, CISSP is an author, consultant, and educator. He has decades of experience in information technology and information security. He has been a programmer, system administrator, network engineer, security engineering manager, VoIP engineer, consultant, and professor. He is currently Director for Cyber Academic Programs at Circadence and was formerly the Program Director for Cybersecurity and Digital Forensics at Champlain College in Burlington, VT. He has published several books on information security and digital forensics.

About the Technical Editor

Charlie Brooks first encountered the Internet in 1978, and hasn't strayed far from it since. Charlie spent 25 years in software development as a developer, technical lead, and software architect, working on software systems for network management, network performance analysis, and managed VPN services. He has been working in information security since 2005 as a course developer and instructor, first in data storage at EMC and then in network security analysis and forensics at RSA. Charlie has developed and taught graduate level courses in network security, data communications, incident response and network forensics, and software security at several colleges and universities in the Greater Boston area, including Boston University and Brandeis University. He currently teaches and develops courses for the Continuing Professional Studies division of Champlain College in Burlington, VT, in the master's programs for Digital Forensics and Operational Security.

Charlie has served as a technical editor for several books, and is the author of *All-In-One CHFI Computer Hacking Forensics Investigator Certification Exam Guide* from McGraw-Hill (2014), and "Securing the Storage Infrastructure" in *Information Storage and Management: Managing and Protecting Digital Information* (EMC Education, 2011). He holds an MS in Computer Information Systems from Boston University, and the CISSP, CHFI, and CTT+ certifications.

Credits

Project Editor

Tom Dinse

Production Editor

Athiyappan Lalith Kumar

Copy Editor

Kimberly A. Cofer

Production Manager

Katie Wisor

Manager of Content Development & Assembly

Mary Beth Wakefield

Marketing Manager

Christie Hilbrich

Professional Technology & Strategy Director

Barry Pruett

Business Manager

Amy Knies

Executive Editor

Jim Minatel

Project Coordinator, Cover

Brent Savage

Proofreader

Nancy Bell

Indexer

Nancy Guenther

Cover Designer

Wiley

Cover Image

© Andrey Prokhorov/iStockphoto

Contents at a Glance

Introduction	xxi
1 Introduction to Network Forensics.	1
2 Networking Basics.	13
3 Host-Side Artifacts.	53
4 Packet Capture and Analysis	81
5 Attack Types	113
6 Location Awareness.	143
7 Preparing for Attacks.	159
8 Intrusion Detection Systems	187
9 Using Firewall and Application Logs.	211
10 Correlating Attacks	245
11 Network Scanning.	265
12 Final Considerations	291
Index	319

Contents

Introduction	xxi
1 Introduction to Network Forensics.....	1
What Is Forensics?	3
Handling Evidence	4
Cryptographic Hashes	5
Chain of Custody	8
Incident Response	8
The Need for Network Forensic Practitioners	10
Summary	11
References	12
2 Networking Basics.....	13
Protocols	14
Open Systems Interconnection (OSI) Model	16
TCP/IP Protocol Suite	18
Protocol Data Units	19
Request for Comments.....	20
Internet Registries.....	23
Internet Protocol and Addressing	25
Internet Protocol Addresses	28
Internet Control Message Protocol (ICMP).....	31
Internet Protocol Version 6 (IPv6).....	31
Transmission Control Protocol (TCP).....	33
Connection-Oriented Transport.....	36
User Datagram Protocol (UDP).....	38
Connectionless Transport.....	39
Ports.....	40
Domain Name System	42

Support Protocols (DHCP)	46
Support Protocols (ARP)	48
Summary	49
References	51
3 Host-Side Artifacts	53
Services	54
Connections	60
Tools	62
netstat	63
nbtstat	66
ifconfig/ipconfig	68
Sysinternals	69
ntop	73
Task Manager/Resource Monitor	75
ARP	77
/proc Filesystem	78
Summary	79
4 Packet Capture and Analysis	81
Capturing Packets	82
Tcpdump/Tshark	84
Wireshark	89
Taps	91
Port Spanning	93
ARP Spoofing	94
Passive Scanning	96
Packet Analysis with Wireshark	98
Packet Decoding	98
Filtering	101
Statistics	102
Following Streams	105
Gathering Files	106
Network Miner	108
Summary	110

5	Attack Types	113
	Denial of Service Attacks	114
	SYN Floods	115
	Malformed Packets	118
	UDP Floods	122
	Amplification Attacks	124
	Distributed Attacks	126
	Backscatter	128
	Vulnerability Exploits	130
	Insider Threats	132
	Evasion	134
	Application Attacks	136
	Summary	140
6	Location Awareness	143
	Time Zones	144
	Using whois	147
	Traceroute	150
	Geolocation	153
	Location-Based Services	156
	WiFi Positioning	157
	Summary	158
7	Preparing for Attacks	159
	NetFlow	160
	Logging	165
	Syslog	166
	Windows Event Logs	171
	Firewall Logs	173
	Router and Switch Logs	177
	Log Servers and Monitors	178
	Antivirus	180
	Incident Response Preparation	181
	Google Rapid Response	182

Commercial Offerings	182
Security Information and Event Management.	183
Summary	185
8 Intrusion Detection Systems	187
Detection Styles	188
Signature-Based	188
Heuristic	189
Host-Based versus Network-Based	190
Snort	191
Suricata and Sagan	201
Bro.	203
Tripwire	205
OSSEC	206
Architecture	206
Alerting	207
Summary	208
9 Using Firewall and Application Logs.	211
Syslog.	212
Centralized Logging.	216
Reading Log Messages.	220
LogWatch	222
Event Viewer.	224
Querying Event Logs	227
Clearing Event Logs	231
Firewall Logs.	233
Proxy Logs.	236
Web Application Firewall Logs	238
Common Log Format.	240
Summary	243
10 Correlating Attacks	245
Time Synchronization.	246
Time Zones	246

Network Time Protocol	247
Packet Capture Times	249
Log Aggregation and Management	251
Windows Event Forwarding	251
Syslog	252
Log Management Offerings	254
Timelines	257
Plaso	258
PacketTotal.	259
Wireshark.	261
Security Information and Event Management.	262
Summary	263
11 Network Scanning.	265
Port Scanning.	266
Operating System Analysis	271
Scripts	273
Banner Grabbing	275
Ping Sweeps.	278
Vulnerability Scanning	280
Port Knocking.	285
Tunneling	286
Passive Data Gathering	287
Summary	289
12 Final Considerations	291
Encryption	292
Keys.	293
Symmetric	294
Asymmetric	295
Hybrid	296
SSL/TLS	297
Cloud Computing	306
Infrastructure as a Service	306

Storage as a Service	309
Software as a Service	310
Other Factors.	311
The Onion Router (TOR).	314
Summary	317
Index	319

Introduction

One of the best things about the different technology fields, should you have the stomach for it—and many don't—is the near constant change. Over the decades I have been involved in technology-based work, I've either had to or managed to reinvent myself and my career every handful of years or less. The world keeps changing and in order to maintain pace, we have to change too. In one of my incarnations that ended not many months ago now, I ran graduate and undergraduate programs at Champlain College in its online division. One of my responsibilities within that role was overseeing development of course materials. Essentially, either I or someone I hired developed the course and then I hired people who could teach it, often the people who did the development, though not always.

In the process of developing a course on network forensics, I discovered that there wasn't a lot of material around that covered it. At the time, I was able to find a single book but it wasn't one that we could make use of at the college because of policies focused on limiting costs to students. As a result, when I was asked what my next book would be, a book on network forensics that would explore in more detail the ideas I think are really important to anyone who is doing network investigations made the most sense to me.

What This Book Covers

I like to understand the why and how of things. I find it serves me better. When I understand the why and how, I don't get stuck in a dinosaur graveyard because at its core, technology continues to cycle around a number of central ideas. This has always been true. When you understand what underpins the technology, you'll see it's a variation on something you've seen before, if you stick around long enough. As a result, what is covered in this book is a lot of "how and why" and less of "these are the latest trendy tools" because once you understand the how and why, once you get to what's underneath, the programs can change and you'll still understand what it is you are looking at, rather than expecting the tools to do the work for you.

This is the reason why this book, while offering up some ideas about investigations, is really more about the technologies that network investigations are looking at. If you understand how networks work, you'll know better where to look for the information you need. You'll also be able to navigate changes. While we've moved from coax to twisted pair to optical to wireless, ultimately the protocols have remained the same for decades. As an example, Ethernet was developed in the 1970s and your wireless network connection, whether it's at home or at your favorite coffee shop down the street, still uses Ethernet. We're changing the delivery mechanism without changing what is being delivered. Had you learned how Ethernet worked in the early 1980s, you could look at a frame of Ethernet traffic today and still understand exactly what is happening.

The same is true of so-called cloud computing. In reality, it's just the latest term for outsourcing or even the service bureaus that were big deals in the '70s and '80s. We outsource our computing needs to companies so we don't have to deal with any of the hassle of the equipment and we can focus on the needs of the business. Cloud computing makes life much easier because delivery of these services has settled down to a small handful of well-known protocols. We know how they all work so there is no deciphering necessary.

At the risk of over-generalizing, for many years now there has been a significant emphasis on digital forensics, seen particularly through the lens of any number of TV shows that glorify the work of a forensic investigator and, in the process, get huge chunks of the work and the processes completely wrong. So-called dead-box forensics has been in use for decades now, where the investigator gets a disk or a disk image and culls through all the files, and maybe even the memory image for artifacts. The way people use computers and computing devices is changing. On top of that, as more and more businesses are affected by incidents that have significant financial impact, they have entirely different needs.

The traditional law enforcement approach to forensics is transitioning, I believe, to more of a consulting approach or an incident response at the corporate level. In short, there will continue to be a growing need for people who can perform network investigations as time goes on. With so many attackers in the business of attacking—their attacks, thefts, scams, and so on are how they make their living—the need for skilled investigators is unlikely to lessen any time in the near future. As long as there is money to be made, you can be sure the criminal incidents will continue.

As you read through this book, you will find that the “what's underneath” at the heart of everything. We'll talk about a lot of technologies, protocols, and products, but much of it is with the intention of demonstrating that the more things change, the more they stay the same.

How to Use This Book

I've always been a big believer in a hands-on approach to learning. Rather than just talking about theories, you'll look at how the tools work in the field. However, this is not a substitute for actually using them yourself. All of the tools you look at in this book are either open source or have community editions, which means you can spend time using the tools yourself by following along with the different features and capabilities described in each chapter. It's best to see how they all behave in your own environment, especially since some of the examples provided here may look and behave differently on your systems because you'll have different network traffic and configurations. Working along with the text, you'll not only get hands-on experience with the tools, but you will see how everything on your own systems and networks behaves.

How This Book Is Organized

This book is organized so that chapter topics more or less flow from one to the next.

Chapter 1 provides a foundational understanding of forensics. It also looks at what it means to perform forensic investigations as well as what an incident response might look like and why they are important. You may or may not choose to skim or skip this chapter, depending on how well-versed you are with some of the basic legal underpinnings and concepts of what forensics and incident response are.

Chapter 2 provides the foundation of what you should know about networking and protocols, because the rest of the book will be looking at network traffic in a lot of detail. If you are unfamiliar with networking and the protocols we use to communicate across a network, you should spend a fair amount of time here, getting used to how everything is put together.

Chapter 3 covers host-side artifacts. After all, not everything happens over the bare wire. Communication originates and terminates from end devices like computers, tablets, phones, and a variety of other devices. When communication happens between two devices, there are traces on those devices. We'll cover what those artifacts might be and how you might recover them.

Chapter 4 explains how you would go about capturing network traffic and then analyzing it.

Chapter 5 talks about the different types of attacks you may see on the network. Looking at these attacks relies on the material covered in Chapter 4, because we are going to look at packet captures and analyze them to look at the attack traffic.

Chapter 6 is about how a computer knows where it is and how you can determine where a computer is based on information that you have acquired over the network. You can track this down in a number of ways to varying levels of granularity without engaging Internet service providers.

Chapter 7 covers how you can prepare yourself for a network investigation. Once an incident happens, the network artifacts are gone because they are entirely ephemeral on the wire. If you are employed by or have a relationship with a business that you perform investigations for, you should think about what you need in place so that when an incident happens, you have something to look at. Otherwise you will be blind, deaf, and dumb.

Chapter 8 continues the idea of getting prepared by talking about intrusion detection systems and their role in a potential investigation.

Along the same lines, **Chapter 9** is about firewalls and other applications that may be used for collecting network-related information.

Chapter 10 covers how to correlate all of that information once you have it in order to obtain something that you can use. This includes the importance of timelines so you can see what happened and in what order.

Chapter 11 is about performing network scans so you can see what the attacker might see. Network scanning can also tell you things that looking at your different hosts may not tell you.

Finally, Chapter 12 is about other considerations. This includes cryptography and cloud computing and how they can impact a network forensic investigation.

Once you have a better understanding of all of the different types of network communications and all of the supporting information, I hope you will come away with a much better understanding of the importance of making use of the network for investigations. I hope you will find that your skills as a network investigator improve with what you find here.

Network Forensics

1

Introduction to Network Forensics

In this chapter, you will learn about:

- What network forensics is
- Evidence handling standards
- Verification of evidence

Sitting in front of his laptop he stares at a collection of files and reflects on how easy it was to get them. He sent an e-mail to a sales manager at his target company—almost silly how obviously fake it was—and within minutes he knew that he had access to the sales manager’s system. It took very little time for him to stage his next steps, which included installing a small rootkit to keep his actions from being noticed, and to ensure his continued presence on the system wouldn’t be detected. It also provided him continued access without the sales manager needing open the e-mail message again. That had taken place weeks back and so far, there appeared to be no evidence that anyone had caught on to his presence not only on the system but, by extension, on the business network the sales manager’s laptop was connected to.

It was this network that he was poring over now, looking at a collection of files related to the business’s financial planning. There were also spreadsheets including lists of customer names, contact information, and sales projections to those customers. No really big score but definitely some interesting starting points. Fortunately, this user was well-connected with privileges in the enterprise network. This ended up giving him a lot of network shares to choose from, and for the last several weeks he has been busy looking for other systems on the network to take over. Getting access to the address book on this system was really helpful. It allowed him to send messages looking as though they came from this user, sending co-workers to a website that would compromise their systems with some client software, adding them to the growing botnet he had control over. File shares were also good places to not only get documents to make use of, but also to drop some more infected files. The key loggers that were installed have generated some interesting information and keeping an eye on all of that is an ongoing project.

Ultimately, this is becoming quite a little stronghold of systems. It’s not exactly the best organization he’s been in with respect to quality data from an intellectual property or large caches of credit card numbers or even health care information. However, having more systems to continue building the botnet is always good

and at some point months or even years down the road, more interesting information may show up. In the meantime, there may be vendors who have trust relationships with this network that could be exploited.

Once inside the network, he has so many potential places to go and places to probe. There is a lot of data to be found and even though it appears that disk encryption is being used fairly consistently across the organization, all of that data is accessible to him as an authenticated user on the network. Wiping logs in places where they were turned on was trivial. This little network was all his for the taking for apparently as long as he felt it would be useful.

Does this sound scary at all to you? In reality, this is far too common and although it's dramatized, it's not that far off from how networks become compromised. Not long ago, technical intrusions were more common than the type of attack just described. In a *technical intrusion*, attackers use software vulnerabilities to get into a system remotely. This type of attack targets servers sitting in a data center because those are exposed to the outside world. That's not the case anymore. As we continue to learn, attackers are using people to get into systems and networks. This was vividly illustrated in 2013 in Mandiant's report, "APT1: Exposing One of China's Cyber Espionage Units" (<https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf>). Attackers send e-mail with malicious attachments, get someone to visit a website, or just simply park malicious software on a known website and wait for people to visit in order to infect their systems. Unfortunately, this is the world we now live in, a world where companies who haven't had systems compromised are becoming the minority rather than the majority.

This is one reason forensics is becoming such a hot skill to have. Well, that and the fact that the folks on various TV shows make it seem really cool, interesting, and easy. The reality is a different story, of course. Although the news and other media outlets make it seem as though attacks are carried out by solo *hackers* (an ambiguous and misleading word), the majority of outside attacks businesses are subject to today are perpetrated by well-funded and organized criminal enterprises. There is money to be made from these crimes; criminals are starting to use ransom and extortion to go directly for the money rather than trying to steal something to sell off later on.

The term *forensics* can be ambiguous. Because of that, it's helpful to have an understanding of what forensics currently is and isn't. Particularly when it comes to network forensics, it's more and more becoming part of incident response. Digital forensics practitioners have to be capable of more than locating images and deleted files that may be common for the large volume of child pornography cases that traditional law enforcement practitioners may be looking for. Sometimes, knowing how to extract files from a computer system isn't enough because information can be obscured and deleted very effectively. Certainly operating system forensics is important, but sometimes it takes more than just understanding what happened on the system itself.

Network forensics is becoming an extremely important set of skills when it comes to situations like the one described at the beginning of the chapter. Rather than relying on what the operating system and disks may be able to tell you, a network forensic investigator can go to the network itself and collect data of an attack in progress or look up historical information that may be available after

a company has suffered a security breach with someone taking up long-term residence, someone who has the ability to observe and piece together what they see into a coherent picture. This coherent picture may include information from other sources such as firewalls, application logs, antivirus logs, and a number of other sources.

One advantage to watching the network is that the network can't lie. Applications may not be doing what they are supposed to be doing. Logs may not be available or they may have been wiped. There may be root kits installed to obscure what is happening on a system. Once a network transmission is sent out on the wire, though, the bits are the bits.

Because of situations like the one described in the chapter-opening scenario, it's important to know exactly what forensics is as a practice as well as its role in incident response. Finally, there is a need for not only forensic practitioners in general because of the large number of incidents that occur in businesses around the world, but specifically, there is a need for network forensic practitioners.

What Is Forensics?

Before going further, let's define some terms.

The word *forensics* comes from the Latin *forens*, meaning belonging to the public. It is related to the word *forum*. If you have ever been involved in debate teams, you may be familiar with it as being related to debate and argumentation. If you are skilled in forensics, you may make a good lawyer. It is from this sense that the connotation of the word has come to mean something other than debate and argumentation. Investigating evidence, in the field or in the lab, to be used in a court case is the practice of forensics because the activity is related to the courts or trials.

This chapter expands on that by talking more specifically about *digital forensics*. Computer or *digital forensics* is the practice of investigating computers, digital media, and digital communications for potential artifacts. In this context, the word *artifact* indicates any object of interest. We wouldn't use the word *evidence* unless it's actually presented as part of a court case. You may say that an artifact is potential evidence. It may end up being nothing, but because it was extracted from the piles of data that may have been handed to the investigator, we need to refer to it in a way that makes clear the object is something of interest and potentially warrants additional investigation.

Because the word forensics is used in legal settings, you will often find that talk about forensics is involved with law enforcement. Traditionally, that has been the case. However, because many of the techniques and skills that are used by law enforcement are the same as those that may be practiced by an incident response specialist—someone who is investigating a suspicious event or set of events within a business setting—the word *forensics* also describes the process of identifying digital artifacts within a large collection of data, even in situations where law enforcement isn't involved.

For our purposes, the data we are talking about collecting is network information. This may be packet captures, which are bit-for-bit copies of all communication that has passed across a network

interface. The data collected may also come in the form of logs or aggregated data like network flow information.

Any time you handle information that could potentially be used in a court case, it's essential that it be maintained in its original condition, and that you can prove that it hasn't been tampered with. There are ways to ensure that you can demonstrate that the evidence hasn't been tampered with, including maintaining documentation demonstrating who handled it. Additionally, being able to have verifiable proof that the evidence you had at the end is the same as at the beginning is important. The reason for this is that in a court case, technical evidence, such as that from a digital forensic examination, is expected to adhere to an accepted set of standards.

Handling Evidence

The United States of America uses a *common law* legal system. This is at the federal as well as the state level, with the exception of the state of Louisiana, which uses a *civil law* system. The United Kingdom also uses a common law system. This means that legislatures enact laws and those laws are then interpreted by the courts for their applicability to specific circumstances. After a court has issued a ruling on a case, that case can then be used as a precedent in subsequent cases. This way every court doesn't have to make a wholly original interpretation of a law for every case. They build on previous cases to create a common interpretation of the law.

When it comes to addressing technical evidence in court cases, a couple of cases are worth understanding. The first case, *Frye vs. United States*, was a case in 1923 related to the admissibility of a polygraph test. As we continue to make technological advances, courts can have a hard time keeping up. The Frye standard was the one of the first attempts to codify a process that could help ensure that technical or scientific evidence being offered was standardized or accepted within the technical or scientific community. The courts needed a way to evaluate technical or scientific evidence to ensure that it was able to help the trier of facts determine the truth in a trial.

In essence, the Frye standard says that any scientific or technical evidence that is presented before the court must be generally accepted by a meaningful portion of the community of those responsible for the process, principle, or technique being presented. Acceptance by only a small number of colleagues who are also working in a related area doesn't necessarily rise to the standard of general acceptance by the community. Scientific evidence such as that resulting from DNA testing or blood type testing has passed this standard of reliability and veracity and is therefore allowed to be presented in a trial.

The federal court system and most U.S. states have moved past the Frye standard. Instead, they rely on the case *Daubert vs. Merrell Dow Pharmaceuticals, Inc.* Essentially, the standard of determining whether scientific or technical evidence is relevant hasn't changed substantially. What the majority opinion in the Daubert case argued was that because the Federal Rules of Evidence (FRE) were passed in 1975, those should supersede Frye, which was older. The Supreme Court ruled that in cases where the FRE was in conflict with common laws, such as the standard set by Frye, the FRE had precedence.

The intention of the continuing progress of case law related to technical evidence is to ensure that the evidence presented can be used to assist the trier of facts. The role of the trier of facts in a court case is to come to the truth of the situation. Frye was used to make sure technical evidence was accepted by a community of experts before it could be considered admissible in court. Daubert said that because the Federal Rules of Evidence came later than Frye, it should become the standard in cases of technical evidence. While expert witnesses are used to explain the evidence, the expert witness alone is not sufficient. The witness is a stand-in at trial for the evidence. A witness can be questioned and can provide clarifying information that the evidence directly cannot.

When it comes to digital evidence, we have to consider issues related to the appropriate handling of the data because it can be easily manipulated. For that reason, there's a risk that digital evidence could be considered hearsay if it's mishandled because of the FRE requirements regarding hearsay evidence. Hearsay is relevant here because *hearsay* is any evidence that is not direct, meaning that it doesn't come from a primary source that can be questioned by the opposition. In short, because there isn't someone sitting on the stand indicating what they saw, it's potentially hearsay unless it is a recording of regular business activities. Of course, the legal aspects are much more complicated than this short discussion might imply, but those are the essentials for those of us without law degrees.

All of this is to say that we have to handle potential evidence carefully so it cannot be questioned as being inauthentic and an inaccurate representation of the events. Fortunately, there are ways that we can not only demonstrate that nothing has changed but also demonstrate a complete record of who has handled the evidence. It is essential that when evidence has been acquired that it be documented clearly from the point of acquisition using the techniques outlined in the following sections.

Cryptographic Hashes

The best way to demonstrate that evidence has not changed from the point of acquisition is to use a cryptographic hash. Let's say, for example, that you have an image of a disk drive that you are going to investigate. Or, for our purposes, what may be more relevant is to say that we have a file that contains all of the network communications from a particular period of time. In order to have something we can check against later, we would generate a cryptographic hash of those files. The cryptographic hash is the result of a mathematical process that, when given a particular data set as input, generates a fixed-length value output. That fixed-length value can be verified later on with other hashes taken from the same evidence. Because hashing a file will always generate the same value (that is, output), as long as the file (the input data) hasn't changed, courts have accepted cryptographic hashes (of sufficient complexity) as a reliable test of authenticity when it comes to demonstrating that the evidence has not changed over a period of time and repeated interactions.

Two separate sets of data creating the same hash value is called a *collision*. The problem of determining the collision rate of a particular algorithm falls under a particular probability theory called the *birthday paradox*. The birthday paradox says that in order to get a 50% probability that two people in a given room have the same birthday, month and day, all you need is to have 23 people in the room. In order to get to 100% probability, you would need 367 people in the room. There is a very slim potential

for having 366 people in a room who all have a different birthday. To guarantee that you would have a duplicate, you would need to have 367 ($365 + 1$ for leap day + 1 to get the duplicate). This particular mathematical problem has the potential to open doors for attacks against the hash algorithm.

When you hear *cryptographic*, you may think *encryption*. We are not talking about encrypting the evidence. Instead, we are talking about passing the evidence through a very complicated mathematical function in order to get a single output value. Hashing algorithms used for this purpose are sometimes called *one-way functions* because there is no way to get the original data back from just the hash value. Similarly, for a hash algorithm to be acceptable for verifying integrity, there should be no way to have two files with different contents generate the same hash value. This means that we can be highly confident that if we have one hash value each time we test a file, the content of that file hasn't changed because it shouldn't be possible to make any change to the content of the file such that the original hash value is returned. The only way to get the original hash value is for the data to remain unaltered.

NOTE A cryptographic hash takes into consideration only the data that resides within the file. It does not use any of the metadata like the filename or dates. As a result, you can change the name of the file and the hash value for that file will remain the same.

NOTE Cryptography is really just about secret writing, which isn't necessarily the same as encryption. Hashes are used in encryption processes as a general rule because they are so good at determining whether something has changed. If you have encrypted something, you want to make sure it hasn't been tampered with in any fashion. You want to know that what you receive is exactly what was sent. The same is true when we are talking about forensic evidence.

For many years, the cryptographic hash standard used by most digital forensic practitioners and tools was Message Digest 5 (MD5). MD5 was created in 1992 and it generates a 128-bit value that is typically represented using hexadecimal numbering because it is shorter and more representative than other methods like printing out all 128 binary bits. To demonstrate the process of hashing, I placed the following text into a file:

Hi, this is some text. It is being placed in this file in order to get a hash value from the file.

The MD5 hash value for that file is 2583a3fab8faaba111a567b1e44c2fa4. No matter how many times I run the MD5 hash utility against that file, I will get the same value back. The MD5 hash algorithm is non-linear, however. This means that a change to the file of a single bit will yield an entirely different result, and not just a result that is one bit different from the original hash. Every bit in the file will make a difference to the calculation. If you have an extra space or an end of line where there wasn't one in the original input, the value will be different. To demonstrate this, changing the first letter of the text file from an H to a G is a single-bit difference in how it is stored on the computer since the value for H is 72 and the value for G is 71 on the ASCII table. The hash value resulting from this

altered file is 2a9739d833abe855112dc86f53780908. This is a substantive change, demonstrating the complexity of the hashing function.

NOTE MD5 is the algorithm but there are countless implementations of that algorithm. Every program that can generate an MD5 hash value contains an implementation of the MD5 algorithm.

One of the problems with the MD5 algorithm, though, is that it is only 128 bits. This isn't an especially large space in which to be generating values, leading it to be vulnerable to collisions. As a result, for many purposes, the MD5 hash has been superseded by the Secure Hash Algorithm 1 (SHA-1) hash. The SHA-1 hash generates a 160-bit value, which can be rendered using 40 hexadecimal digits. Even this isn't always considered large enough. As a result, the SHA-2 standard for cryptographic hashing has several alternatives that generate longer values. One that you may run into, particularly in the encryption space, is SHA-256, which generates a 256-bit value. Where the 128-bit MD5 hash algorithm has the potential to generate roughly 3.4×10^{38} unique values, the SHA-256 hash algorithm can yield 1.15×10^{77} unique values. It boggles the mind to think about how large those numbers are, frankly. Generating a SHA-1 hash against our original text file gives us a value of 286f55360324d42bcb1231ef5706a9774ed0969e. The SHA-256 hash value of our original file is 3ebcc1766a03b456517d10e315623b88bf41541595b5e9f60f8bd48e06bcb7ba. These are all different values that were generated against the same input file.

One thing to keep in mind is that any change at all to the data in the source file will generate a completely different value. Adding or removing a line break, for example, would constitute removing an entire character from the file. If that were done, the file may look identical to your eyes but the hash values would be completely different. To see the difference, you would have to view the file using something like a hexadecimal editor to see how it is truly represented in storage and not just how it is displayed.

You can use a number of utilities to generate these values. The preceding values were generated using the built-in, command-line utilities on a Mac OS system. Linux has similar command-line utilities available. On Microsoft Windows, you can download a number of programs, though Microsoft doesn't include any by default. Microsoft does, however, have a utility that you can download that will generate the different hash values for you. The name of the utility is File Checksum Identity Verifier (FCIV).

Any time you obtain a file such as a packet capture or a log file, you should immediately generate a hash value for that file. MD5 hash values are considered acceptable in court cases as of the time of this writing, though an investigation would be more durable if algorithms like SHA-1 or SHA-256, which generate longer values, were to be used. MD5 continues to demonstrate flaws the longer it is used and those flaws may eventually make evidence verification from MD5 hashes suspect in a court case.

Over the course of looking at packet captures in Chapter 4, we will talk about some other values that perform similar functions. One of those is the cyclic redundancy check (CRC), which is also mathematically computed and is often used to validate that data hasn't been altered. These sorts of values, though, are commonly called checksums rather than hashes.

Chain of Custody

Sometimes it seems as though TV shows like *NCIS*, *CSI*, *Bones*, and others that portray forensics simultaneously advance and set back the field of forensics. Although some of the technical aspects of forensics, including the language, are ridiculous, these shows do sometimes get things right. This was especially true in the early days of *NCIS*, as an example, where everything they collected was bagged and tagged. If evidence is handed off from one person to another, it must be documented. This documentation is the *chain of custody*. Evidence should be kept in a protected and locked location if you are going to be presenting any of it in court. Though this may be less necessary if you are involved in investigating an incident on a corporate network, it's still a good habit. For a start, as noted earlier in this chapter, you never know when the event you are investigating may turn from a localized incident to something where legal proceedings are required. As an example, the very first well-known distributed denial of service (DDoS) attack in February 2000 appeared as a number of separate incidents to the companies involved. However, when it came time to prosecute Michael Calce, known as Mafiaboy, the FBI would have needed evidence and that evidence would have come from the individual companies who were targets of the attacks—Yahoo, Dell, Amazon, and so on.

Even in the case of investigating a network incident in a business setting, documenting the chain of custody is a good strategy. This ensures that you know who was handling the potential evidence at any given time. It provides for accountability and a history. If anything were to go wrong at any point, including loss of or damage to the evidence, you would have a historical record of who was handling the evidence and why they had it.

Keeping a record of the date and time for handing off the evidence as well as who is taking responsibility for it and what they intend to do with it is a good chain-of-custody plan. It doesn't take a lot of time and it can be very important. As always, planning can be the key to success, just as lack of planning can be the doorway to failure. The first time you lose a disk drive or have it corrupted and that drive had been handed around to multiple people, you will recognize the importance of audit logs like chain-of-custody documentation. Ideally, you would perform a hash when you first obtain the evidence to ensure that what you are getting is exactly what you expect it to be. You should have a hash value documented so you will have something to compare your hash to in order to demonstrate that no changes have occurred.

Incident Response

Incident response may be harder to get your head around if you are a forensic practitioner. If you are a system or network administrator trying to get your hands around the idea of forensics, incident response should be old hat to you. When networks belonging to businesses or other organizations (schools, non-profits, governmental, and so on) are subject to a malware infestation, as an example, that would probably trigger an incident response team to get the incident under control as well as

investigate the cause of the incident. Depending on who you talk to you may get different answers, but the process of incident response can be boiled down to four stages: preparation; detection and analysis; containment, eradication, and recovery; and post-incident activity.

What exactly is an incident? How does an incident differ from an event? This is another area where you may find that you get differing opinions depending on whom you talk to. Rather than getting into a deep discussion here, let's go with simple. An *event* is a change that has been detected in a system. This could be something as simple as plugging an external drive into a system. That will trigger a log message in most cases. That would be an event. Someone attempting to ping a system behind a firewall where the messages are blocked and logged may be an event. An event may even be updating system software, as in the case with a hot fix or a service pack.

An *incident*, on the other hand, is commonly something that is attributable to human interaction and is often malicious. An incident is always an event, because every incident would result in some sort of observable change to the system. If all of your web servers were infected by malware, that malware would be observable on the system. It would result in events on all of the systems and you would have an incident on your hands. A single system being infected with malware would be an event but wouldn't be enough to rise to a level where you would call an incident response team.

A forensic practitioner would obviously be necessary at the detection and analysis phase but they would typically be involved in the preparation stage as well. Over the course of the book, we will be going over some items that you may want to make sure are in place as an organization goes through preparation stages. Preparation is a very large category of activities, including planning, but from the standpoint of a forensic investigator, it is primarily when you make sure you will have what you need when it comes to doing an analysis. There may also be activity when it comes to eradication, to ensure that the source of the incident has been completely removed. Finally, a forensic investigator would be involved in post-incident activities for lessons learned and process improvement.

In most cases, you would have an incident response team, even if it is small and ad hoc, to deal with incidents because handling incidents is a process. The larger the organization and the more systems involved, the larger the incident response team would likely be. Creating a team up front would be another important activity when it comes to planning. Your organization, as part of the creation of security policies, standards, and processes, should create an incident response team or at least have documentation for how to handle an incident, should one occur. Considering that it's widely believed that a significant proportion of companies in the United States have been breached, meaning they have had attackers compromise systems to gain unauthorized access, "should one occur" is a bit euphemistic. In reality, I should say *when* an incident occurs. If you haven't had to deal with an incident, it may simply be a result of lack of appropriate detection capabilities.

Forensic practitioners are definitely needed as part of the incident response effort. They need not be full-time forensic practitioners, but simply people already employed at the company who happen to have the knowledge and skills necessary to perform a forensic investigation. They can get to the root cause of an incident, and that requires someone who can dig through filesystems and logs and look in other places within the operating system on the affected hosts.

Without understanding the root cause, it would be difficult to say whether the incident is under control. It would also be difficult to know whether you have found all of the systems that may be impacted because incidents, like unauthorized system access or malware infestations, will commonly impact multiple devices across a network. This is especially true when there is a large commonality in system deployments. In other words, if all systems are created from the same source image, they will all be vulnerable in the same way. Once an attacker finds a way into one, all of the others that have been built using the same image are easy targets.

The forensic investigator will need to be focused on identifying the source of the attack, whether it's a system compromise or a malware infection, to determine what may need to be addressed to make sure a subsequent, similar attack isn't successful. They will also need to be focused on finding any evidence that the attacker attempted to compromise or infect other hosts on the local network. If there is evidence of attempts against systems not on the organization's network, the incident response team should have the capability to reach out to other organizations, including a computer emergency response team (CERT) that may be able to coordinate attacks across multiple organizations.

This is where you may run into the need for the collected artifacts in a larger investigation and potential criminal action. Coordinating with law enforcement will help you, as a forensic investigator, determine your best course of action if there is evidence of either substantial damage or evidence that the attack involves multiple organizations. This is another area where planning is helpful—determining points of contact for local and federal law enforcement ahead of time for when an incident occurs.

The Need for Network Forensic Practitioners

In early 2016, a task force was assembled to talk about how to best approach educating more professionals who are capable of filling thousands of jobs that are expected to be available in the coming years. While this is generally referred to as a need for cybersecurity workers, the term *cybersecurity* is fairly vague and covers a significant amount of ground. The federal government alone is planning on large spending around making sure they can support a growing need for skilled and/or knowledgeable people to prevent attacks, defend against attacks, and then respond when an attack has been detected. The initial plan was to spend \$3.1 billion to modernize and if the plan is implemented properly, there will continue to be a need for people who are capable of responding to incidents.

This is just at the level of the federal government. Large consulting companies like Mandiant and Verizon Business as well as the large accounting companies that are also involved in security consulting are hiring a lot of people who have skills or knowledge in the area of forensics. When companies suffer a large-scale incident, particularly smaller or medium-sized companies that can't afford full-time staff capable of handling a complete response, they often bring in a third party to help them out. This has several advantages. One of them is that a third party is less likely to make any assumptions because they have no pre-existing knowledge of the organization. This allows them to be thorough rather than potentially skipping something in the belief they know the answer because of the way "it's supposed to work." Hiring information technology people who are skilled in information security

and forensics can be really expensive. This is especially true for smaller companies that may just need someone who knows a little networking and some Windows administration.

Large companies will often have a staff of people who are responsible for investigations, including those related to digital evidence. This means that the federal government, consulting companies, and large companies are all looking for you, should you be interested in taking on work as a network forensic investigator. This will be challenging work, however, because in addition to an understanding of common forensic procedure and evidence handling, you also need a solid understanding of networking. This includes the TCP/IP suite of protocols as well as a number of application protocols. It also includes an understanding of some of the security technology that is commonly in place in enterprise networks like firewalls and intrusion detection systems.

Because there is currently no end in sight when it comes to computers being compromised by attackers around the world, there is no end in sight for the need for skilled forensics professionals. For forensic investigators without a foundation in network protocols and security technologies, this book intends to address that gap.

Summary

Businesses, government agencies, educational institutions, and non-profits are all subject to attack by skilled adversaries. These adversaries are, more and more, well-funded professional organizations. They may be some form of organized crime or they may be nation-states. The objectives of these two types of organizations may be significantly different but the end result is the same—they obtain some sort of unauthorized access to systems and once they are in place, they can be difficult to detect or extricate. This is where forensics professionals come in.

Forensics is a wide and varied field that has its basis in the legal world. Forensics, in a general sense, is anything to do with court proceedings. For our purposes, while the practice of digital forensics may have some foundation in law enforcement professionals performing investigations as part of criminal proceedings, the skills necessary to perform those investigations cross over to other areas. When it comes to investigations performed within an enterprise rather than by a law enforcement agency, the skills and techniques are the same but there may be differences in how artifacts and evidence are handled. That isn't always the case, of course, because even if you are just looking for the root cause, there is a possibility of what you find being necessary as part of a court case.

Because there is a possibility that artifacts and evidence may be used in court, it's generally a good idea to make use of cryptographic hashes as well as keeping a chain-of-custody document. These two activities will help you maintain accountability and a historical record of how the evidence and artifacts were handled. This is helpful if you have to refer to the events later on.

When it comes to working in an organization that isn't law enforcement, you may be asked to perform forensic investigations as part of an incident response. Incident response teams are becoming common practice at all sizes of organization. It's just how any organization has to operate to ensure that they can get back on their feet quickly and efficiently when an attack happens—whether it's

someone who has infiltrated the network by sending an infected e-mail or whether it's an attacker who has broken into the web server through a commonly known vulnerability.

Given the number of organizations around the world that have suffered these attacks, including several highly publicized attacks at Sony, Target, Home Depot, TJ Maxx, and countless others, there is a real need for forensics practitioners who can work with network data. This is because companies are using intrusion detection systems that will generate packet captures surrounding an incident and some organizations will actually perform a wire recording on a continuous basis simply in case an incident takes place. The network is the best place to capture what really happened because the network—the actual wire—can't lie.

References

Morgan, Steve. "Help Wanted: 1,000 Cybersecurity Jobs At OPM, Post-Hack Hiring Approved By DHS." (*Forbes*, January 13, 2016.) Retrieved June 22, 2016, from <http://www.forbes.com/sites/stevemorgan/2016/01/31/help-wanted-1000-cybersecurity-jobs-at-opm-post-hack-hiring-approved-by-dhs/#3f10bfe12cd2>.

Umberg, Tommy and Cherrie Warden. "Digital Evidence and Investigatory Protocols." *Digital Evidence and Electronic Signature Law Review*, 11 (2014). DEESLR, 11(0). doi:10.14296/deeslr.v11i0.2131.

2

Networking Basics

In this chapter, you will learn about:

- What protocols are and how they work
- The basics of TCP/IP
- The difference between the OSI model and the TCP/IP architecture

Sitting at his desk, he was looking for his next target. A couple of quick Google searches and digging through various job sites gave him some ideas but he needed to know more. He was in need of addresses and hostnames and he knew of several places he would be able to locate that information. With just a few commands in his open terminal window he had a number of network addresses that he could start poking at. That gave him a starting point, and a few DNS queries later he had not only network addresses but some hostnames that went along with them. He was also able to get some contact information that could be useful later on.

Once he had his hostnames and addresses, he could figure out what programs may be listening on the ports that were open at those addresses. He knew that the application layer was where the money was—all of the problems lower down in the stack had long since been corrected, so the best way into a system was going to be through any program that was sitting behind one of those open ports. Once he knew what applications he needed to target, he would be golden and he could make his move. There was so much that he might be able to do with a poorly implemented web application environment, for example. He could just see his bank account growing with all of the credit cards and other information he may be able to steal.

I wouldn't be doing much of a job of talking about network forensics without going over the basics of networking protocols and where all of the important information about the Internet and all of the networks attached to it is stored. The people who are attacking networks know at least enough to make their way around the Internet and local networks so forensics investigators need to know at least as much as the adversaries do in order to determine what they are doing. Even if the adversary is a piece of malware or someone internal to the company, you'll need to understand how it got to the system and interacted with the applications there.

We're going to start by talking about what a protocol is. In the course of going deeper into analysis, we'll be talking about protocols a lot so it's important to have a foundation on which to build those later conversations. When we are talking about networking, the different protocols are sometimes best thought about in layers, and that's actually how you will see them represented. There are two conceptual ideas for thinking about the layers of network protocols. One of them is the Open Systems Interconnect (OSI) model, which describes seven layers in its stack. The other is the Transmission Control Protocol/Internet Protocol (TCP/IP) suite, which has only four layers and evolved into a model after it had finally stabilized in its implementation.

The Internet protocols associated with the Advanced Research Projects Agency (ARPA) and later the Internet Engineering Task Force (IETF) have, almost since the very beginning, been created in an open, collaborative manner. As such, they start as documents that are called requests for comments (RFCs). Understanding these documents can be very useful. If there is ever a question about what you are looking at in practice, you can refer back to the original documentation to look up details about the protocols and standards to see what it is expected to look like.

The Internet is collaborative because it's a global entity, and as a result a number of interested parties want a say in how it's managed. As a global network, information related to networks and domains is stored a number of places. Knowing where the information is stored and how you can look up that information will provide essential information during the course of an investigation. Once we are done here, you will have a better understanding of how all of the information is stored and where you can get at it.

Protocols

To explain what a protocol is, we're going to step out of the world of networking and technology altogether. I can't help but think of the Goldie Hawn movie *Protocol* when thinking about this topic, and though that may be dating me somewhat, it's relevant. In the movie, Goldie Hawn plays a waitress who saves the life of an Arab dignitary and ends up with a job in the State Department working in Middle East affairs. You may be wondering why the movie is called *Protocol* and why this has anything at all to do with networking. A protocol is a standard of communication. In order to have productive conversations between two parties, we need protocols. This is especially true when you are talking about entirely different cultures, as in the Arabic countries and the United States. For the conversation and any negotiations to go smoothly, they rely on protocols—standards of behavior and communication that both parties adhere to so nothing is misunderstood.

When you think about it, the same is true in the networking world. For two systems, especially ones that speak entirely different languages, as might be the case with a Linux system trying to communicate to a Windows system, there must be standards of behavior and communication. In the early days of the Internet, back when it was still called the Arpanet in the late '60s and early '70s, many more operating systems were around than might seem to be the case today. Although there still are many, once you start factoring in larger systems, the day-to-day experience of the vast majority of people is with three operating systems: Windows, macOS, and Linux. Two of those come from

the same root operating system—Unix. However, they have just enough differences even today that protocols are important to make sure every conversation takes place smoothly.

Most of the time, when there is a conversation about protocols, you will hear someone refer to *layers*. This is because protocols are generally placed into stacks to explain how they relate to one another. Every type of communication on a network will involve multiple protocols across multiple layers, though each protocol is generally only aware of its own layer. There is one exception to that, but we'll get to it later in this chapter. Network protocols are mapped into two stacks. One is a generic model, and the other is a description of a set of protocols specifically designed to work together. Even the TCP/IP protocols can be mapped into the generic model, however.

Regardless of which way you think about the protocols, one important factor to keep in mind is that every layer only ever talks to its own layer on the other side. If you think about writing someone a letter, you can conceive of how this operates. You write a letter, you put it in an envelope, seal the envelope, address it, put a stamp on it, and then put it in the mailbox. For every action you put into pulling the letter together, there is a corresponding action on the receiving end. Your post office on the sending end determines how the envelope should get to the recipient by looking at the ZIP code. The sending post office has no interest in anything inside the envelope and really doesn't have any interest in the street address or the name of the recipient.

Let's say that the letter you are sending is to someone at a business. The address you have placed on the envelope is for the business. Once the envelope reaches the destination post office (the one that owns the ZIP code), the postal workers there have to look at the street address in order to determine which truck to put it on for delivery. The person driving the truck and out delivering the mail doesn't look at the ZIP code because it's irrelevant—the truck only delivers to a single ZIP code. Likewise, the name on the envelope is also irrelevant; the only important part is the street address. Once it gets to the business and lands in the mail room or with the receptionist, or whoever gets the mail when it arrives, that person will look at the name on the envelope and deliver it. The recipient then gets the letter, opens it, and reads the contents.

The same is true when we talk about protocol stacks. At every point during the process of sending and receiving, there is a specific piece of information that is intended for and handled by a specific person or target. The ZIP code tells the sending post office how to get to the destination. The street address tells the receiving post office how to get to the destination. The name on the envelope tells the receiving party who the letter is actually destined for, and in the end, the letter is probably only meaningful in any way to the recipient. None of these parties has much interest in looking at the other information because it doesn't help them to do their job. Certainly, each party can see the rest of the information (except, perhaps, the contents of the letter), but they only focus on the information they actually need. You will see this repeated over and over as we start talking about the different protocol stacks and then the specific protocols from the TCP/IP suite of protocols.

An essential concept that you should understand before we get started is *encapsulation*. Regardless of which communications stack you are referring to, data passes from one layer to another. Each layer distinguishes itself by applying some data associated to that layer before passing it on to the next layer down. This process is called encapsulation. Going back to our mail example, the letter is encapsulated inside the envelope and then the person's name is added to the envelope. After that,

the street address and then finally the ZIP code (since the city/town and state are just the long form of the ZIP, they are redundant) are added. This addressing information encapsulates the information that comes before, though in a less obvious way than you will get from the IP addresses and other forms of address discussed below.

On the receiving end, the communication goes through de-encapsulation by removing the headers that were added on the sending end before the data is sent to the next layer up the stack. You will see this process of encapsulation as we start talking about the two different models and then, more concretely, when we start looking at the different protocols in operation.

Open Systems Interconnection (OSI) Model

In the 1970s, a number of communication protocols including the nascent TCP were used on the Arpanet as well as System Network Architecture (SNA) from IBM, DECnet from Digital Equipment Corporation, and many others. The International Organization for Standardization (ISO) decided a single model was needed to fit all communication protocols. In 1977, the ISO made use of work done by the Honeywell Corporation to create an abstract model describing different functions used in communications systems. By 1983, it had merged its standard with a similar standard by the International Telephone and Telegraph Consultative Committee to create the current Open Systems Interconnection (OSI) model.

NOTE The acronym “ISO” is a compromise, recognizing the different abbreviations across the three languages used within ISO and is based on the Greek *isos*, meaning equal.

The OSI model consists of seven separate and distinct layers, each describing a particular set of functions and behaviors. Although every protocol used for communication will fit into one of these seven layers, not all communication streams will make use of all seven layers. Some types of communication are far more simplistic than others and may not need some of the higher layers of the protocol stack, depending on the intention of the communication. You can see a representation of the OSI model, drawn as a stack of boxes, in Figure 2-1.

We will go through the model from the bottom to the top, as though we were reading a message off the wire. At the very bottom of the stack, at layer 1, is the physical layer. The physical layer includes all of the tangible components that you can touch—cabling, network interfaces, and the actual signaling medium, whether it’s light or electrical. Since the name is pretty straightforward and descriptive, this one will be the easiest to remember and keep straight.

The next one up is the data link layer, layer 2. The data link layer is how systems on the same physical network communicate.

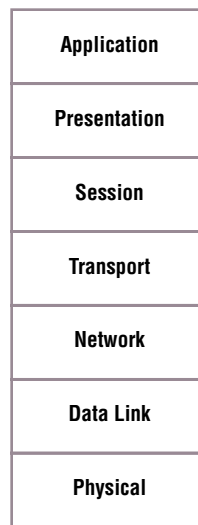


Figure 2-1: The Open Systems Interconnection seven layer model.

For every layer in the stack, there is generally a way to differentiate communication streams—a way of addressing. At layer 2, this is the Media Access Control (MAC) address. The MAC address is attached directly to the network interface, which is why it is sometimes called the physical interface. The data link layer makes sure that devices on the same physical network can communicate reliably with one another. If you are using a switch on your network, the switch is operating at layer 2 because it makes use of the MAC address to determine where to send network messages.

NOTE The MAC address is six bytes and it is expected to be globally unique, meaning no other network interface in the world will have the same MAC address as the network interface on your system. Those six bytes are broken into two separate sections, three bytes per section. The first half, 24 bits, is the *organizationally unique identifier* (OUI) that identifies the vendor of the network interface. The second half is the identifier for the interface itself. The OUI is something that can be looked up in one of several online databases so if you have the OUI, you can know the vendor of the interface.

The third layer is the network layer. Layer 3 makes sure that devices that are not on the same physical layer can communicate. Layer 3 messages typically require a router to pass messages from one network to another. This layer also requires an address. The Internet Protocol (IP) and the Internet Packet Exchange (IPX) protocol from Novell both operate at layer 3, providing network addresses, as well as addresses for the hosts on those networks.

Layer 4 is the transport layer. Where previous layers were about getting messages to the host, this is the first layer where the message has fully arrived at the host. Layer 4 allows for multiplexing of network communications on a single host. It does this by using ports. Each network address may have a large number of ports to communicate to. Systems that use the TCP/IP protocols will have 65,536 ports to communicate to on the different transport protocols. The User Datagram Protocol, the Transmission Control Protocol, and the Sequenced Packet Exchange Protocol (SPX) are all at this layer.

Layer 5 is the session layer. While the transport layer can support a connected form of communication between two systems, that is strictly system to system. Layer 5 is where the communication stream between those two hosts is managed. Depending on the implementation and the protocols being used, you may only have one-way traffic or you may have bi-directional traffic. The session layer determines how that communication will happen. The protocols at this layer handle the negotiation of the communication flow. Telnet, Secure Shell (SSH), and the File Transfer Protocol (FTP) are at this layer, though they also are commonly said to live at the application layer as well. Many session layer protocols straddle multiple layers.

Layer 6 is the presentation layer. This layer handles the conversion between the network communication and the application. Any data encoding and decoding as well as data formatting would be done at this layer. JPEG and GIF files are at this layer. The Hypertext Transport Protocol (HTTP) is also at this layer. Anything that does encryption/decryption or compression would be at the presentation layer.

Finally, layer 7 is the application layer. Any application programming interfaces (APIs) would exist at this layer. This is where the interface to the user is.

TCP/IP Protocol Suite

The TCP/IP protocol suite was developed over a number of years and evolved into what we have today. While it is sometimes referred to as a model, the TCP/IP protocol suite is a description of an as-built set of protocols designed to work together. The communication protocols on the Arpanet were developed as they were determined to be necessary rather than planned well ahead of time. For instance, initially there was no Internet Protocol (IP). The Internet Protocol was part of the Transmission Control Program and offered connectionless service between two systems. If the two systems wanted the communication to be connection-oriented and have the connection managed by the Transmission Control Program rather than a higher-layer application, it would use the Transmission Control Protocol (TCP). Eventually, IP was separated out to handle network addressing and other network functions. On top of that, other protocols were developed. So, the TCP/IP architecture or model is documentation of what is in place.

NOTE The TCP/IP protocol suite is sometimes referred to as the Department of Defense (DoD) model, because the DoD provided funding for the Arpanet, where TCP and IP were developed.

Whereas the OSI model is seven layers, TCP/IP, or the Internet Protocol suite, is only four layers. While it is much simplified over the OSI model, you will see that all of the same functions are described within the four layers. Even though the Internet uses the Internet Protocol suite to operate, it's more common in my experience at Internet service providers and network equipment vendors for networking professionals to refer to the layers of the OSI model, partly because of the granularity it offers, which helps to differentiate the functionality being referred to.

The first layer of TCP/IP is the Link layer. This encompasses functionality from the first two layers of the OSI model. Both the physical and the data link layer of the OSI model are represented in this layer, so the same functionality and examples from those layers apply here. This is where the MAC address lives and this layer makes sure that systems on the same physical network can communicate with one another.

The second layer is the Internet layer. This is the same as the network layer in the OSI model. This is where IP lives. IP provides network addressing and helps to ensure that messages can get from one network to another. IP is a routable protocol, though not all network layer protocols are. Of course, every host on a network gets its own address, so talking about network addressing is incomplete. The important distinction, though, is that the bulk of any IP address is the network address. The smallest portion is the actual host component. This reflects the large number of networks that are connected together across the Internet where the number of hosts on any given network is comparatively much smaller.

The third layer is the Transport layer, corresponding to layer 4 in the OSI model. It shares the same name between the OSI model and the TCP/IP model. This is where multiplexing on each system happens, through the use of ports. Ports provide a way for multiple applications to listen simultaneously on the same IP address as well as for multiple applications to originate traffic using separate source ports, allowing return traffic to get back to the correct application.

Finally, the fourth and last layer in the TCP/IP model is the Application layer. While it shares the same name as layer 7 in the OSI model, it encompasses all of the functions of layers 5–7 of the OSI model. Applications reside here. If they need presentation functions or session management, the applications take care of all of that and those functions aren't broken out and described separately from the application itself.

As you can see, the TCP/IP model is quite a bit simpler to think about than the OSI model. If you want to get fine-grained about functionality, though, the OSI model is better as a reference point. Ultimately, they are both just for conceptualizing and referring to the functions without specific reference to the protocols in use.

Protocol Data Units

We've talked about the various layers of the two communication models. Ultimately, the purpose for those models is to build different means for multiple systems to communicate with one another. The protocols don't exist for the purpose of the protocols. They exist to be able to effectively and efficiently send data from one system to another. The data is wrapped up with the different headers from each layer that allow the receiving system to identify where the data is headed, including what application.

As different protocols add their headers, encapsulating the data that is already there, the result is a different chunk of data than what was there before the protocol got its say. The resulting chunk of data, just as the chunk of data that started out, is called a protocol data unit (PDU). Each layer of the communications stack has a different protocol data unit associated with it. This means that at most layers, we use a different word to describe the chunk of data, or protocol data unit, we are looking at.

In order to talk about the different words, we are going to start at the very top of the stack. This is because when a message is being prepared for sending, it starts at the application. The application creates data. The protocol data unit at the application layer is just "data." As we move down through the presentation and session layers, we are still talking about just data. You may not actually be working with protocols in layers 5–7, so there isn't really a PDU associated with it. It's just the data until we get to layer 4 of the OSI model.

Once we get to the transport layer, whether we are talking about the OSI model or the TCP/IP model, we are talking about the data that has the transport headers stacked on top. After those headers, which include the source and destination port numbers, are in place, you have a *segment* if you are using TCP and a *datagram* if you are using the User Datagram Protocol (UDP). The segment or datagram is then handed to IP to add some additional headers.

The IP headers include the source and destination IP address as well as some additional information, including indications as to whether what we have is just a fragment of a larger communication stream or just an individual message. Once the IP headers are on and sitting atop the TCP or UDP headers, you have a *packet*. A few protocols may be in use at layer 2, including Ethernet, Asynchronous Transfer Mode (ATM), Point to Point Protocol (PPP), or 802.11 (WiFi). No matter what the protocol is, there will be a set of headers that includes the source and destination MAC addresses. Once the layer 2 headers are on, you have a *frame*. The frame is what is placed onto the network.

Once the frame is converted to the right signaling mechanism, either an optical signal or an electrical signal, we are looking at bits. In the end, no matter what data you are sending, it is sent a bit at a time. If you are looking at the data as it is passing across the network, you are looking at a stream of bits. Later on, we'll look at more details of the different protocols you will see as we start pulling these messages—frames, packets, segments, and datagrams—apart.

Request for Comments

The very first request for comment (RFC) was written in 1969 by Steve Crocker. Crocker created RFCs and not only wrote the first, but wrote several others as well over the years. RFCs make available on the Internet the best possible technical description of protocols and processes. In 1969, the Advanced Research Projects Agency (ARPA) awarded a contract to Bolt, Beranek, and Newman (BBN) to design and build a network that was capable of including hosts from around the country. The idea was to connect research facilities at universities and government agencies in order to facilitate collaboration and allow for more efficient use of limited computing resources. At the time, computers were very large and very expensive, so being able to network the computers that did exist allowed for research to be conducted across the country without having to necessarily duplicate computing resources.

BBN had to design and build the very first system that was capable of moving packets from one system to another over the telephony network that was in place at the time. Initially, the device used to create the network was called the Interface Message Processor (IMP). You may think of it as a router, considering what it does. Such devices simply didn't exist, though, so the functionality of a router was handled in a specialized interface built into a Honeywell computer with software designed to move messages from the computer on site to the network, on its way to the destination IMP. The very first RFC specified the software that was to run on the IMP. Just as a point of history and also to give you a sense of what an RFC looks like, you can see the very first part of the very first RFC in Figure 2-2.

In addition to historical curiosities, RFCs provide detailed design documentation for processes and protocols. Of course, there is also the occasional joke, like the periodic April Fool's Day RFC that introduces protocols like the transmission of IP datagrams over avian carrier. This RFC was issued in 1990 and was inspired by a scene from the movie *Monty Python and the Holy Grail*. Much more to the point, though, if you are ever interested in knowing how a particular protocol like TCP, IP, UDP, HTTP, or any of hundreds of other protocols, enhancements and processes, works, you can get the last word by reading the RFC.

```

Network Working Group                                Steve Crocker
Request for Comments: 1                             UCLA
                                                    7 April 1969

                Title:  Host Software
                Author:  Steve Crocker
                Installation:  UCLA
                Date:    7 April 1969
    Network Working Group Request for Comment:  1

CONTENTS
INTRODUCTION
    I.  A Summary of the IMP Software
        Messages
        Links
        IMP Transmission and Error Checking
        Open Questions on the IMP Software

```

Figure 2-2: The top of RFC 1.

Currently, the Internet Engineering Task Force (IETF) manages all RFCs and it's responsible for providing oversight of the various working groups and discussion groups that work on creating the protocols and standards that the Internet operates using. It's worth noting, perhaps, that all of the various protocols in use are open and entirely voluntary. Any vendors choosing to implement a protocol or standard developed by the IETF veers from the written description at their peril. Not because anyone from the IETF is going to come knock their door down and wag a finger at them telling them they did something wrong. Instead, any vendor that implements a protocol or standard at variance with how it is written runs the risk of simply not functioning with other devices that do adhere more closely to the written specifications. Most hardware manufacturers don't have the luxury of developing products that only work with their own products, but some have certainly tried.

There is, though, quite a bit of latitude in most of the protocols. As you might imagine, the documentation can run to dozens of pages in some cases but not every aspect of a protocol is required in order to be considered in compliance. Words and phrases are used in RFCs to indicate whether a particular set of functionality is required or if it's just a "nice to have" feature. In fact, an entire RFC was written to describe the level of requirement. RFC 2119 was written in 1997 and is flagged as a Best Current Practice, another category of RFCs. The word "must" (as well as its counterpart "must not") indicates a hard requirement. In other words, don't follow this at your peril, lest you be considered not in compliance and not entirely functional. The word "should" (and its counterpart "should not") means that aspect or feature is recommended. Finally, the word "may (and "may not") indicates something that is optional.

The language of the RFCs is usually specific, though it's not always clear and often leaves the door open to interpretation. This is especially the case when actual implementation details are left out. If a random number is called for, as is the case in some of the lower-layer protocols like IP and TCP, the

RFCs don't specify how that random number is to be generated. This means that some vendors will choose to implement one way while others will implement in a completely different way. Despite the intentions of the engineers who write these documents, the challenge with English or any language is the multitude of ways to interpret a collection of words. Where you may read a passage in the TCP RFC one way, I may read it slightly differently. This may be similar to case law, as discussed in the previous chapter. Different people will read the RFC in different ways. Over time, there comes to be a generally understood way that things are to work.

For a long time, one man managed the RFCs. Jon Postel was the editor of the RFCs and the man who assigned Internet addresses up until his death in 1998. Of course, the Internet has grown substantially since that time and it now takes a much larger organization to take on the work he did for years. Literally thousands more RFCs have been written in the intervening years. Not all RFCs actually become anything. The idea of an RFC is to throw something out into the community of engineers and professionals working on the Internet infrastructure who then comment on it (it is a request for comments, after all). The document remains as an RFC until such time as it becomes a standard, which may never happen. Many widely used protocols have remained RFCs. This may be because they remain works in progress that are continuing to be altered, amended, and revised. However, the original RFC does not usually get revised; instead, another RFC is written to document changes to the original and a note may be added that it supersedes or makes obsolete the older document, if that's the case.

One example of this altered functionality through successive RFCs is for the Session Initiation Protocol (SIP), which is a protocol used primarily for Voice Over IP (VOIP). SIP itself has gone through two RFCs for the two versions that have been released. In addition, though, it has a large number of extensions. In fact, so many extensions have been written for SIP that in 2009, RFC 5411 was written as an Informational RFC to document all of the extension RFCs that have been written for SIP. In that case, you have the primary RFC, RFC 3261, which documented the second version of SIP, as well as a large number of extensions and then a whole RFC just to provide an index and glossary of all of those extensions.

All of the RFCs are public documents, so you can read up on the various protocols to see what the documentation says about how they are supposed to work. If you go to the tools site at IETF (tools.ietf.org), you can search for RFCs as well as get the complete list of all of the available RFCs, including much of the original documentation for how the Arpanet was to have worked, if you are interested in seeing the evolution of the various protocols and processes across the Internet.

To save you some trouble reading through all of the various RFCs, which can be difficult reading unless you have a particular taste for it, we will go over some of the finer points of the protocols you are likely to run into just so you are familiar with how they are meant to work in general, as well as what you should expect to see once we start actually looking at packets in great detail later on.

Internet Registries

The Internet runs on numerical addresses, which we get into in the “Internet Protocol Addresses” section later in this chapter. The problem with this is that humans are simply lousy at remembering numbers without context. Instead, we like to use names. Using names requires organizing related names together, usually grouped into domains, and each domain has an owner. Additionally, the addresses aren’t handed out randomly. They are assigned to owners and the owners need to be tracked to make sure that any changes that need to be made come from the right people. Domains are managed the same way. All of this is to say that a fair amount of information needs to be tracked and that information is, much like the Internet itself, scattered in repositories around the world.

At the very top level is the Internet Corporation for Assigned Names and Numbers (ICANN). Previously, the job of handing out addresses was handled by one man—Jon Postel. However, he passed away at about the same time that the number of networks was beginning to increase at a significant rate. The time had come to create an entity that could handle not only handing out addresses but also administering all of the names. ICANN owns the entire Internet address space, ultimately, but it hands out blocks to other organizations that then assign them to Internet service providers or companies that have need for large address spaces. These organizations that are the tier below ICANN are called Regional Internet Registries (RIRs). They are responsible for handling regional management of addressing and numbers associated with routing traffic on the Internet.

In North America, the RIR is the American Registry for Internet Numbers (ARIN). ARIN not only takes care of the United States and Canada but also parts of the Caribbean and Antarctica. Africa’s RIR is the African Network Information Center (AfriNIC). Asia and parts of the Pacific Ocean are handled by the Asia-Pacific Network Information Center (APNIC). APNIC takes care of the countries in Asia as well as New Zealand, Australia, and other neighboring countries. The Latin American and Caribbean Network Information Center (LACNIC) handles Latin America and the parts of the Caribbean that aren’t handled by ARIN. Finally, the *Reséaux IP Européens* Network Coordination Centre (RIPE) takes care of Europe, Russia, the Middle East, and Central Asia.

If you need to know who owns a particular block of IP addresses, you can check with these RIRs. This is typically done using the *whois* utility, which is available as part of some operating systems like Linux or macOS. You can also do these lookups using various web interfaces. As an example, you can see the result of a *whois* using the website GeekTools shown in Figure 2-3. This particular lookup was on a common IP address, 4.2.2.1. This is for one of the servers that are used around the world by Level 3 Communications to translate IP addresses to names and vice versa. The results indicate that this particular address is part of a much larger block. This particular block was inherited when Level 3 acquired Genuity. Genuity was a direct descendent of BBN by way of some acquisitions and spin offs.

```
#
# The following results may also be obtained via:
# https://whois.arin.net/rest/nets;q=4.2.2.1?
# showDetails=true&showARIN=false&showNonArinTopLevelNet=false&ext=netref2
#

NetRange: 4.0.0.0 - 4.255.255.255
CIDR: 4.0.0.0/8
NetName: LVL-ORG-4-8
NetHandle: NET-4-0-0-0-1
Parent: ()
NetType: Direct Allocation
OriginAS:
Organization: Level 3 Communications, Inc. (LVL)
RegDate: 1992-12-01
Updated: 2012-02-24
Ref: https://whois.arin.net/rest/net/NET-4-0-0-0-1

OrgName: Level 3 Communications, Inc.
OrgId: LVL
Address: 1025 Eldorado Blvd.
City: Broomfield
StateProv: CO
PostalCode: 80021
Country: US
RegDate: 1998-05-22
Updated: 2012-01-30
Comment: ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
Ref: https://whois.arin.net/rest/org/LVL
```

Figure 2-3: A whois lookup on 4.2.2.1.

The whois results will tell you the entire block that the IP address belongs to as well as, probably more importantly, who owns it. Not shown in this particular capture are the contact addresses. When we are talking about owners of IP address blocks, each company has to provide an abuse contact, a contact for the network operations center, and a technical contact. The abuse contact is who someone would get in touch with should any sort of network abuse like spam or a denial of service attack appear to be originating from that network. The network operations center (NOC) is the team responsible for operating the network. This means they keep it running; if there is any problem with getting to this network, someone would likely want to get in touch with the NOC. The technical contact is someone who is technically responsible for the IP addressing. This could be someone who may be handling all of the systems that take care of routing traffic through the network. In most cases, you will find these contacts are generic in nature, meaning they aren't a specific person. In the case of Level 3, they use the same phone number for all of the contacts and that phone number will get you in touch with the network operations staff. The e-mail addresses provided are role addresses and not addresses for specific people.

In the case of smaller network allocations, as in those for a company specifically, you may end up with actual people who you can get in touch with. This isn't to say that you won't get in touch with a person at Level 3, but you may end up with a name and a direct e-mail address for a much smaller organization. This is becoming less and less likely, however. There are a couple of reasons for this. The first is that people tend to change jobs after a period of time. If you register with contact information for a specific person, that contact will need to be updated when that person leaves and then

updated again when the next person leaves. It's better to just use generic contact information that can be adjusted internally as needed. This is especially true since only the contacts can make changes to records with the RIRs.

Another reason is that these databases are public, which makes it very easy to mine them for information. These databases were used by people who were looking for addresses to send solicitation messages to. Typically, these were unsolicited commercial e-mail (UCE) messages, though I'm reluctant to use the term "spam" because that generally connotes bogus messages and these weren't necessarily bogus. They were trying to sell people something, though. This isn't to say that the spammers didn't also harvest addresses from these databases, however. As a result, it was just wiser not to have personal and direct e-mail addresses or phone numbers in the contact records with the RIRs.

Of course, the public availability of this information is one reason why trying to make contact through one of these e-mail addresses isn't likely to get you very far when it comes to large Internet service providers (ISPs) specifically. When I was working at Genuity and involved in operations security, many hundreds of messages came into the abuse mailbox each day. Most of these were junk mail of one sort or another. There was a team responsible for attempting to keep up with the volume of messages that might be from someone who actually needed help. However, that was only a fraction of the entire volume of messages. It may be useful to remember that if the address isn't simply a sinkhole you will likely get an automated response and it may take some time to reach a person.

Internet Protocol and Addressing

We aren't going to spend any more time on the data link layer than we did earlier. Suffice it to say that you have MAC addresses and those MAC addresses are used to get frames from an originating system to another system. These are important concepts for a networking professional, but for our purposes all you really need to know is that the MAC address exists and, for the most part, it identifies a particular network interface. What is important is to have a pretty solid understanding of the Internet Protocol (IP). We're going to look at what IP is for and how it works.

The primary purpose of IP is to make sure that data gets from one end to the other. You may hear about the end-to-end principle and it's the job of IP to make that happen. This means that when UDP or TCP has a datagram or segment to transmit, it hands that off to IP. The IP layer then encapsulates the data and puts a header on it with appropriate addressing to ensure that it gets from the source host all the way to the destination host. The addresses used are structured in such a way that allows for routing, meaning that the packets can find their way through the network by simply knowing their destination. The network itself determines the pathway based solely on the destination address.

Before we go any further, we should look at what the header for an IP packet looks like. In Figure 2-4 you can see the diagram from RFC 791, which described IP in 1981. The header fields you see in this diagram have remained unchanged since then.

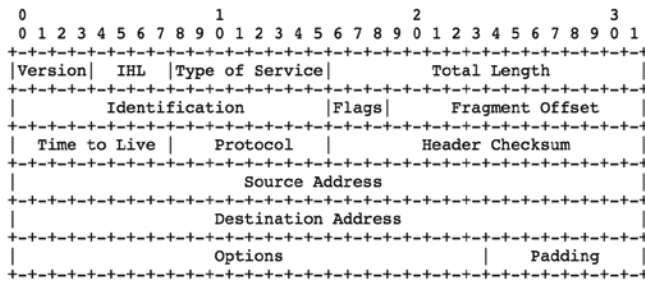


Figure 2-4: IP header diagram.

The first four bits (half a byte) of the header are the version number. Currently you may see either a 4 or 6 in that field because the predominant version across the Internet at the moment of this writing is version 4, or IPv4. For the last decade and a half or more, there has been a move to get service providers and businesses to switch to version 6, or IPv6. The IPv6 header looks different than the IPv4 header, primarily because the length of the IP address has increased dramatically. We'll take a look at the IPv6 header in the "Internet Protocol Version 6 (IPv6)" section later in this chapter. The second half of the first byte (half a byte is sometimes called a *nibble*) is the length of the header. The minimum length for an IP header is 20 bytes. This particular field is calculated as the number of 32-bit words. Since 32 bits is 4 bytes, the minimum (and common) value you will see in this field is 5. That means 20 bytes, since the 4 bytes in 32-bits times the value 5 in the header field is 20 bytes.

The second byte is used for Type of Service (ToS). This was originally meant to be used to provide a value that might allow devices in the network to prioritize traffic. While the ToS may be used, it has effectively been superseded by other methods to prioritize traffic. Because these aren't strictly relevant to us for the most part, we're not going to go into those mechanisms or the additional fields that may be in use to support them. Just know that the second byte is the ToS byte.

Byte number three is the total length of the packet. This includes all of the headers as well as the data. Because the IP layer only knows what it is doing, it is the length of its own header plus the data, which would include the layer 4 header. This value will not include the data link header. This value is in octets and the length of this field is 2 bytes, or 16 bits.

The next 16 bits is the identification field. This is used to identify related packets. If a datagram has to be fragmented in order to be sent out on the wire, the IP ID field will be the same across all of the fragments. The reason that a datagram may be fragmented is because each layer-2 protocol has what is called a *maximum transmission unit* (MTU). This means that there is a maximum size of a frame that can be sent whether the frame is being sent over Ethernet, Asynchronous Transfer Mode (ATM), Frame Relay, or some other layer-2 protocol. In the case of Ethernet, the MTU is 1500 bytes. This includes all of the headers. (I reference Ethernet here because it is a common layer-2 protocol used by copper cabling on local area networks as well as WiFi networks.) If IP has to break up a datagram, which could be 65,536 bytes, into multiple fragments in order to get it out on the wire, it will use the IP ID field to identify all of the different fragments.

The next three bits are used for flags, though only two of the three are actually used. One is the Don't Fragment (DF) bit and it is the second bit since the first bit is unused. The third bit is the More Fragments (MF) bit. If this bit is set to 1, there are more fragments associated with this particular packet. If it is 0, there are no more fragments, which may mean that the datagram wasn't fragmented before transmission. If a datagram or segment only requires a single packet to be sent, this flag will be set in the IP header, indicating that the receiving party shouldn't look for any additional transmissions before sending this message up the stack to the next layer. The next 13 bits in the IP header tell the receiving system where to place the data in the puzzle that is the datagram.

If a 5500-byte datagram is sent, for example, the first packet will have an offset value of 0. This indicates that it is the first piece in the puzzle. Think of the puzzle as a space that has 5500 bytes or puzzle pieces in it except that the puzzle pieces often come bound together in larger, consecutive chunks. The first packet is carrying 1450 bytes in its payload. The next packet will have an offset of 1450. This is because the count starts at 0 so 1450 bytes takes us to a position of 1449 from the first packet being laid in. The next packet will take us from 1450 to 2899. Let's say the next two packets, for whatever reason, only had 1000 bytes for payload but they don't arrive in order. When the fourth packet arrives, there will be a blank spot starting at position 2900 up through position 3899 and the fourth packet will get placed into positions 3900 through 4899. When the third packet does come in, though it arrives after the fourth packet, it will get placed into the 5500-byte chunk in the correct position.

It's the potential for packets coming in out of order that necessitates this field to begin with. Because the Internet is created using something called packet switching, meaning each packet finds its own way through the network, each packet could potentially take a different route, which may mean that a route that initially seemed faster may have gotten slower after the packet was sent, causing it to arrive at its destination after one that was sent later in the stream. In reality, all offsets will be a multiple of 64 (8 bytes). The numbers used for this example were used to make the math slightly easier to follow. The starting to count at zero issue can be confusing for some so using multiples of 10 rather than 64 seems easier to digest as a concept. Because you won't generally be the one to reorder packets into datagrams, this is all conceptual anyway.

After the offset field is a byte that is used for the time to live (TTL). Each time a packet traverses a routing device, the time to live is decremented by 1. If the time to live field becomes 0, the packet is discarded and a message is returned to the sending system using the Internet Control Message Protocol (ICMP, discussed later in this chapter, indicating that time was exceeded in transit. This simply means that the allotted time for the packet to get to its destination expired. A single byte means that you could have a TTL of 255. In practice, the TTL value is rarely that high. It's more likely to be 40 or 64, or sometimes higher. In network terms, anything more than 64 hops away, or passes through a routing device, is a very long way away. Nowhere on the Internet is generally that far away unless there is an error in getting the packet to its destination.

The next byte is the protocol field. This value tells the receiving computer what the next header is going to be so it knows what to look for. Each protocol has a defined value. TCP, for example,

is protocol 6 whereas UDP has a protocol number of 17. One reason that UDP has a much higher number is that IP was originally intended to be the datagram delivery service. UDP was created later to take on that role. The protocol field lets any program doing packet decoding determine how to decode the next set of headers.

The two bytes following the protocol number are the header checksum. A mathematical computation is performed on the header and only the header. The value that is obtained is called a *checksum*. This allows the receiving system to also calculate a checksum on the received header to determine whether it had been damaged in transit. If the checksums match, the header is considered to be complete and intact.

The checksum is followed by 8 bytes (64 bits) for the source and destination IP addresses, in that order. The source address is necessary so the receiving computer not only knows who sent the message but, more importantly, knows where to send any messages back to. In the case of a conversation, the addresses are reversed on a return message. The IP addresses are composed of 4 bytes for an IPv4 address and are constructed in a particular way. There are also some specific addresses that anyone looking at network communications should be aware of.

Internet Protocol Addresses

An IP address is composed of 4 bytes and these bytes are generally displayed being separated by a period or dot. Because of this, the addresses are sometimes called *dotted quads*. Each component of the address is 8 bits, and each component is sometimes called an *octet*. You would have four octets in an IPv4 address and the complete address may be called a dotted quad. Simple, right? Well, it actually gets more complex than that. An IP address is broken up into two components that are different from the bytes that the address is composed of. In every IP address is a network address and a host address. The important thing to know is how you tell the difference between the network component and the host component. In fact, the network itself has no idea which is which. As you can see in the IP header, the only thing you are provided is the address for the source and the address for the destination. How does one know which is the network part and which is the host part?

This begins on the sending computer's side. Every system, when you configure an IP address to a network interface, also has a *subnet mask* that tells the computer which is the network part and which is the host part. Like the IP address, a subnet mask is composed of 4 bytes. The subnet mask is used to identify which bits are in use for the subnet component of the IP address. For example, assume that 255.255.255.0 is your subnet mask. Because 255 is the maximum value of a byte, you know that all the bits in each of the first three octets are used for the network. Because no bits are in use in the last octet, that whole byte is used for the host address. Every host address belongs to the subnet identified in the first three octets of the IP address.

Let's look further at this. If you have an IP address of 172.16.42.25 and a subnet mask of 255.255.255.0, the network component of the IP address is 172.16.42. The 25 is the host address. Subnet masks can sometimes get quite a bit more complicated than this and for the most part, you wouldn't worry that much

about them. What you need to know is that the subnet mask tells the sending computer which destination hosts are on the local network and which are not. If a destination host is in the local network, the sending computer will just send straight to the MAC address and there would be no need to do anything else. If a destination host is not on the local network, the computer will forward the message off using the MAC address of whatever gateway (router) is associated with the destination address.

Each computer has a table called a routing table so it knows where to send packets in case it has multiple interfaces. If there is a single network interface, there is usually just a single default gateway and all packets that are going to another network will be sent to that gateway to be handled there. Once the packet hits the gateway or router, the router would consult its own routing table. As the packet gets closer, the path to the destination gets more specific because the network designations will get tighter. In a very simple example, the first router may know how to get to the 172.0.0.0 network so it forwards off to a router that knows about that network. That router may know about the 172.16.0.0 network and so it forwards off to the next router it knows that knows about the 172.16.42.0 network. Because that router is on the same 172.16.42.0 network as the destination, it then forwards the frame off using the MAC address of the .42 system, which it either knows or can figure out. That, in an exceedingly small nutshell, is routing. The bits associated with the network address are used on the sending side to determine whether a packet is destined for a local network. Once the packet leaves that network, the subnet mask is never referred to or determined again. The routing tables across the Internet allow the packet to get to its destination, without ever really needing to know the subnet mask of the target system.

Because every byte of the address can hold values between 0 and 255, those are the theoretical potential values of each octet in the IP address. In practice, something else happens. First, in the very first octet, the maximum value you will see in a real IP address is 223. This is because, initially, addresses were broken up into classes, based on the bit pattern in the first octet. If the highest-order bit was a 0, the address was a Class A address. That would give you values of 0–127 in the first octet. A Class B address would have a bit pattern starting with 10 with the high-order bits. That would give you values from 128–191. A Class C address would have a bit pattern of 110 giving values of 192–223. A Class D address would be 1110 and have values of 224–239. The problem is that Class D addresses are reserved for multicasting. You may see those addresses, but they would be used for streaming media, typically, and never as an actual IP address of a real system.

In addition to values over 223 in the first octet, there are certain address ranges that you won't see on the open Internet. Large blocks of addresses were set aside early on for private addresses. These private addresses are used on local networks where the messages never need to get out onto the open Internet as they are. If they do need to get out onto the open Internet, a process called network address translation (NAT) has to happen where the private addresses are converted (or translated) to another address that can be sent out onto the open Internet. These private addresses are commonly referred to as non-routable because, by convention, they are not ever put into routing tables on the Internet. They may be used internally to corporate networks but they are never routed between carriers. The private addresses were allocated by classes. Table 2-1 shows the three blocks of private addresses.

Table 2-1: Private Addresses

Class	Network Address	Subnet Mask	Range
A	10.0.0.0	255.0.0.0	10.0.0.0-10.255.255.255
B	172.16.0.0	255.240.0.0	172.16.0.0-172.31.255.255
C	192.168.0.0	255.255.0.0	192.168.0.0-192.168.255.255

The usage of private addresses is specified in RFC1918 and so the various blocks in the table are sometimes called RFC1918 addresses. Any of these blocks can be used on any private network at any time. You will commonly see the address range 192.168.0.0–192.168.0.255 or maybe 192.168.1.0–192.168.1.255 in use on home networks since those addresses are often used on networking devices designed for home use. The router on a home network, often part of a cable or DSL modem, hands out addresses from these ranges as a default. That means that if you have the address range 192.168.1.0–192.168.1.255, the vast majority of people who have the same device you are using will also have that address range on their networks. However, because the devices are translating your IP address for you, there is no fear of the packets from your computer getting lost. Out in the real world on the Internet, your packets will have the IP address that your service provider has given your router.

NOTE The first and last address out of each range of addresses is not used because they have special purposes. The first address, such as 192.168.1.0 out of the range 192.168.1.0–192.168.1.255, is the network address. It identifies and belongs to the network. The last address, such as 192.168.1.255 in the aforementioned range, is the broadcast address. It is used to communicate to all hosts in that network range.

One other range of addresses you will not see in practice is the 127.0.0.0 block of addresses ending at 127.255.255.255. The most common address you will find from that block, by far, is 127.0.0.1. This is the localhost or loopback address. This address refers to the system you are on. Any attempt to communicate with any of the addresses in the 127 block will remain on the system that originated the communication. While the .1 address is the one you will most commonly see configured on your computer, any of the addresses out of the 16,777,214 possible addresses in the block can be used as a loopback address.

NOTE Although you may not run across it as often unless you work a lot with network engineers, there is another way of indicating how large the subnet is. If you have a subnet mask of 255.255.255.0, you have 3 bytes of 8 bits that are used to indicate the subnet. That is 24 bits. The Classless Internet Domain Routing (CIDR) notation to indicate that is /24. If the network you were on is 172.16.5.0 and the subnet were 255.255.255.0, the shorter way to indicate that is 172.16.5.0/24. In other words, you count up the number of bits that are used for the network part of the address and just use that number rather than converting it into a dotted-quad value.

IP has no capability of determining whether packets that are sent have been received. What IP offers is best-effort delivery. IP will do what it can to make sure that the datagrams and segments get where they need to go. It does this by making sure that the packets are addressed correctly. The system that does the addressing just assumes that the network will take care of the rest, which it will if nothing catastrophic happens. One reason for this best-effort delivery model is to make sure that IP isn't adding additional overhead. IP takes care of network addressing and once the network addressing is in place, all of the intermediate systems, called routers, take care of the delivery. If packets are lost because of a network failure, the sending system will have no idea that it has happened based on IP alone.

Internet Control Message Protocol (ICMP)

There is a partner protocol called Internet Control Message Protocol (ICMP). Among other purposes, ICMP is used to transmit error messages. If something happened to the IP packet in the network, one of the devices along the path might send an ICMP error message back to the sending host. This assumes that the router was capable of sending when the failure occurred, which may not always be the case. However, the objective of ICMP is to provide a means for error and other control messages to be transmitted through the network. You may see ICMP used for diagnostic purposes as well, including determining whether a system is up. ICMP echo requests are generally called *pings* because the utility used to send those messages is called ping. When a system is up, it will send back an ICMP echo reply. The utility is called ping because it acts like sonar on an ocean-going vessel. A ship looking for a submarine, for example, would send a sonic burst that sounds like ping, then wait to hear an echo. This is what the ping program does. You may find indications that ping is short for Packet Internet Groper, but in reality, ping was named for what sonar sounds like.

Internet Protocol Version 6 (IPv6)

Once the Internet was opened to commercial interests rather than being limited to the academic and research purpose it had long had, there was a significant expansion. A number of factors, including the lower cost of personal computers and more network-friendly operating systems, meant that a lot more people were getting online either through the company they worked for or at home through an Internet service provider (ISP) that allowed them to use a modem to connect to the network. This increase in usage meant a lot more need for IP addresses. Because an IP address is 32 bits long, a maximum number of 4.3 billion addresses are available. Many of those immediately get dropped out because of the RFC1918 and loopback address spaces. Other addresses, above the multicast address range, are simply unused. Even if the whole 32 bits of address space were usable, there simply weren't enough addresses to allow for the large numbers of people who were getting online starting in the mid-1990s. Because of the limited number of addresses, private addresses were often implemented, particularly on home networks. The router or some other device would replace the private address with an Internet-routable address in a process called *network address translation* (NAT).

The next version of IP was designed to fix the issue of limited address space as well as a few other problems in the existing version 4 of the protocol. The first thing that was necessary was an increase in the number of potential addresses. At the time IP was originally designed, it was felt that 32 bits would be more than adequate considering that only very large and expensive computers were connected to the network. This time, to avoid similar problems in the future, the address space was dramatically increased. Instead of 32 bits, IPv6 has 128 bits in every IP address. While on the face of it, that looks like a factor of 4 increase in the number of addresses, keep in mind that capacity doubles for every bit increase. Instead of multiplying the number of addresses available by 4, we are multiplying the number of addresses by 2^{96} . This is the difference between the new 128-bit addresses and the 32-bit addresses previously used. What this means is that you multiply the current 4 billion addresses by $7.9 * 10^{28}$ to get the total number of IPv6 addresses, which is $3.4 * 10^{34}$.

Because it would be a lot of work to represent the address in decimal, since you would have sixteen 3-digit numbers, an IPv6 address is represented as sixteen hexadecimal digit pairs. Each pair of hexadecimal digits represents a single byte since ff is equal to 255, which is the maximum value of a byte. Rather than dots, the hexadecimal digit pairs are separated by colons. With IPv6, you might see something like fe80:0000:0000:0000:8b11:fc3f:7015. You will see a lot of 0s in the middle, and since that takes up a lot of space for no good reason, we can shorthand the address to be fe80::8b11:fc3f:7015. The :: means fill all of this in with 0s. Since we were missing 8 bytes, all of those 8 bytes there are just 0s.

The address space wasn't the only fix that was put into IPv6. The specification for IPv4 doesn't include any capacity for privacy or preventing tampering. A number of extensions, originally developed for IPv6, called IPSec (for IP Security) were added onto IPv4 by various vendors to allow networks and individual users to communicate with, say, a corporate network in an encrypted and private manner. Since all of this was created for IPv6, it's no surprise that IPv6 has all of these features and capabilities natively. This ability to support end-to-end encryption without requiring the session layer to provide encryption, is a large leap forward for privacy but it will also make life much harder for network forensic investigators since all of the information will be encrypted.

One advantage introduced by IPv6 is the ability for each host to automatically configure itself with a network address that all other hosts on the network could get to. Previously, that typically required a special system called a Dynamic Host Configuration Protocol (DHCP) server that handed out IP addresses and other network configuration information. Without a DHCP server in IPv4, hosts would assign themselves link-local addresses, sometimes called *zeroconfig addresses*. There is no guarantee that hosts assigned these addresses could communicate with other hosts on the same physical network. While the need for DHCP doesn't entirely go away, systems can automatically configure their network address. Rather than contacting a DHCP server, they send out a message looking for the router on the local network. The router replies with the network address. The system appends its MAC address since all MAC addresses are expected to be globally unique, so there should never be two systems with the same MAC address on the same network. This avoids the potential for multiple systems with the same IP address. IPv6 also supports its own version of DHCP, referred to as DHCP6, which provides host configuration in a similar way to the older version of DHCP.

The considerably larger address space and the utilization of the already unique MAC address alleviates the need for the private addresses previously used. While there are a number of other enhancements over IPv4 in IPv6, the address space and the use of encryption and verification of message contents are really the two big ones that would impact a network forensic investigator.

Transmission Control Protocol (TCP)

As mentioned previously, IP offers best-effort delivery. However, many applications need something better than best effort. Most of the time, you want packets you are sending to be guaranteed to get to the destination host so there needs to be another means to guarantee the delivery of the application data. Of course, it could be left up to the application protocol but that would be a lot of duplication. Instead, the transport protocol layer could be used to provide that guaranteed delivery. Of the different transport layer protocols available, the Transmission Control Protocol (TCP) offers guaranteed delivery meaning that the sending host will either get the message through or receive a message indicating why a failure occurred. In reality, there is no getting around a catastrophic network failure or outage but TCP is designed in such a way that messages are tracked to ensure they get to their destination, in order. If this doesn't happen, the application will be notified by TCP that a failure occurred. Since a communication stream between two endpoints may require a number of segments before being considered complete, those segments need to be in order or they simply won't make any sense. Imagine, for example, reading "love run smooth never did The course of true." That's what happens when you get chunks of data out of order. You may be able to look at that and rearrange the words to get the line from *A Midsummer Night's Dream*, but it would be quite a bit better if all the words were in the correct order to begin with.

To understand how TCP works, we need to take a look at the TCP header. Keep in mind that the TCP header encapsulates the data that is being sent by the application. The encapsulated data is then sent on to IP to be addressed. In Figure 2-5 you can see the header as diagrammed in RFC793, published in September 1981 as the specification for how TCP was intended to work.

TCP has its own manner of addressing to make sure the correct application running on the system gets the segments. Where IP uses an IP address to make sure the packets get to the correct host system, TCP makes use of *ports*. This is a way of allowing multiplexing so many applications can be communicating at the same time. Ports in TCP are 16-bit values, and just as there are source and destination addresses, there are source and destination ports. The first 32 bits of a TCP header are the source and destination port. Because there are 16 bits, you can have values of 0 to 65535. This gives you a total of 65,536 ports available to applications.

NOTE The first 1024 ports, 0–1023, are called *privileged* ports. That means that only the administrative user can establish a listener on those ports. Applications with listener ports are server applications. Applications that are clients are assigned ephemeral ports, meaning they are transient and will go away as soon as the application no longer needs them.

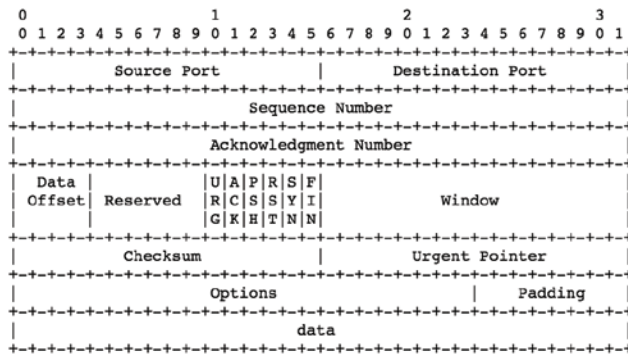


Figure 2-5: TCP header diagram.

After the ports, the next 32 bits is the *sequence number*. The sequence number is expected to be a random number established at the beginning of a communication stream. The sequence number is incremented by the number of bytes transmitted as the communication progresses. The sequence number allows all of the segments to be in order since the receiving system will know exactly where in the stream a particular segment belongs. The sending system is responsible for maintaining the sequence number as it transmits. It is directly related to the acknowledgment number.

The *acknowledgment number* is the next 32 bits. The acknowledgment number is sent by the receiving system to the sending system to indicate the next number byte it expects. Let's say that the initial sequence number for a communication stream is 545. The sending system had transmitted 20 bytes. In order to indicate to the sending system it had received 20 bytes, the receiving system would send back an acknowledgment of 565, indicating it had received bytes 545–564 (20 bytes) and was awaiting byte number 545. If the sending system had sent 40 bytes but it only received an acknowledgment for 20 bytes, it would know that after a specific period of time it would need to retransmit. This is how TCP defines guaranteed delivery. The sending and receiving parties keep track of where they are in the conversation and acknowledge everything that is sent. If an acknowledgment isn't received, the sending system needs to retransmit until it either receives an acknowledgment or the number of sends, determined by the operating system implementation though three is the specified minimum, is reached, at which point the connection will generate an error.

The data offset, which is a 4-bit value, indicates where the data begins. This effectively tells the receiving system how long the TCP header is. Because the TCP header supports optional values, it is not fixed length so there needs to be a value in the header that indicates where the data can be found. There are 6 bits after the offset that are reserved and used for nothing. This is followed by 6 bits that are used for flags. These flags are either set or not set, so a single bit represents a single flag. The flags available in TCP are as follows:

- URG—The urgent flag indicates that the urgent pointer field is significant and should be read.

- ACK—The acknowledgment field has a value that should be read. This flag would be set if a receiving system were sending an acknowledgment back to a sending system to indicate where in the data stream they were.
- PSH—Data coming in on a TCP connection is held for a period of time. This is a process called *buffering*. Once a buffer is full, the data will be sent up to the application. If a segment comes in with the PSH flag set, the data in the buffer will be pushed up to the application, regardless of whether the buffer is full.
- RST—If this flag is set, the connection is reset, meaning that any communication that may be ongoing is terminated. If a system wants to communicate again, it needs to re-establish the connection.
- SYN—This flag is used to indicate that there is a sequence number that needs to be synchronized. The receiving system should make note of the sequence number for further communications with the sending system.
- FIN—This flag is used to indicate that communication is ending. No further messages are going to be sent from the sending system.

After the flag bits are 16 bits of a window field. The window field indicates the number of bytes (including this field) the sender of the message is willing to accept. This means that the sender of this message will send an acknowledgment prior to the window that tells the receiving system when it should expect acknowledgments from the sender. If the number of bytes transmitted goes outside of this window, there is likely a problem that needs to be addressed with retransmits.

The next 16 bits are the checksum. This is a calculation that indicates whether the header and the data associated with it have been corrupted in any way. This field will allow the receiver to also calculate a checksum and make sure it is equivalent to the checksum sent. If not, there is something wrong with the message. While the checksum is being calculated, the checksum field itself is filled with 0s in order to ensure the calculation is consistent on both ends, since it won't be filled in at the time it is being calculated on the sending end.

Finally, in the standard TCP header is the urgent pointer. If the URG flag is set, this indicates where the receiving system should be looking for the urgent data. This field is an offset value from the sequence number in this segment. After the urgent pointer field, there may be options. In order to get aligned to a multiple of 32 (4 bytes), the header may have some padding if the options don't line up correctly. After any optional header fields is the data that is being transmitted for the application. The standard TCP header is 20 bytes long. If there are no options in either the IP header or the TCP header, the total length of the IP and TCP headers together is 40 bytes.

You can start to see how TCP manages to provide guaranteed delivery to the applications it serves. One piece that is missing from what you have seen in this section is the communication that establishes some of the header fields between the sender and the receiver. When two systems want to start communicating over TCP, they perform a three-way handshake that establishes the important data fields. It's the three-way handshake that establishes a connection between the two systems.

NOTE While this chapter has primarily been using the word “byte,” the RFCs for the protocols described here use the word “octet.” The reason for this is that the byte wasn’t a standardized value. Some bytes were 8 bits while others were 10 bits. It depended entirely on the system. To avoid any confusion, the word octet is used to indicate an 8-bit value. For our purposes, because the byte on personal computers is 8 bits, the two words are synonymous.

Connection-Oriented Transport

TCP not only provides guaranteed delivery, it also makes use of connections. This means that before two systems begin to transmit actual data, they go through a process to establish a connection. The process is called the three-way handshake. It’s actually a four-step process but two of the steps have been collapsed into a single segment, making it three segments that are transmitted. The three-way handshake guarantees that both parties are alive and reachable, which prevents one side from pretending they are someone else. It also establishes the sequence number that is needed to guarantee orderly delivery during the communication between the two systems.

The very first segment that is sent is called a SYN message. This is because the SYN flag is set. This flag tells the receiving host that it should take note of the initial sequence number (ISN). That will be the value that the remainder of the communication is built on since every message of data that is sent will increment the sequence number by the number of bytes sent. This allows the two hosts to both know where in the communication stream they are. The diagram in Figure 2-6 shows the first message with the SYN flag set and the initial sequence number established. This initial message would generally be originated from a client looking to establish a connection with a server in order to obtain a service. If you were looking to connect to a web server, for instance, you would go through this process between your system and the system the web server application is on.

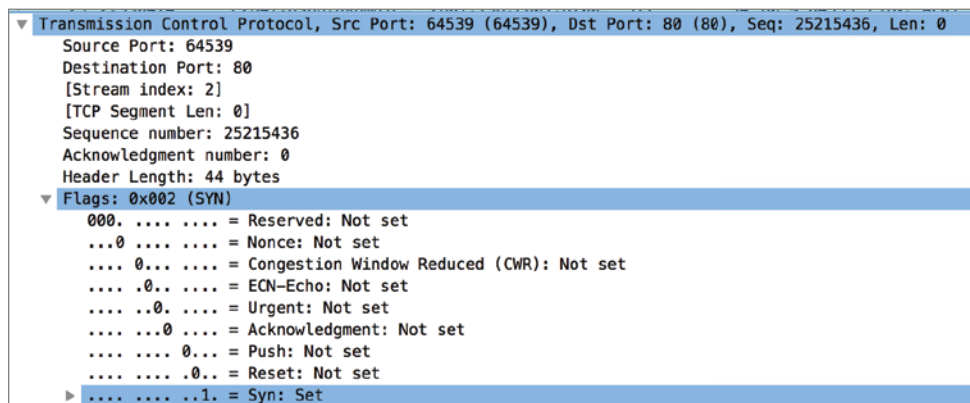


Figure 2-6: SYN message.

The second segment that is sent is a collapse of two steps into a single message. The first part is the ACK, indicating that the ISN has been received. The ACK indicates that the acknowledgment field has a value that will be the ISN + 1, indicating that the first message has been received and the recipient is awaiting the very first byte of data. At this point, no actual application data has been transmitted so the segments only include headers and no data payloads. The second part of the message is the server establishing its own ISN. As a result, the SYN flag is also set. Since both ends of the communication are expected to be sending data, both parties need to keep track of where they are in the communication stream and both need to acknowledge receipt of segments. This would be impossible with just one sequence number. Both ends need to have their own sequence number. This segment will have both the SYN and ACK flags set and it will also have values in both the sequence number and acknowledgment number fields. Because both the SYN and ACK flags are set, this is commonly called the SYN/ACK message.

The SYN/ACK message establishes to the client that the server is alive and responsive to messages. The server responds to the source IP address in the SYN message. Assuming that the SYN/ACK doesn't generate any errors, the client is expected to also be up and responsive. The last message in the handshake is the response to the SYN message from the server. This is a segment that has the ACK flag set and the acknowledgment field is set to the server's ISN + 1 to indicate that the initial sequence number was received and the client is ready to receive data from the server.

Once the three-way handshake is complete, the two systems have a foundation on which to build a communication stream. Because pretending to have a different IP address is so easy, the full handshake is important to protect against spoofing. Spoofing is when one system pretends to be another one. Because both ends of a TCP communication have to establish ISNs and respond as though they know the ISN, it mostly protects against spoofing. Spoofing is commonly used in an attack against TCP servers, though. If an attacker sends just the initial SYN message with a spoofed source IP address, the server will respond to the bogus SYN message to a system (the real owner of the falsified IP address) that isn't expecting it. This leaves the connection in a half-open state. The server is awaiting the ACK message so it holds a buffer open for the expected connection with the other system. Because a finite number of buffers can be held open like this, a SYN flood, as this attack is called, is focused on using up all of those buffers so legitimate clients won't be able to establish connections with the server.

When one system receives an unexpected TCP message, as in the case of the real owner of the spoofed IP address, it will respond with a message with the RST flag set. This reset message tells the sending system that it has no idea what is going on so shut down what it's doing and stop sending messages. It's a simple way of saying don't bug me with these unsolicited communications. If the SYN/ACK makes it to a live system that sends back a RST message, the half-open connection will get torn down. As a result, the best way to perform a SYN flood is with source addresses that won't respond. If the spoofed source doesn't respond to the SYN/ACK with a RST, the host under attack will leave the connection half-open for longer.

During the course of a conversation, both sides will send periodic messages with the ACK flag sent and the acknowledgment number indicating the next byte number that is expected. As noted previously, the spread of these acknowledgments is based on the size of the window. The size of the window can change through the course of the conversation, depending on the conversation and how smoothly it is going from a network flow perspective.

When the conversation is over, both sides will need to tear down the connection. This is a two-step process, just as the connection is set up using two steps in both directions. Rather than a SYN and then an ACK, both sides will send a message with the FIN flag set, indicating that the party sending the FIN is done communicating. The only response to a FIN is an ACK. All this says is that the sending party is done sending messages and the receiving party has acknowledged that. This doesn't say that the receiving party is done sending messages itself. Because there are two pathways in every conversation, $A \leftrightarrow B$ and $B \leftrightarrow A$, both parties need to indicate separately that they are done communicating. If A is done talking, it will send the FIN and B would respond with an ACK. B may still have things to say. It can continue to transmit and A would be required to ACK any messages that are sent by B until B sends a FIN. Once A receives the FIN from B, it sends an ACK back.

This ends up creating a four-way teardown, and this is necessary because it's not possible to completely synchronize when both parties are prepared to tear down. It's easiest and quickest to allow both sides to initiate their own teardowns separately. In practice, you will generally see the two FIN messages from both sides happen at about the same time. It would be unusual for there to be a long interval between the two parties sending their teardown requests. Once the teardown is complete, the connection can be removed from the connection table that each operating system keeps track of.

User Datagram Protocol (UDP)

Where TCP offers a connection-oriented service, the User Datagram Protocol (UDP) is connectionless. This means that there is no establishment of a connection and no tracking at the operating system of the communication between the two systems. Every communication message is done as a one-off. At the operating system level, there is no tracking. Messages are sent and promptly forgotten. By this, I mean that the network stack within the operating system kernel, the core of the operating system, handles UDP and TCP messages. If any tracking is done, it can be done within the application.

You may be wondering, if TCP offers guaranteed delivery and sequencing and all the other goodness it has to offer, why would anyone have any use for UDP? In reality, there are a lot of uses for UDP. UDP is fast and lightweight. It has little in the way of overhead and you don't have to wait for a connection to be established before you send a message along. Once the application has prepared a message, it's out the door. Any application that is concerned with speed, like streaming video, will want to use UDP. If you are streaming video, you don't want to be hung up waiting for a particular segment to arrive if there is out-of-order delivery. With UDP, you send it on its way; if one or two datagrams are dropped or lost or arrive out of order, they are simply discarded or forgotten. You likely won't miss them at all. Waiting for them would cause a jarring experience for watching or

listening. Trying to insert out-of-order messages into the stream when they arrive would be similarly jarring, though for different reasons. As a result, UDP is best for any application that needs real-time transmission capabilities.

Because the protocol itself doesn't do a lot other than offer a means of transport and multiplexing, the headers are very simple. This also helps with speed because fewer bytes, even if it's not all that many fewer for each datagram, means it can get to the recipient much faster and requires less processing on the receiving end. Figure 2-7 shows the short header that UDP uses.

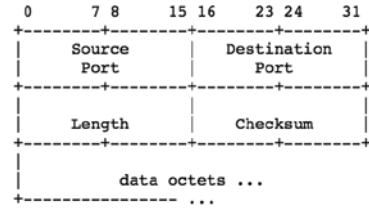


Figure 2-7: UDP header diagram.

Just as with TCP, the first 4 bytes are the source port and the destination port. Because each port field is composed of 2 bytes, there are 65,536 ports available in UDP. Again, there is a source and a destination so the datagram not only knows the address when it gets to the system, but also any response can come back to the right port on the system that originated the message. Of course, with a UDP message, there is no guarantee of any response. This is not to say that the messages will get lost. Instead, UDP itself specifies no response to any message it receives. As a result, communications can be entirely one-way if that is the way the application wants it to be.

After the port numbers is the length field. As with TCP, this provides the size of the data to the receiving end so it knows how many bytes will be read as part of this datagram. Since the length field is 2 bytes, the maximum size of a UDP datagram is 65,535 bytes. That is the maximum value of 16 bits, expressed in hexadecimal as 0xFFFF. Although the UDP datagram can be up to 65,535 bytes long, it would never go out onto the wire intact. The IP layer would fragment it on the way out and then reassemble it at the receiving end before forwarding the reassembled datagram up to the UDP layer.

After the length field is the checksum. This verifies that the datagram that has been received is the same one that has been sent. Just as with TCP, it is calculated as though there were 0s in that field.

That's really it. A total of 8 bytes in the UDP header. There are no options. That is all that is needed to make UDP work. Of course, because it's so simple, there are some challenges that the application layer needs to take care of.

Connectionless Transport

Where IP is best-effort delivery and TCP is connection-oriented and considered guaranteed, nothing is guaranteed with UDP other than the best-effort delivery used by the underlying IP layer. UDP is, in contrast to TCP, connectionless. TCP makes use of the three-way handshake to establish a baseline for communication. It uses this baseline to ensure segments are delivered through the process of acknowledgments. UDP has no such handshake, and as a result, UDP datagrams are sent out onto the network in the hopes that they arrive at their destination. Since there is no defined behavior for how a system receiving a UDP message is supposed to act, the sending system has no way of knowing that the datagram has been received. All of that behavior is left up to the application to take care of.

Some applications take advantage of the almost transient nature of UDP datagrams, considering they leave almost no trace in terms of connection states. As an example, the Domain Name System (DNS) uses UDP transport for requests for an IP address from a hostname. Most systems are configured with multiple DNS systems for looking up IP addresses. These systems will often request information from more than one DNS server at a time. Since sending the request out doesn't get any sort of acknowledgment, the requestor can't know whether it was received unless it receives an answer, and those can sometimes be delayed or lost. As a result, systems can send multiple DNS requests nearly simultaneously to increase the chances of getting an answer without worrying about any overhead or additional time. The first DNS server to respond to the request effectively wins. Any additional responses are just discarded.

Ports

At the transport layer, ports are used to provide multiplexing. It's the way we provide an address at that layer to make sure that the correct application gets the message that has been sent its way. This has been referenced a few times but it may be helpful to dig a little deeper into ports to understand how the concept operates on a running system. There are a number of ways of conceptualizing the idea of ports. You could think about a wall of mailboxes, as you might see in your local post office. Messages go in and are retrieved. The problem with mailboxes as an analogy is that mailboxes are more store and forward than the real-time nature of ports, meaning a message may sit in a mailbox for a period of time until the message is retrieved. In reality, there may be a bit of that going on at the program level, depending on how the program has been written. Network communication, when we are talking about sending a segment or a datagram to a TCP or UDP port, is real time, though.

In the end, everything is about the applications communicating. The ports identify who is doing the communicating and the ports are handled by the operating system kernel. For our purposes, let's say there are two types of programs: a server and a client. The server has something to offer to the rest of the network. The client is a program that consumes the services offered by the server application. Perhaps the most common example of this model is the web server and the web browser. When you communicate with `www.google.com`, you are sending a request from your web browser to the web server sitting in a Google data center. On one side, you have the server. On the other, you have the client.

On the server side, the program that is performing the server functions sends a request to the operating system asking to bind to a particular port. Because only one application can be bound to a port at any time, the operating system is the final arbiter and it uses a first-come, first-served approach. In other words, if someone gets there before you, you've lost out. If the port is available, the operating system will say okay. At that point, the application will begin listening on that port. This means that the application will wait to be informed of a connection request coming from a client. You can see one of those requests in Figure 2-8.


```

▼ Transmission Control Protocol, Src Port: 55514 (55514), Dst Port: 80 (80), Seq: 3208058599, Ack: 4101703762, Len: 131
  Source Port: 55514
  Destination Port: 80
  [Stream index: 6]
  [TCP Segment Len: 131]
  Sequence number: 3208058599
  [Next sequence number: 3208058730]
  Acknowledgment number: 4101703762
  Header Length: 32 bytes
  ▶ Flags: 0x018 (PSH, ACK)
  Window size value: 4117
  [Calculated window size: 131744]
  [Window size scaling factor: 32]
  ▶ Checksum: 0x6e4c [validation disabled]
  Urgent pointer: 0
  ▶ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▶ [SEQ/ACK analysis]
  ▶ Hypertext Transfer Protocol

```

Figure 2-8: Client to server communication.

In the TCP header, you can see the source port and the destination port. The destination port is the port the server is listening on. The destination port in this message is 80, one of the well-known ports assigned to the Hypertext Transfer Protocol (HTTP) for web communication. A well-known port is one that has been assigned to a particular service like HTTP (web), FTP (file transfer), DNS, and a number of others. Essentially, a well-known port is the default port number for commonly used application protocols. Just because a port is commonly assigned to a particular service doesn't constrain the port that can be used by a server. You can establish a web server on port 5489 if you want to. It's just that the web browser expects to go to port 80, so you would need to do something extra to get a client to communicate with that server.

On the source port side, operating systems make use of a block of port numbers that are handed out on an as-needed basis. Since they are for temporary use, they are called *ephemeral* ports. They are assigned and returned as the communication is established and torn down. The Internet Assigned Numbers Authority (IANA) is responsible for IP addresses and also port number assignments for the well-known ports. IANA recommends using ports 49152–65535 for ephemeral ports. This isn't a requirement, however. Many Linux kernels use 32768–61000 for ephemeral ports. Modern Windows systems have used the IANA-recommended ports for ephemeral port assignment but the ports can be redefined if a system administrator wants to.

A small number of ports are reserved for administrative access only. The program may have to have administrative or root user permissions in order to establish a listener on ports below 1024. Any port over 1023 can be used as a listener by anyone on the system, though ports 1024–49151 are in a block of ports whose assignment is managed by the Internet Assigned Numbering Authority (IANA). While any programming running as any user can use the numbers, they may collide with a service that has already been assigned that number. Since programs that use the low-numbered ports are often considered essential services, they were grouped together with the assumption that these essential services could only be started by an administrator at the system level and not a program that any user on the system could start up.

NOTE Because UDP uses what is effectively one-way communication, a source port isn't required in a UDP header, depending on whether the application expects a response. In cases where no response is expected, a source port may not be assigned and the value 0 may be used instead.

Domain Name System

While systems are all provided IP addresses in order to communicate across the Internet, we aren't great at memorizing IP addresses. Not only that, IP addresses can change and if the address changes, everyone would have to memorize the new address, assuming they can even discover it. It's far easier for people to remember hostnames because there is usually some context for that. In the early days of the Arpanet, a hosts file was shared to all of the sites on the network. This hosts file was centrally maintained and when changes were made, the edited file had to be redistributed to the entire network. As the Arpanet continued to grow, it became clear that the hosts file approach wasn't sustainable or scalable. In response, the Domain Name System (DNS) was created. DNS alleviated the challenge of maintaining and distributing an up-to-date hosts file but also allowed multiple hosts with the same name as long as they belonged to different domains.

A domain is a way of organizing multiple hosts into a single naming group. Each host will have its own hostname, which is the common name that is provided to it, and every host will belong to a domain. When you put the hostname and the domain name together, you have a fully-qualified domain name (FQDN). The FQDN says that you are as specific as you can be with respect to what host you are referring to. If I were to just mention zaphod, calvin, or opus, for example, there may be countless systems around the Internet with those names. The way to get to a unique system is to add the domain name to it and make it an FQDN.

NOTE There are ways to have an FQDN refer to multiple IP addresses for the purposes of load balancing, but for simplicity we are going to assume a one-to-one relationship between FQDN and IP address.

Domains are organized hierarchically. At the very top are the top-level domains (TLDs). While the most well-known ones are likely to be .com, .net, .edu, .gov, and .org, all of the country-specific domains like .sk, .us, and .ca are all TLDs. Basically, when you carve up a domain name, the chunk after the last dot is the TLD. Beneath the TLDs are second-level domains. In the case of a US business, this might be the business name as would be the case of Microsoft.com, Google.com, or ATT.net. Businesses in other countries, since they may be using the country identifier for their TLD, may have a different organizational structure. At a basic level, though, imagine it as the hierarchy

shown in Figure 2-9. At the top of the hierarchy are the TLDs, which would be the right-most part of the domain name. You then build on that by adding additional components right-to-left as you work down the hierarchy.

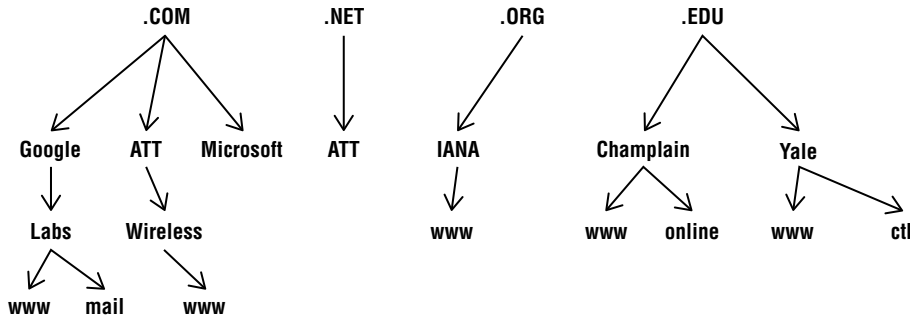


Figure 2-9: Domain hierarchy.

Underneath the second-level domains are the subdomains. You may not run into these, though some businesses have them. For example, `wireless.att.com` is a subdomain of `att.com`. At some point, though, you start adding on hostnames. In the case of `wireless.att.com`, the website would be the FQDN `www.wireless.att.com`. That is the website for the subdomain `wireless.att.com`, which is a different host than `www.att.com`.

In order to get from a hostname to an IP address, we have to do a DNS lookup, called resolving the address. There are two types of DNS servers. One is an authoritative server, which is the server you want to go to in order to get definitive answers about hosts within a particular domain. Every domain will have a set of authoritative servers. Performing a whois lookup on a domain name will give you the authoritative servers for that domain because the RIRs store the name servers along with the remainder of the records for the domain. The following code shows the three authoritative servers for the domain ATT.COM. To determine an IP address in the domain ATT.COM, the most accurate answer will always be with one of these hosts:

```

Name Server: ns1.attdns.com
Name Server: ns2.attdns.com
Name Server: ns3.attdns.com

```

The other type of DNS server, and the type that your system would be interacting with on a regular basis, is the caching or local DNS. As one example, Google maintains a caching DNS server for public use at 8.8.8.8. My ISP is Comcast and one of my DNS servers is 75.75.75.75. These are the servers your computer will contact in order to look up IP addresses and other DNS information. It does this using a recursive query. Because DNS information is a hierarchy, the query starts at the very top and successive queries get more specific until we have the information that we are looking for. Let's look

at how this would work, using `www.wireless.att.com` as an example. You can follow the hierarchy on this one by referring to Figure 2-9. Additionally, you can follow the queries in Figure 2-10. The first query goes from your computer to the caching DNS server, which does all of the work from there.

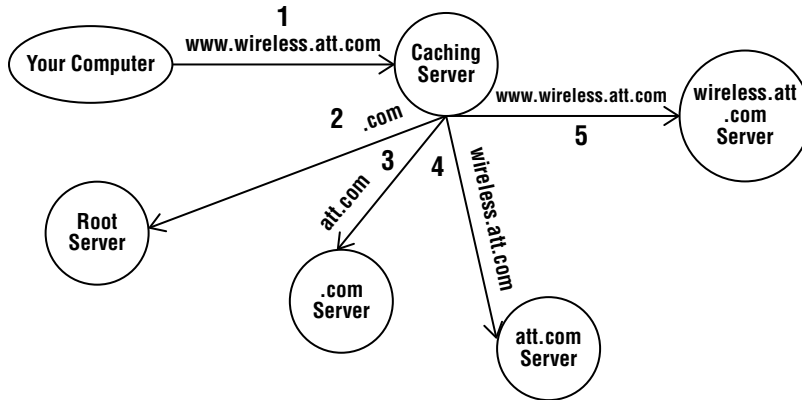


Figure 2-10: Recursive DNS query diagram.

Every DNS server has a set of hints, which are the addresses of the root name servers. The root name servers have addresses of the appropriate DNS servers for the TLDs. The second query goes from the caching DNS server to the root servers to get the IP address for the `.com` DNS server. The third query goes to the `.com` DNS server to get the IP address for the domain `att.com`. The fourth query goes to the domain server for `att.com` asking about `wireless.att.com`. Finally, once we know the IP address for the subdomain `wireless.att.com`, we can ask about the hostname `www.wireless.att.com`, getting an address back that can be returned to your computer. All of these requests are done over UDP, which allows them to be quick. Still, though, it takes time to make the number of requests that are involved in the query referenced in Figure 2-10. This is why the local servers are called caching servers. In order to save time, they cache responses so they don't have to look up the information every time.

Domain records are stored in zones and every zone has a set of metadata associated with it. This metadata is stored in a Start of Authority (SOA) record in the DNS server. The SOA record provides a serial number for the zone, which can indicate whether the zone was updated since the last time the SOA was queried. If the serial number is different, information in the zone has changed. Additionally, the SOA record has timer information indicating when information within the zone will time out. Caching servers pay attention to this information in order to determine how long they can hold onto a record. We can see the values associated with the records in the trace of the query `www.wireless.att.com` done using the `dig` command in Figure 2-11.

The first query gets the root server values. Because of the size of the Internet, there are a lot of root servers. The second query gets the value for the DNS servers associated with the `.com` domain. Again, there are a number of these in order to accommodate the large number of queries directed

there. The third query gets the address of the `att.com` domain name servers. In this case, `att.com` doesn't have a separate set of DNS servers for their `wireless.att.com` subdomain so there isn't an additional query. The numbers you see in the second column of the answer output are the time to live (TTL) values for each answer. The TTL value is in seconds. The long values associated with the root name servers indicate that they don't expect values to be changed very often and they don't want to get a lot of queries, so caching servers should hold onto the values for a long time. Comparatively, the TTL for `www.wireless.att.com` is much shorter.

```
kilroy@oliver:~$ dig +trace a www.wireless.att.com

; <<>> DiG 9.8.3-P1 <<>> +trace a www.wireless.att.com
;; global options: +cmd
.          517011 IN      NS      l.root-servers.net.
.          517011 IN      NS      j.root-servers.net.
.          517011 IN      NS      k.root-servers.net.
.          517011 IN      NS      l.root-servers.net.
.          517011 IN      NS      m.root-servers.net.
.          517011 IN      NS      a.root-servers.net.
.          517011 IN      NS      b.root-servers.net.
.          517011 IN      NS      c.root-servers.net.
.          517011 IN      NS      d.root-servers.net.
.          517011 IN      NS      e.root-servers.net.
.          517011 IN      NS      f.root-servers.net.
.          517011 IN      NS      g.root-servers.net.
.          517011 IN      NS      h.root-servers.net.
;; Received 496 bytes from 75.75.75.75#53(75.75.75.75) in 330 ms

com.       172800 IN      NS      h.gtld-servers.net.
com.       172800 IN      NS      i.gtld-servers.net.
com.       172800 IN      NS      f.gtld-servers.net.
com.       172800 IN      NS      e.gtld-servers.net.
com.       172800 IN      NS      b.gtld-servers.net.
com.       172800 IN      NS      l.gtld-servers.net.
com.       172800 IN      NS      d.gtld-servers.net.
com.       172800 IN      NS      k.gtld-servers.net.
com.       172800 IN      NS      g.gtld-servers.net.
com.       172800 IN      NS      m.gtld-servers.net.
com.       172800 IN      NS      a.gtld-servers.net.
com.       172800 IN      NS      c.gtld-servers.net.
com.       172800 IN      NS      j.gtld-servers.net.
;; Received 510 bytes from 192.36.148.17#53(192.36.148.17) in 399 ms

att.com.   172800 IN      NS      ns3.attdns.com.
att.com.   172800 IN      NS      ns2.attdns.com.
att.com.   172800 IN      NS      ns1.attdns.com.
;; Received 147 bytes from 192.33.14.30#53(192.33.14.30) in 95 ms

www.wireless.att.com. 10800 IN      CNAME   www.wireless.att.com.edgekey.net
;; Received 84 bytes from 144.160.20.47#53(144.160.20.47) in 52 ms
```

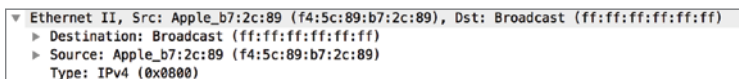
Figure 2-11: DNS query trace.

A number of commands can be used to perform IP address lookups. On the Linux and macOS side, you can use `dig`, `nslookup`, or `host`. On a Windows system, `nslookup` is installed by default. In order to use another utility, you could find and install one or use Microsoft's PowerShell. There are also web interfaces that would allow you to perform lookups, like the one at Google. Google allows you to perform a `dig` query like the one in Figure 2-11 at <https://toolbox.googleapps.com/apps/dig/>. In addition to looking up IP addresses from hostnames, you can look up hostnames from IP addresses, mail server addresses, and other resource records for the domain. DNS uses a number of different record types to distinguish what the record indicates. Using a tool like `dig`, you can specify which record type you want to look up.

Support Protocols (DHCP)

You will run into a large number of protocols after you have looked at enough network traffic and over the course of the book, we will look into some of them in more detail. Two protocols are required for basic network functioning, though, that you will see on a regular basis. The first and most straightforward is the Dynamic Host Configuration Protocol (DHCP) for IPv4, which is a little different than the one for IPv6. This protocol is used to automatically configure a number of network-related settings on any system that needs to be configured. For the most part, you may be used to just starting up a new computer, adding it to your local network—either by plugging an Ethernet cable into the computer or, more often, adding the computer to the WiFi network—and you're ready to go. This is because there is a DHCP server running on your network. Most consumer devices like Wireless Access Points (WAPs) or cable/DSL modem/routers include DHCP servers. This makes life considerably easier for people. All they need to do is do some basic settings on the WAPs or modem/routers and then get started. This assumes they don't want to take the defaults, including the Service Set Identifier (SSID) that comes on the WAP. This is the network name and it's generally a good idea to change it to something more unique to you. However, if you don't want to change, it's usually very simple to add one of these devices to your network and then just add your computer to the network. DHCP takes care of all of the configuration details.

The way DHCP works is a client, meaning a computer that needs configuration settings, sends out a request called a DHCP Discovery message, looking for a DHCP server. This is an IP packet, but we have a small snag because the host in question doesn't have an IP address. For this reason, the DHCP server must be on the local network where the client system can just use a broadcast address at layer 2, as you can see in Figure 2-12. The broadcast MAC address is ff:ff:ff:ff:ff:ff.



```
▼ Ethernet II, Src: Apple_b7:2c:89 (f4:5c:89:b7:2c:89), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ► Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ► Source: Apple_b7:2c:89 (f4:5c:89:b7:2c:89)
  Type: IPv4 (0x0800)
```

Figure 2-12: Layer 2 broadcast request.

There are ways around having the DHCP server directly on every local network if you are dealing with an enterprise network, but that's done in the network infrastructure and the requests will look identical. While the layer 2 addressing is very straightforward since every network interface has its own MAC address, there is still the challenge of filling in details in the IP header. We still don't yet have an IP address. As a result, a system looking for an IP address uses 0.0.0.0 as a source address since all of the communication is being done on the local network and won't need any routing that would require a real source IP address. The client system has no idea where to find the DHCP server, either from a layer 2 addressing perspective or a layer 3 addressing perspective. Since that's the case, the system needs to use a broadcast address in the IP header. The broadcast address for IP is

255.255.255.255, so the source is all 0s (all bits set to 0) and the destination is all 255s (all bits set to 1). DHCP uses UDP as the transport protocol and uses 67 as the destination port to the server. The source port on the client side is port 68.

Once the DHCP server is located on the network, the server replies with an Offer datagram. The offer includes all of the configuration parameters the client will need, based on how the DHCP server has been configured.

If a system already has an IP address it wants to request, it will skip the discovery and offer and move to the DHCP, which is the next step for a system that has gone through the first two steps. To save time, the client system will try to reuse an IP address it already knows about—the last IP address it received from a DHCP server. For systems that are fixed, meaning they are desktops that don't move, this will work well since the IP address likely won't change much, if ever. Systems that are more mobile are more prone to having changes of IP addresses because they are on different networks, depending on where they have been. As a result, the IP address they request may have nothing at all to do with the IP network they are on, so the DHCP server will need to provide them with another one, rather than just acknowledging the one they already have.

When the DHCP server responds, it does so with a DHCP ACK message, indicating that the configuration that has been requested, based either on the offer or on the existing configuration, is acceptable to the server. You can see a portion of this message in Figure 2-13. The ACK message from the DHCP server confirms the requested IP address since, in this case, the address was requested on the same network as the system previously had an address on. In addition, the DHCP server provided a number of other configuration settings. Figure 2-13 shows some of these settings.

```

> Bootp flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0 (0.0.0.0)
Your (client) IP address: oliver.local (172.30.42.13)
Next server IP address: 172.30.42.1 (172.30.42.1)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: Apple_b7:2c:89 (f4:5c:89:b7:2c:89)
Client hardware address padding: 00000000000000000000
Server host name not given
Boot file name not given
Magic cookie: DHCP
▼ Option: (53) DHCP Message Type (ACK)
  Length: 1
  DHCP: ACK (5)
▼ Option: (1) Subnet Mask
  Length: 4
  Subnet Mask: 255.255.255.0
▼ Option: (2) Time Offset
  Length: 4
  Time Offset: (0s) 0 seconds
▼ Option: (3) Router
  Length: 4
  Router: 172.30.42.1 (172.30.42.1)
▼ Option: (23) Default IP Time-to-Live
  Length: 1
  Default IP Time-to-Live: 64
▼ Option: (51) IP Address Lease Time
  Length: 4
  IP Address Lease Time: (3600s) 1 hour

```

Figure 2-13: DHCP ACK message.

The settings are provided in options, as you can see. The options are defined by various RFCs and each option is variable length, based on the needs of the option. Some of the options may include the subnet mask, the router address, and the domain name server. On Windows networks, the DHCP server may include additional options for Windows-specific parameters including a name server for Windows naming. On UNIX networks, a system may get an option for an X-Windows server indicating where applications can display windows across a network. This is just a small sample of the different parameters that may be provided to a client. There are dozens of options defined that can be passed to the client.

NOTE DHCP is based on the bootstrap protocol (Bootp) that was developed as a way to provide an IP configuration to diskless workstations that had no ability to store an IP address. Bootp provides the IP address based on a stored MAC address.

Support Protocols (ARP)

We have gone through the process of how to look up IP addresses from FQDNs but that's only good if you need to know the IP address. You would only use the IP address if you were looking to route your packet across the Internet or just between two different networks internal to your corporate network. If you are already on the target network and you are looking to get the frame to its destination, you need to know the MAC address. There is no database that stores IP address to MAC address information. Instead, you simply ask the network. The Address Resolution Protocol (ARP) is a helper protocol that is used to ask for the MAC address that belongs to a particular IP address. ARP is a very simple protocol that is very lightweight. IPv6 has an equivalent protocol called the Neighbor Solicitation Protocol.

When your system needs to fill in layer 2 information for a frame in order to get it out on the wire, it would use an ARP request. You can see an example of the ARP request in Figure 2-14. The request has information about the protocols in use like Ethernet and IP. It also includes an Opcode to indicate that it is a request. Because it is a request, the target MAC address field is empty. The source MAC and IP address are filled in with the requesting system.

```

▶ Ethernet II, Src: Apple_b7:2c:89 (f4:5c:89:b7:2c:89), Dst: 172.30.42.1 (44:e1:37:f4:f2:34)
▼ Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: Apple_b7:2c:89 (f4:5c:89:b7:2c:89)
  Sender IP address: oliver.local (172.30.42.13)
  Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
  Target IP address: 172.30.42.1 (172.30.42.1)

```

Figure 2-14: ARP request.

The response to this request would come from the system that owns the IP address being asked about. It would have the Opcode of 2 for a Reply and the requested MAC address would be filled in. If you are looking at these requests and responses using network diagnostic tools, the ARP exchange is often short-handed as who-has and is-at. The who-has is the request for information and the is-at is the response.

There is an issue with ARP, though. There is no validation for the information that is being provided. As a result, anyone can pretend to be anyone else. Of course, if you are relying on the operating systems to just do their job, they are always going to respond with the correct information. Since systems commonly cache information about IP address and MAC address resolution in an ARP table to avoid having to ask again, an attacker can circumvent the whole process by just telling everyone that a target IP address is-at the MAC address belonging to the attacker. If the attacking system keeps sending that information out on a regular basis, the systems on the network will all store that information and never ask. As a result, anyone wanting to communicate with an IP address will address the layer 2 frame with the attacker's MAC. This means the attacker will receive all the traffic that was supposed to go to the victim.

This doesn't work for very long if the victim stops getting anything at all on their system. If the network appears to be down because all of the frames are being sent to the attacker, the attacker won't get much information that is really useful. Additionally, the victim may quickly notice their system isn't working very well. Instead, what the attacker does is forward the hijacked frames onto the victim. The attacker gets to see all of the traffic without completely disrupting the network communications of the victim. This process is called ARP spoofing. It does require that the attacker be on the physical network or at least have access to a system on the physical network.

Summary

Entire books are written about the topic of networking. This was a very quick breeze through some of the important protocols that you would run across regularly as a forensic investigator. Since you are working with networks and network protocols as a forensic investigator, you need to know how the protocols work so you will know what you are looking at when you are investigating.

When we talk about networking, it's sometimes easier to have a model to refer to so you can talk about functionality more easily rather than getting it confused by talking about specific protocols. Fortunately, we have two models we can use to specifically talk about functionality. Both of these models use layers to separate the functions used in the process of communications. One of them is specific to a set of protocols and the model or architecture was designed as the protocols were being developed. This is the TCP/IP or DoD model and it has just four layers: data link, internet, transport, and application.

The second model often used to describe network functionality is the OSI Reference Model. The OSI model was developed separately from any protocol and was only created as a way of describing

communications conceptually. The OSI model has seven layers: physical, data link, network, transport, session, presentation, and application.

The TCP/IP model maps into the OSI model. The data link layer from the TCP/IP model is represented in the physical and data link layers in the OSI model. Similarly, the session, presentation, and application layers in the OSI model are represented in the application layer in the TCP/IP model.

If you want to know anything about any of the protocols discussed in this chapter, the very best place to go is to the Request for Comments (RFC) document. The Internet Engineering Task Force manages the entire RFC process and its website is where you can locate any of the RFCs starting from the very beginning of the Arpanet in 1969. Understanding the language of the RFCs, the specific words and phrases used in them to indicate how the protocols are meant to work and what features are necessary versus nice to have, will help you to read and interpret them better. They are, after all, descriptions of protocols. This means they aren't implementation specifics, but instead are guides to implementing the protocols so your implementation will work with someone else's implementation.

You may have noticed in the discussion about the different protocols that the OSI "layers" were referenced. For example, layer 2 refers to a local network, where addressing is done using the Media Access Control (MAC) address. The layer 2 header, including the MAC address, will get removed any time the frame passes across a network or layer 3 device. The process of adding headers is called encapsulation and that will get done every time a message gets passed from one layer to another.

In order to get data off your local area network to another network, you need an IP address. Transmitting packets from one network to another is done using a process called routing. The routing is managed by network devices that understand routing, commonly called routers. These routers will get messages through the network understanding how best to get to the destination network. IP is a best-effort service and handles the process of breaking up datagrams or segments into packets, based on the maximum transmission unit of the underlying network medium. If the datagram or segment has to be fragmented, IP has a header field to indicate that the packet is a fragment and where the fragment goes in the entire segment or datagram.

Above the network layer is the transport layer where TCP and UDP live. TCP is a connection-oriented protocol. Systems create a connection using the three-way handshake before they can start to send data. The three-way handshake, using two TCP flags—SYN and ACK—ensures that both systems are there, available on the particular port and responding. Ports are used to allow for multiplexing. Without ports, systems would be limited to a single application since every communication would come into the system without a way of addressing between the different listening applications. Ports are essentially used to redirect network communication to an application that is meant to handle the network communication. The other commonly used transport protocol is UDP. Where TCP is connection-oriented, offering guaranteed delivery, UDP is connectionless and there is no guarantee offered by the protocol for delivery. All the application using UDP gets is the best-effort delivery of IP.

There are two supporting protocols that you will see a lot if you are looking at network communications. The first is the Dynamic Host Configuration Protocol (DHCP). DHCP allows systems to not require specific network configuration. Instead, the system requests its configuration and a

DHCP server will provide the network configuration for the system. DHCP will not only provide the IP address and subnet mask, but also the router address, time server information, and a number of other configuration parameters that may be necessary based on the needs of the network the system is on. The other support protocol that is far more essential than DHCP, since some networks will use static network configurations, is the Address Resolution Protocol (ARP). ARP is used to translate IP addresses to MAC addresses. Although you can look up an IP address from a hostname using the Domain Name System (DNS), there is no database associated with mapping IP addresses to MAC addresses. Systems on the network use ARP in order to let the systems do the mapping. A system needing a MAC address asks the network by providing an IP address. The system owning that IP address responds with the MAC address.

All of this foundation will serve you well as we start to really dig into network communications in a very hands-on way in the next chapter. We will be capturing real packets and pulling them apart to see how all of the protocols work more closely.

References

Crocker, S., "Host Software," RFC 1, DOI 10.17487/RFC0001, April 1969, <<http://www.rfc-editor.org/info/rfc1>>.

Day, J., & Zimmermann, H. (1983). The OSI reference model. *Proceedings of the IEEE Proc. IEEE*, 71(12), 1334-1340. doi:10.1109/proc.1983.12775

Droms, R., "Dynamic Host Configuration Protocol," RFC 1531, DOI 10.17487/RFC1531, October 1993, <<http://www.rfc-editor.org/info/rfc1531>>.

Postel, J., "Internet Protocol," STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<http://www.rfc-editor.org/info/rfc791>>.

Postel, J., "User Datagram Protocol," STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<http://www.rfc-editor.org/info/rfc768>>.

Postel, J., "Transmission Control Protocol," STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<http://www.rfc-editor.org/info/rfc793>>.

3

Host-Side Artifacts

In this chapter, you will learn about:

- Network-based artifacts that operating systems retain
- Tools that can be used to extract those artifacts
- Protocols and ports that make up services offered over the network
- Connections and the states that connections can have

He sits at his computer, typing away, looking for the next system on the network to compromise. He is talking to his target over the network and is aware that a savvy user could identify the communication stream. Because his target is a computer protected by a corporate firewall, he wasn't able to set up a listening service. In this case, he would only do that on other systems on this network. He can get access to them through this computer more easily than having all of them trying to communicate out. The great thing about firewalls for his purposes is that they typically allow network traffic out without much in the way of restrictions. This is especially true if what is being transmitted looks an awful lot like a commonly used protocol over well-known ports.

One challenge to this approach is that a savvy user may be able to detect his communication. He has done his best to cover his tracks but there is always a possibility of being caught. For the moment he is safe, but because every passing day continues to leave the door to his detection open, he works fast and tries to be smart, restricting the network traffic to a minimum where he can. If what he does is mostly during off-hours while the user is home and entirely unaware, avoiding detection is quite a bit easier.

Network forensics can take advantage of the fact that every operating system has to keep track of network communications. When the operating system keeps track of something, it's possible to extract that information. Most operating systems provide programs that are capable of presenting information about network communications, but not all users are aware of these programs and not all of them are as easy to use as a typical application is. Most are command-line-based, which may scare some people. Once you get the hang of how to use them and how to read their output, though, they are invaluable.

In order for systems to communicate, one of those systems has to be listening. That is done with a type of program that is commonly called a *service*. Services are implemented differently on different operating systems but the end result is the same—when a network service is running, a program is listening for network communication to arrive. Understanding more about those networks and how to determine what is running is important.

Attackers may use something called a *rootkit* to help disguise their existence on the system. A rootkit may open up a network service called a *backdoor*, allowing the attacker to more easily get into the system remotely at any time. Backdoors are useful because, if the service the attacker exploited was later patched, the attacker wouldn't be able to continue to use that way in. So, a backdoor allows continued access, regardless of what happens to the original entry vector. A rootkit may also install programs that will hide the existence of the attacker. This may include overwriting the programs that are included with the operating system that will tell you what is being done on the network. Fortunately, we don't have to rely on the operating system for these sorts of tools. Third-party programs are available that don't rely on anything having to do with the programs and libraries provided by the operating environment.

Services

Imagine that you want to read the latest news about your favorite mobile operating system. Maybe Google is working on a new one and you need to know the release date. What do you do? You might use Google Chrome to connect to Google to search for a page that has information about the release date. What is happening is that your browser (Chrome) is making a connection to a service on a system on Google's network. This service is a program that takes Hypertext Transfer Protocol (HTTP) requests on TCP port 443. HTTP is the protocol being used but port 443 indicates that it has been encrypted. Port 80 is commonly for unencrypted communication. Ultimately, a service is just a program that runs, mostly quietly, in the background waiting for connections from client programs across the network. Any computer system may have a number of services running and the majority of them will start when the computer boots. The operating environment knows about these services and starts them automatically when the computer starts up.

NOTE Most programs you use run in the foreground, meaning you can see them running. A program running in the background commonly can't be seen. In the case of services, there is usually no visible component. The operating environment starts the service and gives it processor time to perform whatever tasks it is responsible for.

Each operating system will engage with its services in different ways, both from an administrative perspective as well as how the services are handled by the operating system. From a management perspective, perhaps Windows is the easiest to understand. Windows also offers the most options in terms of how services are started up. To make it easier to see this and understand it, take a look at the screen capture of the Services applet from Windows 10 in Figure 3-1. You can see a list of some of the services available on the system.

Name	Description	Status	Startup Type	Log On As
ActiveX Installer (AxInstSV)	Provides User Account Control validation ...		Manual	Local Syste...
AllJoyn Router Service	Routes AllJoyn messages for the local All...		Manual (Trig...	Local Service
App Readiness	Gets apps ready for use the first time a us...		Manual	Local Syste...
Application Identity	Determines and verifies the identity of an ...		Manual (Trig...	Local Service
Application Information	Facilitates the running of interactive appli...	Running	Manual (Trig...	Local Syste...
Application Layer Gateway Service	Provides support for 3rd party protocol pl...		Manual	Local Service
Application Management	Processes installation, removal, and enu...		Manual	Local Syste...
AppX Deployment Service (AppXSVC)	Provides infrastructure support for deploy...		Manual	Local Syste...
Background Intelligent Transfer Service	Transfers files in the background using idl...	Running	Automatic (D...	Local Syste...
Background Tasks Infrastructure Service	Windows infrastructure service that contr...	Running	Automatic	Local Syste...
Base Filtering Engine	The Base Filtering Engine (BFE) is a servic...	Running	Automatic	Local Service
BitLocker Drive Encryption Service	BDESVC hosts the BitLocker Drive Encrypt...		Manual (Trig...	Local Syste...
Block Level Backup Engine Service	The WBENGINE service is used by Windo...		Manual	Local Syste...
Bluetooth Handsfree Service	Enables wireless Bluetooth headsets to ru...		Manual (Trig...	Local Service
Bluetooth Support Service	The Bluetooth service supports discovery ...	Running	Manual (Trig...	Local Service
BranchCache	This service caches network content from...		Manual	Network S...
CDPSvc	CDPSvc		Manual	Local Service
Certificate Propagation	Copies user certificates and root certificat...		Manual	Local Syste...
Client License Service (ClipSVC)	Provides infrastructure support for the Mi...	Running	Manual (Trig...	Local Syste...
CNG Key Isolation	The CNG key isolation service is hosted in...	Running	Manual (Trig...	Local Syste...
COM+ Event System	Supports System Event Notification Servi...	Running	Automatic	Local Service
COM+ System Application	Manages the configuration and tracking ...	Running	Manual	Local Syste...
Computer Browser	Maintains an updated list of computers o...	Running	Manual (Trig...	Local Syste...
CoreMessaging	Manages communication between syste...	Running	Automatic	Local Service
Credential Manager	Provides secure storage and retrieval of cr...	Running	Manual	Local Syste...
Cryptographic Services	Provides three management services: Cat...	Running	Automatic	Network S...

Figure 3-1: Windows Services applet.

On a Windows system, a service is a program that has no user interface component, so it isn't visible to the user. Additionally, a Windows service is a program that is written in a particular way so that it responds to interactions from the Service Control Manager. A service is expected to be able to start and stop, based on messages from the Service Control Manager. Services also have multiple startup options. The Service Control Manager may start the service automatically when the system starts; it may also start automatically with a delayed start or it may be set to start manually. A service that starts manually either has to be started by a user through the Services system utility or it can be started if another system requires it as a dependency in order to operate correctly. You can see the Properties dialog box for the Block Level Backup Engine Service with the different startup options showing in Figure 3-2.

This is not to say that programs that listen for network connections have to be Windows services, but if you want a program that starts automatically at boot and runs quietly in the background listening for requests, it has to be a Windows service. One way to check whether a service is installed and running is to simply open the Services utility and look for a Status indication, as demonstrated in Figure 3-1. Services that aren't running have no entry in the Status column.

Unix-like operating systems have programs that can run in the background as well. A service on a Unix-like system is typically called a *daemon*, named, in part, for the benevolent spirits from Greek mythology. Since macOS (formerly Mac OS X) is based on a Unix-like operating system,

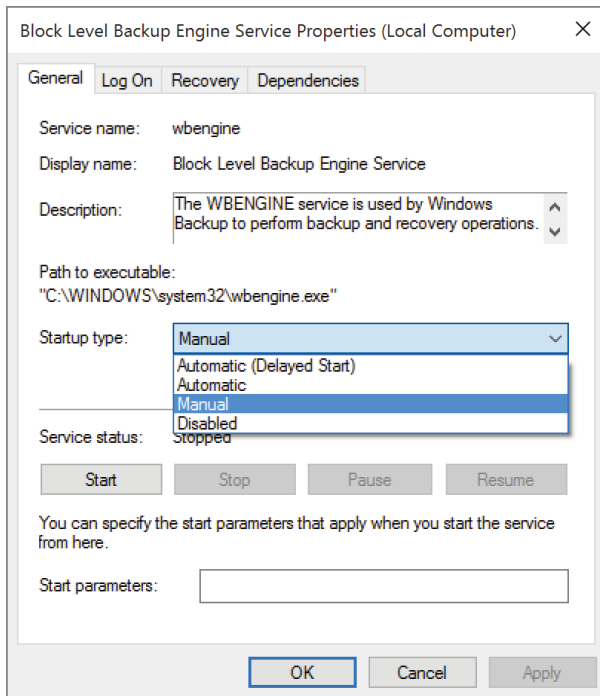


Figure 3-2: Windows Service Properties dialog.

it uses daemons. However, macOS doesn't use the same means to start services as Unix-like operating systems traditionally do. The macOS startup program is `launchd`. It serves the same function as a program like `init` on a traditional Unix-like operating system.

NOTE The spelling of daemon comes from the Greek, but the idea of a helper sitting quietly doing work without being seen comes from Maxwell's Demon. Maxwell's Demon was a thought experiment related to the Second Law of Thermodynamics and conceived by James Clerk Maxwell in 1867. Maxwell's Demon controlled a door between two gas-filled chambers, opening it to allow fast moving (hot) molecules to pass from one chamber to the other.

Services on a macOS system are stored as property list files in `/Library/LaunchDaemons`, as shown in Figure 3-3. The program that is the actual service process is stored as a string in the `ProgramArguments` key. You can see that the program in the `com.microsoft.autoupdate.helpertool` launch daemon is `/Library/PrivilegedHelperTools/com.microsoft.autoupdate.helpertool`. It may go without saying but this is a daemon that runs in the background, helping Microsoft products on this system stay up to date.

```

[kilroy@oliver:~$ ls /Library/LaunchDaemons/
com.adobe.fpsaud.plist
com.google.keystone.daemon.plist
com.microsoft.autoupdate.helpertool.plist
com.microsoft.office.licensingV2.helper.plist
com.oracle.java.Helper-Tool.plist
org.wireshark.ChmodBPF.plist
[kilroy@oliver:~$ cat /Library/LaunchDaemons/com.microsoft.autoupdate.helpertool.]
plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>com.microsoft.autoupdate.helpertool</string>
  <key>MachServices</key>
  <dict>
    <key>com.microsoft.autoupdate.helpertool</key>
    <true/>
  </dict>
  <key>Program</key>
  <string>/Library/PrivilegedHelperTools/com.microsoft.autoupdate.helperto
ol</string>
  <key>ProgramArguments</key>
  <array>
    <string>/Library/PrivilegedHelperTools/com.microsoft.autoupdate.
helpertool</string>
  </array>
</dict>
</plist>

```

Figure 3-3: macOS launch daemons.

NOTE A property list is a collection of key/value pairs. The key is referred to in order to retrieve the value so a program can retrieve stored information about the user, environment, or other maintained configuration of the program. For example, if a program wants to store the value of the location of the primary application window, it may have two keys, XVal and YVal, to indicate the origin of the window, and two more keys, Xsize and Ysize, to indicate the number of pixels the window requires. This lets the program orient itself in the same location when it is run again. The key value pairs might be:

```

XVal: 10
YVal: 50
Xsize: 1000
Ysize: 740

```

Whereas Microsoft uses the Windows registry to store configuration settings, Apple uses property list files, sometimes called “plists.”

Linux systems may have two different ways to manage services, depending on what distribution they are. Traditional Unix systems use `init` as the process that starts everything and services that start at boot time are managed by `init`. Services that are managed by `init` use shell scripts for their

management: start, stop, restart, status. Similar to Windows and the Service Control Manager, these scripts are expected to respond to particular messages. These messages are parameters that are passed on a command line into the script. In Figure 3-4, you can see a listing of the `/etc/init.d/` directory where all of the startup scripts are kept on a system that operates using `init`. At the bottom of Figure 3-4 is a section of the service script for the Secure Shell (SSH) service. This is the piece of the script that responds to a request for status. Similar sections handle the start and stop requests. In the case of this script, it also handles a number of other requests like restart and reload, though not all services will handle other management functions.

NOTE Programs like `init` and `systemd` are superprocesses (or supervisory processes) that are responsible for managing all the processes running on the system. If you were to show a genealogy of processes on a Linux or Unix-like system where you started with the first process and flowed to child processes and the children of children, the very top of the tree would be `systemd` or `init`. They are the superprocess because everything begins with them.

```
kilroy@quiche ~ $ ls /etc/init.d/
acpid                irqbalance           procs
alsa-utils           kerneloops           rc
anacron              killprocs            rc.local
avahi-daemon         kmod                 rcS
binfmt-support       lm-sensors           README
bluetooth            loadcpufreq          reboot
bootmisc.sh          lvm2                 resolvconf
brltty               lvm2-lvmetad         rsync
cgmanager             lvm2-lvmpolld        rsyslog
cgproxy              mdm                  saned
checkfs.sh            mintsytem            sendsigs
checkroot-bootclean.sh mountall-bootclean.sh single
checkroot.sh          mountall.sh          skeleton
console-setup         mountdevsubfs.sh     speech-dispatcher
cpufrequtils          mountkernfs.sh       ssh
cron                  mountnfs-bootclean.sh thermalid
cryptdisks             mountnfs.sh          udev
cryptdisks-early      networking            ufw
cups                   network-manager      umountfs
cups-browsed          ntp                  umountfs.sh
dbus                  ondemand             umountroot
dns-clean              openvpn              unattended-upgrades
grub-common            plymouth             urandom
halt                   plymouth-log         uuid
hddtemp               pppd-dns             virtualbox-guest-utils
hostname.sh            prltool             x11-common
hwclock.sh            prl-x11
kilroy@quiche ~ $ tail -n 11 /etc/init.d/ssh
status)
    check_for_upstart 1
    status_of_proc -p /var/run/sshd.pid /usr/sbin/sshd sshd && exit 0 || exit 1
;;

*)
    log_action_msg "Usage: /etc/init.d/ssh {start|stop|reload|force-reload|restart|try-restart|status}" || true
    exit 1
esac

exit 0
```

Figure 3-4: Linux service scripts.

Not all Linux systems use `init`, however. A newer system super process that would be responsible for starting and managing system services is named `systemd`. There are a number of reasons for moving to a different super process and one of those has to do with the complexities associated with system startup and services conflicting or starting out of order. On a traditional `init`-based system, the services are started in alphanumeric order based on how they are named in the directory where they are located. This requires a special naming convention for those directories. Commonly, the directories would be based on run level, which is a numeric designation based on the functionality expected from the system—single-user, networking, graphical user interface and networking, and so on. In those directories would be a series of links to the scripts in `/etc/init.d/`. The links would be named based on whether they were intended to be used to start up or kill the service. Startup scripts have names starting with `S`, followed by a number indicating the order in which the script was to start. If your service relied on another service to be operating, your service would need to have a higher number.

In `systemd`-based systems, each service gets a configuration file indicating dependencies. In other words, you can specify which services have to be operating for yours to function correctly. Listing 3-1 shows an example of one of those startup configuration files for the postfix mail service. The `systemd` configuration also allows the service to indicate whether other services would conflict. As an example, this configuration file indicates that `sendmail` and `exim`, both also mail transfer agents, would conflict, so if they are running, this service shouldn't even bother trying to start because they would try to listen on the same port and two applications can't listen on the same port.

Listing 3-1: Postfix Systemd Configuration File

```
[Unit]
Description=Postfix Mail Transport Agent
After=syslog.target network.target
Conflicts=sendmail.service exim.service

[Service]
Type=forking
PIDFile=/var/spool/postfix/pid/master.pid
EnvironmentFile=-/etc/sysconfig/network
ExecStartPre=-/usr/libexec/postfix/aliasesdb
ExecStartPre=-/usr/libexec/postfix/chroot-update
ExecStart=/usr/sbin/postfix start
ExecReload=/usr/sbin/postfix reload
ExecStop=/usr/sbin/postfix stop

[Install]
WantedBy=multi-user.target
```

No matter what the operating system is, not all services relate to network communications. Many services are simply about providing background functionality to the system and other services. Just because you see a number of services on a system doesn't mean that you are looking at a lot of programs that are providing network connections or even potential network connections to other systems.

Connections

When it comes to network connections, we are primarily talking about TCP communication. TCP communication can be thought of as *stateful* because there is always a state to any communication. When you hear the term stateful, you may think firewalls since one type of firewall is a stateful firewall, meaning that it keeps track of the state of network communications between two systems. Stateful firewalls not only have to keep track of the state of TCP communication, which can be discerned by looking at any given packet or set of packets that are passing between two systems, but they will also generally keep track of the state of other protocols simply in order to make determinations about what traffic should and shouldn't be allowed through the firewall. However, before we go any further, we should talk about the different states.

First, we have a network service. This is a program that is listening on a network interface. Programs that have TCP listeners have to bind to a port and IP address to announce their intention to start listening for network traffic. This essentially registers an endpoint with the operating system kernel so if any network communication comes in on the registered port, the kernel will know where to send the data. Once the program has registered the port it wants to listen on by binding to that port, it can start to listen for connections. At this point, the port is in a listening state. As you see in the “Tools” section later in this chapter, the port and program can both be demonstrated to be in a listening state.

Every connection in TCP exists as a particular state, shown in Table 3-1. As we start to look at some of the tools that provide information about the state of connections, it may be helpful to understand what each state means and how we get to that particular state.

Recall from Chapter 2 that the first communication is a SYN message. Once the program has received the SYN message, the communication is in a SYN-RCVD state, indicating that the SYN message has been picked up by the operating system. The other end of the communication is in a SYN-SENT state at that point. Because we are only partway to our final goal of a completed connection, the communication stream is considered half-open while the operating system waits for the final leg of the three-way handshake. During this period of time, it will hold out a small piece of memory assuming that the connection will complete. Once the three-way handshake is completed, the communication is considered to be ESTABLISHED.

NOTE TCP is sometimes presented as a state machine. A *state machine* is a way of thinking about systems. In the case of TCP, each connection can be in only one state at a given time. When an event happens, the state machine, TCP in this case, may transition to another state. However, much like a flowchart, the transitions between states is understood ahead of time. As an example, you wouldn't transition from FIN-WAIT to SYN-SENT because that's not how the state machine (TCP) has been developed to behave.

In the process of tearing a connection down, there are also a number of different states. If a system sends a FIN to the other end of the conversation, it must then wait for that other end to send a FIN

Table 3-1: TCP Connection States

State	Description
CLOSED	The port has no program waiting to communicate on it. Because there is no program associated with it, there are no communication streams.
LISTENING	The application has bound to a port and is waiting for connections.
SYN-SENT	The operating system has sent a SYN message to the other side so this would be on the client side because clients initiate requests to services in most cases.
SYN-RCVD	A SYN message has been received and the communication stream is considered half-open at this point until an ACK has been received in response to the SYN-ACK that would be sent back to the client.
ESTABLISHED	Once the three-way handshake has been completed, the communication stream is considered to be ESTABLISHED.
FIN-WAIT	This may be waiting for an ACK to a FIN that has been sent or it may be the case that a FIN has been sent and an ACK received and the system is now waiting for the FIN on the other side to be sent.
CLOSE-WAIT	The system in question has received a FIN and sent back an ACK. It is now waiting for the application on its side to complete in order to be able to send a FIN.
CLOSING	This is the state where one side is completely torn down with a FIN and corresponding ACK but the other side has sent a FIN without receiving an ACK back yet.
TIME-WAIT	A timer is responsible for this condition. Once you receive a FIN message and send back an ACK, the system has no way of knowing that the ACK has been received without getting an ACK back and that would turn into an infinite loop of ACK messages. Instead, the system waits a specified amount of time, holding the connection in a TIME-WAIT state just to be safe. Once the timer has elapsed, the communication is finally completely torn down.

of its own before the connection is fully torn down. While it is waiting for that FIN, the connection will be in FIN-WAIT state (if this is the first side to tear down, it will be FIN-WAIT-1). If a system receives a FIN from the other side but is waiting for the application handling the communication to complete and request a FIN, the connection is in CLOSE-WAIT state because it is half-closed and waiting to start the process of fully closing the communication channel. Once the application says okay, we're done; the system can send its own FIN. During the brief duration that the FIN has been sent but the corresponding ACK hasn't been received, the connection is in CLOSING state. A connection will also be in a TIME-WAIT state. This is the brief duration, and there is a timer specified by the protocol definition that governs this, while the connection is being torn down and the system is waiting to make sure the ACK in response to a FIN on the other side has been received. When the other party sends their own FIN and are waiting for a response, that is FIN-WAIT-2. Unlike the three-way handshake, the teardown of the connections recognizes that both are capable of communication so each end has to originate its own teardown.

While an application is running and it has been bound to a port, that port is considered to be in a LISTENING state but the other side of that is CLOSED. If a port has no program listening, the port is simply considered CLOSED. The vast majority of ports on a system—recall that there are 32,768 ports available for TCP communication—are in a CLOSED state.

Keep in mind that while a communication stream can be in an ESTABLISHED state, the application and therefore the port can and will still be listening because the operating system allows for multiple communication streams to a given port. Any TCP-based communication is described by a four-tuple, meaning there are four pieces of information you need. Any connection is described by the ports and IP addresses on both ends. For every communication pathway, there is a source and destination IP address as well as a source and destination port. Because communication is bidirectional, one system's source port is the other system's destination port. Each side will have a four-tuple describing the connection with the source and destinations swapped on each end.

While UDP traffic is entirely stateless, some systems like firewalls or any device that has a firewall built into it will have to maintain a state table. Once a UDP message has gone out, there is a chance a response will be received. If you are blocking UDP ports except for responses to messages that have gone out, you need to maintain a table to keep track of the four-tuple in order to match source and destination IPs and ports to make sure the return traffic is allowed back in. While UDP itself is stateless, stateful firewalls will have to keep track of the state of network communication just to make sure everything continues to work as expected.

While this might seem to be largely theoretical—because the communication happens so quickly you wouldn't be able to tell what state any communication is in—programs are available that will show you the state of network communications. This is a crucial artifact because network communications are kept by the operating system. This means the information about those communications is stored in memory. All it takes is the right tool to pull that information out and the ability to read it correctly once it is out.

Tools

As you are looking for artifacts, you will likely be working on a live system. Once you have shut down the system, all of the network artifacts go away. Network information is retained in memory and not on disk because everything is in operating system data structures. The only way to retain any of the information for offline analysis is to get a memory capture. Otherwise, you have to look at the system while it is up and running. There are challenges with this, one of which is that you may be detected if what you are investigating is a system compromise. Once you are poking around on the system, your adversary can see what you are doing and potentially adjust accordingly. This makes it harder to investigate what is really happening. Subsequent chapters look at investigating from outside the system; in this chapter, the only way to get to what the host knows is to actually be on the host.

The second challenge associated with performing an investigation on the host itself is that your tools may be compromised. Each system comes with a number of built-in utilities that are used for diagnostics while the system is operating. Because these utilities are built in, they can be compromised

by an attacker so any output they provide may be suspect. Fortunately, we aren't limited to just the utilities that are provided with each operating environment. A number of third-party utilities are available as well. This is especially important if the third-party utilities can be run independently without making use of any of the installed libraries on the system.

netstat

Netstat is a command-line utility that provides a lot of network information. If you need to know what the routing table looks like, meaning what IP address packets going off the local network need to be sent to based on their destination address, you would use netstat. In addition to the routing table, netstat provides a list of all of the open communication streams, as Figure 3-5 shows. The `netstat -a` command displays all of the existing and active communications including their state. In the top part of the figure, you can see that a large number of communications are in the `ESTABLISHED` state. This means that they are communicating over TCP and they have completed the three-way handshake. It could mean that they are actively transmitting or receiving or it could simply be that there is an open connection that hasn't been torn down because there is an expectation of further messages being sent back and forth.

```
kilroy@oliver:~$ netstat -a
```

Active Internet connections (including servers)					
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	(state)
tcp6	0	0	2601:19b:c601:ec.53990	lga15s43-in-x0e..https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53989	qj-in-x5d.1e100..https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53987	qu-in-x6d.1e100..imaps	ESTABLISHED
tcp4	0	0	172.30.42.16.53976	geoplookup-lb.e.https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53975	upload-lb.eqiad..https	ESTABLISHED
tcp6	120	0	2601:19b:c601:ec.53974	text-lb.eqiad.wi.https	ESTABLISHED
tcp6	120	0	2601:19b:c601:ec.53973	text-lb.eqiad.wi.https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53972	text-lb.eqiad.wi.https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53971	lga15s47-in-x0e..https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53969	lga15s48-in-x0d..https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53968	lga15s43-in-x0e..https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53967	lga15s45-in-x0e..https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53966	lga15s45-in-x0e..https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53923	qg-in-x6d.1e100..imaps	ESTABLISHED
tcp4	0	0	172.30.42.16.53896	whub49.webhostin.imaps	ESTABLISHED
tcp4	0	0	172.30.42.16.53873	199.16.156.120.https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53868	qh-in-x6c.1e100..imaps	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53838	2603:1036:404:67.https	ESTABLISHED
tcp4	0	0	172.30.42.16.53592	whub49.webhostin.imaps	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53582	edge-star-mini6-.https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.53423	qb-in-x7d.1e100..5223	ESTABLISHED
tcp4	0	0	*,*	*,*	CLOSED
tcp4	0	0	172.30.42.16.53380	17.172.232.214.5223	ESTABLISHED
tcp4	0	0	172.30.42.16.52889	17.249.10E.10.5223	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.63680	edge-star€-shv-0.https	ESTABLISHED
tcp6	0	0	2601:19b:c601:ec.63674	qt-in-xbc.1e100..5228	ESTABLISHED
tcp4	0	0	localhost.ipp	*,*	LISTEN
tcp6	0	0	localhost.ipp	*,*	LISTEN
tcp46	0	0	*,19420	*,*	LISTEN
tcp4	0	0	*,kerberos	*,*	LISTEN
tcp6	0	0	*,kerberos	*,*	LISTEN
tcp4	0	0	*,microsoft-ds	*,*	LISTEN
tcp6	0	0	*,microsof	*,*	LISTEN
tcp4	0	0	*,afpovertcp	*,*	LISTEN
tcp6	0	0	*,afpovert	*,*	LISTEN

Figure 3-5: Connection list from netstat.

This is just one of the types of information that `netstat` can provide. To display the routing table, you can use the `netstat -r` command. That will provide the list of all of the network routes that your particular system knows about. In most cases on desktop systems, you will have a number of routes to devices on your local network but there will really only be a single route to other networks. The destination in that case will be 0.0.0.0, meaning that any network that isn't local goes to the gateway (router) that is specified by IP address. `netstat -r` and `netstat -a` (shown in Figure 3-5) display human-readable addresses where possible. This means that `netstat` will do lookups prior to providing any output. The addresses shown in the `Foreign Address` column in the list of active communications are primarily text. Because the column isn't wide enough to support the length of the hostname, much of it has been truncated but what is shown is a fully qualified domain name (FQDN) or at least as much of one as can fit into the column.

Having `netstat` perform lookups can be time consuming. To save time—which would be long if you had unplugged the system from the network since you'd be waiting for a lot of network timeouts—you can add `-n` to the command line. This tells `netstat` to use numeric output and not to do any of the lookups. Addresses are not the only part of the output that get looked up. Port numbers, which are addresses for layer 4, also get translated into names that may mean more to someone than just the number. You may not know off the top of your head, for instance, that `https` is port 443 or that `imaps` is port 993. Both of those are referenced in the output in Figure 3-5.

Another feature of `netstat` is its ability to just output network statistics related to the amount of traffic that has come across or out of the system. You will get different statistics based on the operating system you are on. Although you can run `netstat` on Windows, Linux, and macOS, the different implementations will get you different sets of data. As an example, Figure 3-6 shows a small amount of the data that is available using `netstat -s` on a macOS system. Detailed statistics are available for different protocols like TCP, UDP, IP, ICMP, IPv6, and others.

On a Kali Linux system, the output is very different and not as detailed using the same `netstat -s` command. You can see in Listing 3-2 that the limited amount of network traffic notwithstanding, the output looks very different on this system. The `netstat` implementation on Linux systems is different from that on other systems. As an example, you can use `-listening` on a Linux system and get a list of just the programs that are listening.

Listing 3-2: Linux `netstat -s` Output

```
Tcp:
 44 active connections openings
 6 passive connection openings
 4 failed connection attempts
 1 connection resets received
 1 connections established
102047 segments received
60609 segments send out
125 segments retransmitted
 0 bad segments received.
11 resets sent
```



```

kilroy@oliver:~$ netstat -s
tcp:
    3174427 packets sent
        958511 data packets (398434793 bytes)
        24547 data packets (15690531 bytes) retransmitted
        0 resend initiated by MTU discovery
        1836417 ack-only packets (1792 delayed)
        0 URG only packet
        0 window probe packet
        186578 window update packets
        170015 control packets
        0 data packet sent after flow control
        3162566 checksummed in software
            1506278 segments (331691820 bytes) over IPv4
            1656288 segments (174474312 bytes) over IPv6
    3669732 packets received
        882135 acks (for 397538454 bytes)
        211599 duplicate acks
        0 ack for unsent data
        2787882 packets (2620726308 byte) received in-sequence
        17156 completely duplicate packets (2064100 bytes)
        618 old duplicate packets
        0 received packet dropped due to low memory
        5 packets with some dup. data (385 bytes duped)
        29128 out-of-order packets (29842910 bytes)
        0 packet (0 byte) of data after window
        0 window probe
        7252 window update packets
        5100 packets received after close
        14 bad resets
        0 discarded for bad checksum
        3511441 checksummed in software
            1757547 segments (1692047842 bytes) over IPv4
            1753894 segments (1104308119 bytes) over IPv6
        0 discarded for bad header offset field
        0 discarded because packet too short
    86179 connection requests
    6766 connection accepts

```

Figure 3-6: Protocol-based network statistics from netstat.

Using netstat, you can get the list of ports that are in listening mode, as we've seen. Just knowing that a port is open and there is an application listening on it doesn't provide you a lot of information because any program could be listening on that port. The best approach is to get the process that is listening. Even if you see a common port like 80 listening, it could be that the port had been hijacked by a rogue process so it's best to verify. Some implementations will provide you the process identification number and the program name, as you can see in Listing 3-3, which shows the output from a Kali Linux system.

Listing 3-3: Getting Process IDs Using netstat

```

Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp      0      0 127.0.0.1:9390 0.0.0.0:* LISTEN 673/openvasmd
tcp      0      0 127.0.0.1:9391 0.0.0.0:* LISTEN 681/openvassd: Wait
tcp      0      0 127.0.0.1:9392 0.0.0.0:* LISTEN 592/gsad
tcp      0      0 0.0.0.0:22 0.0.0.0:* LISTEN 18979/sshd
tcp      0      0 127.0.0.1:631 0.0.0.0:* LISTEN 683/cupsd
tcp      0      0 127.0.0.1:5432 0.0.0.0:* LISTEN 748/postgres
tcp      0 188 172.30.42.25:22 172.30.42.16:55140 ESTABLISHED 5009/1

```


All of the information that netstat provides is stored in the kernel memory space in specific data structures that the operating system has to keep track of. This information can be retrieved using other tools that can also query the operating system to get the information, but it's not actually written down anywhere on disk. Even on Linux systems, the `/proc` directory will contain some of this information, but the "files" you will find in that directory are not files that are stored on disk. They are ways to access kernel memory and data structures that are stored there. This means that if you want to get this information, you have to do it while the system is running, and netstat is a good way of retrieving it using a single utility.

Though we have so far only looked at macOS and Linux, Windows also has an implementation of netstat that provides much the same sort of information as the implementations on macOS and Linux. However, netstat focuses on just TCP/IP protocols. Windows also uses other protocols, especially to communicate with other Windows systems.

nbstat

In 1983, as networking on personal computers was getting underway, the company Sytek developed an interface for networking called Network Basic Input Output System (NetBIOS) as a complement to the system BIOS that provided a way for programs to get access to the hardware. NetBIOS was developed as an application programming interface (API) so that programs would have a way to get access to the network. Since that time, NetBIOS has been adapted multiple times to allow it to continue to work on different technologies and different operating systems. Currently, the implementation is referred to as NetBT (or NBT), which is short for NetBIOS over TCP/IP. NetBT makes the same API calls available over a different set of network and transport protocols than NetBIOS was originally designed for.

Windows systems have been using NetBT for the past couple of decades as a way to allow Windows systems to communicate with one another for the purpose of file and print sharing, among other reasons. NetBT provides a name service, a session service, and a datagram service. The name service allows systems to find one another on the network. This is separate from the domain name system (DNS), because NetBT was intended to provide local naming rather than naming across the Internet. In that sense, it is similar to the media access control (MAC) address, which would never be communicated beyond a routing device. The fact that NetBT is NetBIOS over TCP/IP means that the naming of systems is expected to be unique over a larger area than originally intended by NetBIOS. It also means that we are effectively carrying NetBIOS inside of TCP/IP, a practice which is commonly called tunneling. The session service is in place to allow systems to establish communication between them.

The nbstat utility is used to get statistics related to NetBT. Using nbstat, you can get name information on your Windows network. Part of the output, as shown in Figure 3-7, is the type of name. This is stored in the last byte of the name because the last byte helps to address the case where the same name may be used for multiple purposes on the Windows network. The last byte, displayed between brackets (`< >`), is shown in hexadecimal.

The last byte may mean that the name represents a workstation, domain controller, master browser, or a file server. You may run across other values if you work on networks that are heavily dependent

```

C:\Users\kilroy>nbtstat -n

Ethernet0:
Node IpAddress: [172.16.144.139] Scope Id: []

        NetBIOS Local Name Table

    Name                Type             Status
    -----
    RICHESSIERCDCC <20>    UNIQUE         Registered
    RICHESSIERCDCC <00>    UNIQUE         Registered
    WORKGROUP        <00>    GROUP          Registered
    WORKGROUP        <1E>    GROUP          Registered
    WORKGROUP        <1D>    UNIQUE         Registered
    00__MSBROWSE__0<01> GROUP          Registered

Bluetooth Network Connection:
Node IpAddress: [0.0.0.0] Scope Id: []

    No names in cache

C:\Users\kilroy>nbtstat -S

Ethernet0:
Node IpAddress: [172.16.144.139] Scope Id: []

    No Connections

Bluetooth Network Connection:
Node IpAddress: [0.0.0.0] Scope Id: []

    No Connections

```

Figure 3-7: nbtstat output.

on Windows components like Windows Active Directory and Windows Exchange Server. In a listing of nbtstat on a large network, you will typically see multiple entries for each name value. The reason for this is that each Windows client may also be a file server or a Messenger client.

The nbtstat utility is capable of displaying all of the names that your local workstation is registered with. Additionally, you can see the cache of the registered names and their associated IP addresses that the system you are on knows about. If there is nothing in the cache, it doesn't mean there aren't other systems on the network; it just means this particular system hasn't seen any communication from them.

The cache will show the IP address and its associated NetBIOS name. This is not the same as any hostname this system may have that's loaded into DNS. Your system can have a NetBIOS name and a DNS hostname and they can be two separate things. The NetBIOS name is the name you provide to Windows so it's the name the system knows about. DNS is configured on a separate system so the Windows system knows nothing directly about that hostname. Matching these two up requires communication between the people responsible for them. That doesn't always happen, but in most cases it doesn't matter, although it's sometimes worth being aware that you could have two systems on the same network sharing the same name. You could also have a system that has one name in

one context, outside of the local network, and another name in the context of the local network. When you are looking for machines based on a name you may have been provided, you should know that you could be looking in two different places for that name.

ifconfig/ipconfig

One of the most fundamental networking utilities is `ifconfig`, short for interface configuration. The purpose of `ifconfig` is to show the IP address configuration for the network interfaces on your system. On Windows systems, this utility is called `ipconfig` (IP configuration), and it also provides other information that you won't get using `ifconfig` on Linux or macOS, including the DNS servers that have been configured on the system. Figure 3-8 shows the output from `ifconfig` on a Linux system. Other Unix-like operating systems, including macOS, will look very similar to this.

```
root@major:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1c:42:70:68:49
          inet addr:172.30.42.25  Bcast:172.30.42.255  Mask:255.255.255.0
          inet6 addr: 2601:19b:c601:ec00:6597:7f6a:8669:34ee/64  Scope:Global
          inet6 addr: 2601:19b:c601:ec00:21c:42ff:fe70:6849/128  Scope:Global
          inet6 addr: fe80::21c:42ff:fe70:6849/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2795692 errors:0 dropped:0 overruns:0 frame:0
          TX packets:222114 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:637146384 (607.6 MiB)  TX bytes:35684542 (34.0 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:150575 errors:0 dropped:0 overruns:0 frame:0
          TX packets:150575 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:179758060 (171.4 MiB)  TX bytes:179758060 (171.4 MiB)
```

Figure 3-8: `ifconfig` output.

In addition to the IP address and associated subnet mask, you can see that the media access control (MAC) address is also displayed. `ifconfig` tells us that our link connection is Ethernet and refers to the MAC address as `HWaddr`, but the MAC address and the hardware address are the same. Additionally, `ifconfig` displays the IPv6 addresses that are associated with the interface, if there are any. A number of parameters that have been set on the interface are also displayed. One of these is the maximum transmission unit (MTU), indicating how large a message can be sent on the wire. In this case, it's 1500 bytes, which is the standard Ethernet MTU. You can also see the statistics associated with the interface, including the number of bytes transmitted and the number received.

By comparison, you can see the output of `ipconfig` on a Windows system in Figure 3-9. On a Windows system, `ipconfig` is used for more than simply displaying IP information that is associated with network interfaces. You could use `ipconfig` to release a network address that had been dynamically configured and you can also renew that address. If you are having problems with your network interface, `ipconfig` may be a way to make sure that you have a good DHCP-assigned address. While

```

C:\Users\kilroy>ipconfig /all

Windows IP Configuration

    Host Name . . . . . : RICMESSIERCDC
    Primary Dns Suffix . . . . . :
    Node Type . . . . . : Hybrid
    IP Routing Enabled. . . . . : No
    WINS Proxy Enabled. . . . . : No
    DNS Suffix Search List. . . . . : localdomain

Ethernet adapter Ethernet0:

    Connection-specific DNS Suffix . : localdomain
    Description . . . . . : Intel(R) 82574L Gigabit Network Connection
    Physical Address. . . . . : 00-0C-29-87-D2-3B
    DHCP Enabled. . . . . : Yes
    Autoconfiguration Enabled . . . . : Yes
    Link-local IPv6 Address . . . . . : fe80::3567:81ce:ba15:3386%4(Preferred)
    IPv4 Address. . . . . : 172.16.144.139(Preferred)
    Subnet Mask . . . . . : 255.255.255.0
    Lease Obtained. . . . . : Monday, August 15, 2016 4:31:34 PM
    Lease Expires . . . . . : Monday, August 15, 2016 5:01:39 PM
    Default Gateway . . . . . : 172.16.144.2
    DHCP Server . . . . . : 172.16.144.254
    DHCPv6 IAID . . . . . : 134220841
    DHCPv6 Client DUID. . . . . : 00-01-00-01-1B-6A-11-DC-00-1C-42-D7-8B-98

    DNS Servers . . . . . : 172.16.144.2
    Primary WINS Server . . . . . : 172.16.144.2
    NetBIOS over Tcpip. . . . . : Enabled

```

Figure 3-9: ipconfig output.

it may be a little more time consuming than other ways of doing the same thing, if you are already at the command line running diagnostics it may be quicker. As noted earlier, you can see in this output that additional IP configuration like the DNS servers are provided as well. You can also dump the local DNS cache by using `/displaydns`.

Although the same information that is available from these utilities is available in other places, you may have instances where you have to go to the command line to get this information. Sometimes graphical interfaces become unstable or unreliable and you need to drop to a command line and just type a command or two to get the information you need.

Sysinternals

In the mid-1990s, a pair of developers named Bryce Cogswell and Mark Russinovich started up a website named `ntinternals` that later became `Winternals`. A number of programs were available on this website and the collection of tools began to grow. These were programs that exposed some of the internals of the Windows operating system in ways that would be useful for developers, system administrators, and now forensic investigators. About a decade after the website began, Microsoft acquired the site, the tools, and Russinovich and Cogswell. The collection of tools is now called

Windows Sysinternals and, other than a password recovery tool that was previously available, all of the tools are available through Microsoft's website.

A large number of highly useful utilities are available. You can download them individually or just download a zip file with the entire suite of utilities. At the time of this writing, 121 programs are available, though some of those are just 64-bit implementations of 32-bit programs that are also in the suite. Though not all of the programs are unique, there is still a significant number and over the years, a fair number of people have found them to be very useful. As a result, we're going to spend some time looking at the programs that relate to host-side network artifacts. As the list continues to grow, we aren't going to cover everything that is of value but instead highlight the ones specifically relevant to network forensics from the standpoint of identifying network connections and their relation to running processes.

NOTE One significant advantage to the Sysinternals tools is that they run standalone, meaning they don't need to be installed in order to get all of the libraries and registry keys created. You can run these from a directory on your system or from an external drive like a USB storage device. This allows you ensure the output from these programs has not been compromised by libraries that have been hijacked by an attacker.

The first program we are going to talk about, shown in Figure 3-10, is TCPView. This graphical user application provides information similar to netstat. One advantage to this view over what is available from netstat is that you can easily sort this view. Additionally, it is updated in real time, which netstat can also do if you provide it an interval of time that you want to elapse before the information is updated. If a program terminates while you have TCPView open, the entry for the program in the list will get a red background. Once it has completely terminated, the entry will be removed from the list. The second column shown in Figure 3-10 is the process identification number (PID). As with the other columns, you can sort by the PID column. You can also get the process properties by right-clicking a process and selecting it. Just knowing the name of the process may be inadequate. For example, the first entry in this list is `dasHost.exe`, which may not be immediately familiar to you and appear suspicious as a result. Viewing the properties shows that this is the Device Association Framework Provider Host from Microsoft. The Device Association Framework Provider Host maintains pairing relationships with devices like Bluetooth headphones.

In addition to TCPView, Sysinternals provides a console view program. Using the TCPView Console program (TCPVCon), you can get a listing of the programs that are communicating on the network using a text-based or console-based view. Where you can get screen captures of the information in the graphical interface, as in Figure 3-10, you would be able to get a text file from TCPView Console that you could then put into a spreadsheet or word processing document in order to present in different ways.

Sysinternals also comes with PsFile, which can list files that someone else on the network may have open on your system. As an example, see Figure 3-11. In this case, one text file was opened but

Process	PID	Protocol	Local Address	Local Port	Remote Address	Remote Port	State	Sent Packets
dasHost.exe	1160	UDP	RICMESSIERCDDC	ws-discovery	*	*		
dasHost.exe	1160	UDP	RICMESSIERCDDC	ws-discovery	*	*		
dasHost.exe	1160	UDP	RICMESSIERCDDC	63807	*	*		
dasHost.exe	1160	UDPV6	ricmessiercdcc	3702	*	*		
dasHost.exe	1160	UDPV6	ricmessiercdcc	3702	*	*		
dasHost.exe	1160	UDPV6	ricmessiercdcc	63808	*	*		
explorer.exe	3120	TCP	ricmessiercdcc.loc...	2189	bn3sch02001052...	https	ESTABLISHED	
lsass.exe	568	TCP	RICMESSIERCDDC	1542	RICMESSIERCDDC	0	LISTENING	
lsass.exe	568	TCPV6	ricmessiercdcc	1542	ricmessiercdcc	0	LISTENING	
services.exe	560	TCP	RICMESSIERCDDC	1540	RICMESSIERCDDC	0	LISTENING	
services.exe	560	TCPV6	ricmessiercdcc	1540	ricmessiercdcc	0	LISTENING	
spoolsv.exe	1336	TCP	RICMESSIERCDDC	1539	RICMESSIERCDDC	0	LISTENING	
spoolsv.exe	1336	TCPV6	ricmessiercdcc	1539	ricmessiercdcc	0	LISTENING	
svchost.exe	684	TCP	RICMESSIERCDDC	epmap	RICMESSIERCDDC	0	LISTENING	
svchost.exe	900	TCP	RICMESSIERCDDC	1537	RICMESSIERCDDC	0	LISTENING	
svchost.exe	844	TCP	RICMESSIERCDDC	1538	RICMESSIERCDDC	0	LISTENING	
svchost.exe	1016	UDP	RICMESSIERCDDC	ssdp	RICMESSIERCDDC	0	LISTENING	
svchost.exe	1016	UDP	ricmessiercdcc.loc...	ssdp	ricmessiercdcc	0	LISTENING	
svchost.exe	844	UDP	RICMESSIERCDDC	teredo	RICMESSIERCDDC	0	LISTENING	
svchost.exe	1124	UDP	RICMESSIERCDDC	5353	RICMESSIERCDDC	0	LISTENING	
svchost.exe	1124	UDP	RICMESSIERCDDC	llntr	RICMESSIERCDDC	0	LISTENING	
svchost.exe	844	UDP	ricmessiercdcc.loc...	53240	ricmessiercdcc	0	LISTENING	
svchost.exe	1016	UDP	ricmessiercdcc.loc...	62252	ricmessiercdcc	0	LISTENING	
svchost.exe	1016	UDP	RICMESSIERCDDC	62253	RICMESSIERCDDC	0	LISTENING	
svchost.exe	684	TCPV6	ricmessiercdcc	epmap	ricmessiercdcc	0	LISTENING	
svchost.exe	900	TCPV6	ricmessiercdcc	1537	ricmessiercdcc	0	LISTENING	
svchost.exe	844	TCPV6	ricmessiercdcc	1538	ricmessiercdcc	0	LISTENING	
svchost.exe	900	UDPV6	[fe80:0:0:341a:a...	546	[fe80:0:0:3567:8...	546	ESTABLISHED	
svchost.exe	900	UDPV6	[fe80:0:0:3567:8...	546	[0:0:0:0:0:0:1]	1900	ESTABLISHED	
svchost.exe	1016	UDPV6	[0:0:0:0:0:0:1]	1900	[fe80:0:0:3567:8...	5353	ESTABLISHED	
svchost.exe	1016	UDPV6	[fe80:0:0:3567:8...	5353	[fe80:0:0:3567:8...	62250	ESTABLISHED	
svchost.exe	1124	UDPV6	[fe80:0:0:3567:8...	62250	[0:0:0:0:0:0:1]	62251	ESTABLISHED	
svchost.exe	1016	UDPV6	[0:0:0:0:0:0:1]	62251	RICMESSIERCDDC	0	LISTENING	
System	4	TCP	ricmessiercdcc.loc...	netbios-ssn	172.30.42.23	microsoft-ds	ESTABLISHED	
System	4	TCP	RICMESSIERCDDC	2273	RICMESSIERCDDC	0	LISTENING	
System	4	TCP	RICMESSIERCDDC	microsoft-ds	RICMESSIERCDDC	0	LISTENING	
System	4	UDP	ricmessiercdcc.loc...	netbios-ns	*	*		
System	4	UDP	ricmessiercdcc.loc...	netbios-dgm	*	*		

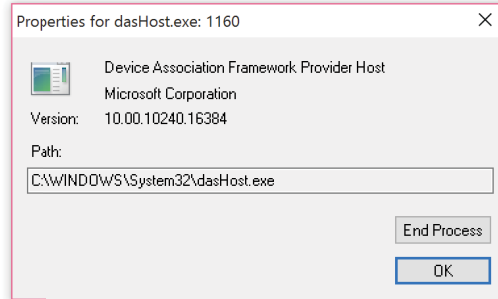


Figure 3-10: TCPView.

```
C:\Users\kilroy\Desktop\Sysinternals>psfile

PsFile v1.03 - Lists files and directories opened remotely
Copyright (C) 2001-2016 Mark Russinovich
Sysinternals

Files opened remotely on RICMESSIERCDDC:

[40] C:\Users\
    User: kilroy
    Locks: 0
    Access: Read
[122] C:\Users\kilroy\Documents
    User: kilroy
    Locks: 0
    Access: Read
[132] C:\Users\kilroy
    User: kilroy
    Locks: 0
    Access: Read
[133] C:\Users\
    User: kilroy
    Locks: 0
    Access: Read
```

Figure 3-11: Looking at remotely opened files with PsFile.
<https://t.me/learningnets>

in order to get to that file, directories also had to be opened. You can see the list using PsFile. All of these entries needed to be opened to get to that one text file from a remote system. PsFile did not need to have any parameters. If you just run PsFile, you will get the list of all the files that are open on this system that are being accessed from a remote system.

The Sysinternals suite comes with the Process Explorer, a utility that provides complete access to the details of each process including allocated memory, the process identification number, the process owner, where it was run from, and any active network communications. In Figure 3-12, the background list displays the properties of one of the processes. This particular process has a number of UDP listeners. Based on what shows in this list, no one is communicating with this process. It is just waiting for a message to come in. While you have to look at each process individually to get this information, the other utilities in this suite should be able to provide the process identification number so that you can get more process details from Process Explorer. This includes all of the external libraries that the program is using, shown in the lower pane. It can also show handles the program has open, which would indicate either files or network streams that the program is responsible for.

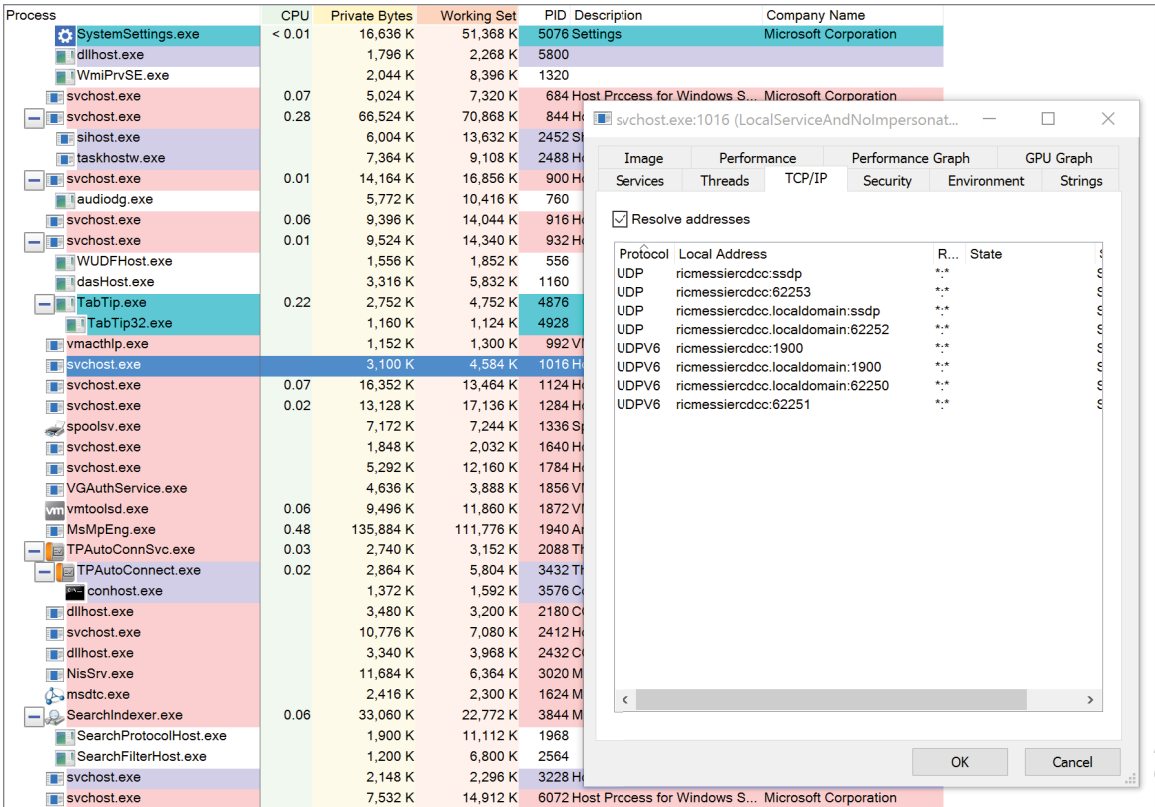


Figure 3-12: Network listeners with Process Explorer.

With nearly 100 unique utilities, the Sysinternals suite has far more functionality than has been covered here and it's worth having the full suite in your bag of tools. The programs we have talked about here are related to getting information about network activity. Other network utilities such as PsPing are more for diagnostics and troubleshooting and less to gather information about what is happening from a network communication perspective. These are all, of course, limited to being used on Windows systems because they were written using the very detailed and rich Windows application programming interfaces. The following sections discuss programs that can be used on the Linux side.

ntop

The top utility is used to show which processes are taking the most resources on a system. The resources measured by top are CPU and memory. This doesn't help when we are looking for network information. Fortunately, there is an analog to top on the network side and it's called ntop. The ntop utility provides a list of the top talkers on the network along with additional statistics. For example, Figure 3-13 shows the IP statistics gathered by running ntop for a short period of time. To get this information, ntop runs in the background as a service, collecting network data. To look at what ntop shows you, use a web browser to connect to <http://localhost:3000>. The ntop application listens there for a browser to connect and launches a web application as the interface to the user.

Network Traffic [IP]: All L3 Hosts - Data Sent+Received							
Host	Location	Data 		Mail_POP	Mail_SMT	MDNS	NTP
172.16.144.141 		1019.3 KBytes	50.0 %	994.9 KBytes	21.8 KBytes	87	2.4 KBytes
 serv...r50.r.cloudfront.net  		378.0 KBytes	18.6 %	378.0 KBytes	0	0	0
 clou...roxy10009.sucuri.net  		267.2 KBytes	13.1 %	267.2 KBytes	0	0	0
 serv...r50.r.cloudfront.net  		125.8 KBytes	6.2 %	125.8 KBytes	0	0	0
 maps.google.com  		119.8 KBytes	5.9 %	119.8 KBytes	0	0	0
 lga2...41-in-f232.1e100.net  		23.8 KBytes	1.2 %	23.8 KBytes	0	0	0
172.16.144.2 		21.9 KBytes	1.1 %	0	21.8 KBytes	0	0
 self-repair.mozilla.org  		13.5 KBytes	0.7 %	13.5 KBytes	0	0	0
 serv...fk1.r.cloudfront.net  		9.5 KBytes	0.5 %	9.5 KBytes	0	0	0
 lga2...41-in-f228.1e100.net  		9.0 KBytes	0.4 %	9.0 KBytes	0	0	0
 lga2...s41-in-f14.1e100.net  		7.7 KBytes	0.4 %	7.7 KBytes	0	0	0
 ocsp.digicert.com  		7.1 KBytes	0.3 %	7.1 KBytes	0	0	0
 ec2-...ompute.amazonaws.com  		5.3 KBytes	0.3 %	5.3 KBytes	0	0	0
 ec2-...ompute.amazonaws.com  		5.1 KBytes	0.2 %	5.1 KBytes	0	0	0

Figure 3-13: IP statistics using ntop.

Figure 3-13 shows the list of hosts that have been seen. If ntop can't locate a hostname by performing a lookup, it displays just the IP address as in the case of the first entry. That is the system that ntop is running on but because it's a private IP address with no local DNS server or other way to provide a hostname, it has no DNS entry. ntop can also provide you a summary of the top talkers that were seen on the network interface on this system. In most cases, that will be the traffic that was sent from and destined to this system. In some cases, you can set up a system to run ntop and funnel more traffic to it to gather this information. In Figure 3-14, you can see some of the top talkers from the last hour prior to this capture. What you get here is the minute-by-minute listing of all of the systems that were communicating. If traffic was seen during that minute, it will show up in the block corresponding to that minute.

Top Talkers: Last Hour					
Time Period		Top Senders		Top Receivers	
1.	Mon Aug 15 19:25:00 2016 Mon Aug 15 19:25:59 2016	172.16.144.141 	1.4 Kbit/s	172.16.144.141 	39.4 Kbit/s
		172.16.144.2 	436.7 bit/s	172.16.144.2 	250.8 bit/s
		lga1...s44-in-f14.1e100.net 	160.9 bit/s	lga15s49-in-f4.1e100.net 	14.8 bit/s
		lga15s49-in-f4.1e100.net 	40.1 bit/s	lga1...s44-in-f14.1e100.net 	3.8 bit/s
		qg-in-f95.1e100.net 	6.4 bit/s	qg-in-f95.1e100.net 	3.5 bit/s
2.	Mon Aug 15 19:24:00 2016 Mon Aug 15 19:24:59 2016	172.16.144.141 	1.5 Kbit/s	172.16.144.141 	41.3 Kbit/s
		172.16.144.2 	457.1 bit/s	172.16.144.2 	262.2 bit/s
		lga2...41-in-f228.1e100.net 	251.0 bit/s	lga2...41-in-f228.1e100.net 	82.7 bit/s
		lga1...s49-in-f14.1e100.net 	171.8 bit/s	lga15s49-in-f4.1e100.net 	15.6 bit/s
		lga1...s44-in-f14.1e100.net 	166.6 bit/s	lga1...s49-in-f14.1e100.net 	4.5 bit/s
3.	Mon Aug 15 19:23:00 2016 Mon Aug 15 19:23:59 2016	172.16.144.141 	1.5 Kbit/s	172.16.144.141 	43.2 Kbit/s
		172.16.144.2 	458.0 bit/s	172.16.144.2 	263.1 bit/s
		lga2...41-in-f228.1e100.net 	260.0 bit/s	lga2...41-in-f228.1e100.net 	87.0 bit/s
		lga1...s49-in-f14.1e100.net 	178.8 bit/s	lga1...s49-in-f14.1e100.net 	4.8 bit/s
		atl14s38-in-f3.1e100.net 	8.0 bit/s	atl14s38-in-f3.1e100.net 	4.1 bit/s

Figure 3-14: Top talkers using ntop.

The challenge with ntop is that it needs to be installed on a system and it needs to be listening. This can be highly intrusive. A service also needs to be running in order to gather information. If the service isn't running and collecting data, you won't see anything. The web interface won't exist, for a start, because there will be no application listening on the port for the interface. Beyond that, though, if you start ntop without the service just to get to the web interface, no data will have been collected to look at. So using ntop takes some planning. Even if ntop was installed but not noticed by an attacker, the activities of the service—running, collecting data, and listening on a network interface—are also easy to notice.

One advantage of ntop is that, unlike some utilities, you can install and run it on multiple operating systems. There are some utilities that are only available on specific operating systems. That's especially true with utilities that come with the operating environment. Windows, as an example, has a lot of programs and utilities built in that can be used for monitoring and diagnostics.

Task Manager/Resource Monitor

Windows Task Manager provides a view into network activity, though you may be more familiar with it as a way to look at process lists. Over the last several releases of Windows, the Task Manager has become a very powerful monitoring utility. When you start it up, you will get a list of applications that you are running. In Windows 10, if you click the “More details” link, you will get a set of tabs across the top allowing you to view more details about different resources on the system including Processes and Performance. The Performance tab displays dynamic graphs that show how the CPU, Memory, Network, and Disk are performing. You can see the network activity in Figure 3-15. Though this doesn't have a lot of detail, it does show a running graph of how much network traffic is coming into and going out of your system.

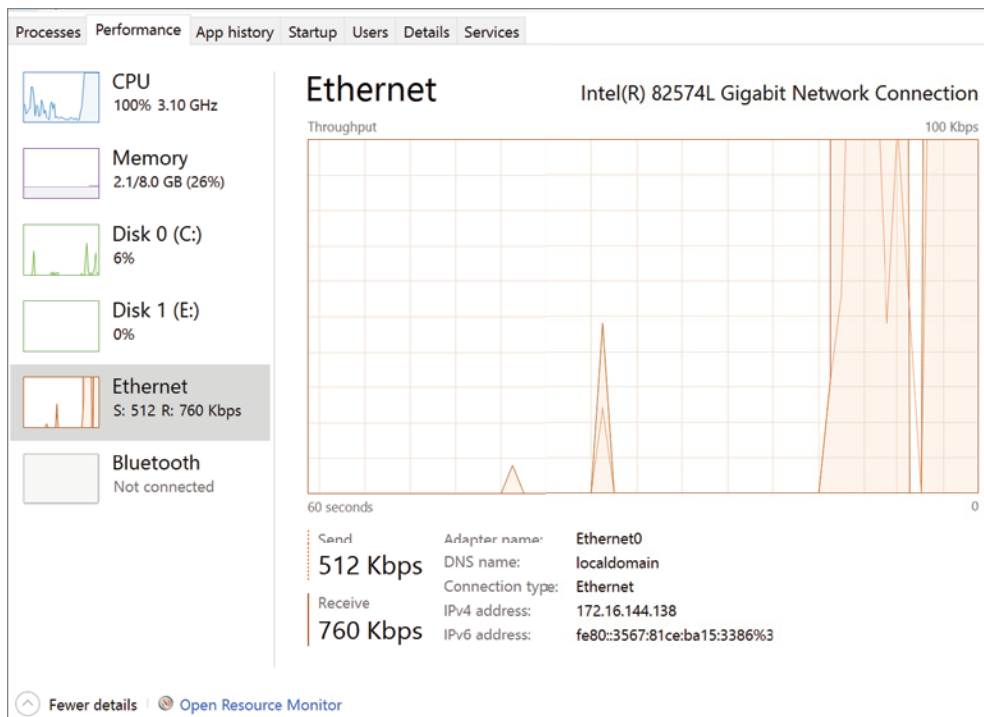


Figure 3-15: Task Manager Ethernet display.

This view does not show what is happening with applications or even the ports that are open and being used. To get access to that information, you can open the Resource Monitor from the Task Manager. As its name suggests, the Resource Monitor allows you to get more specific information about various resources. Again, just like in the Task Manager, you can look at Memory, Network, Disk, and CPU. The Network tab provides a quick view of a lot of the information we have been talking about. You can see a portion of this Network tab in Figure 3-16. This provides a place to look at all of the network communications with details and statistics. If you look at the TCP Connections section, you can see all of the processes, the process ID, the local address, the remote address, the packet count, and the latency value.

NOTE Latency is a measure of how long it takes to get from one end of a communication stream to the other.

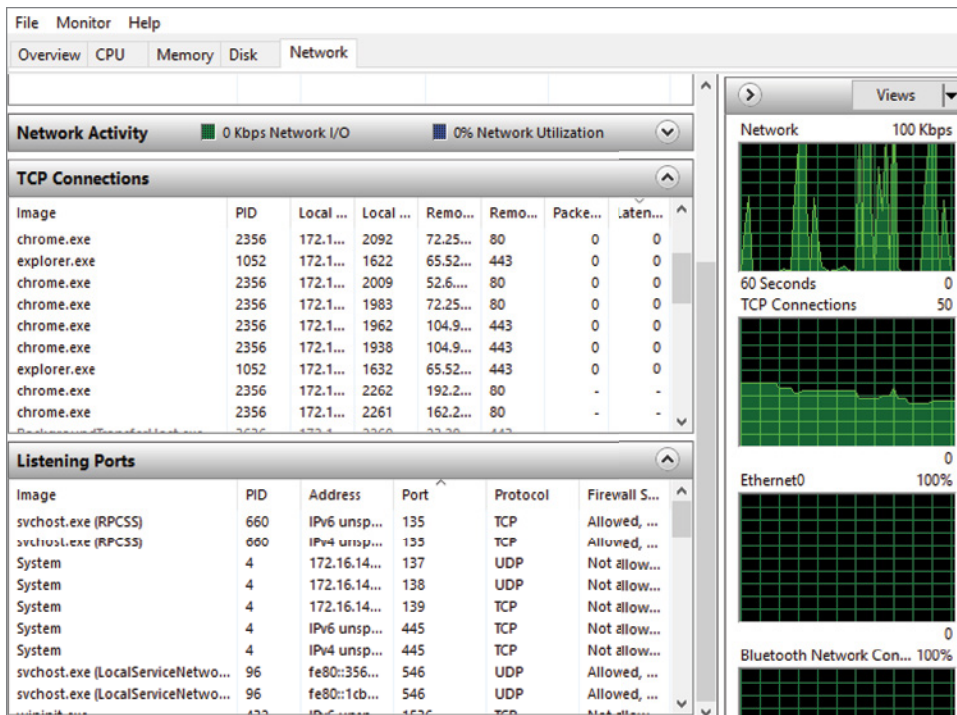


Figure 3-16: Resource Monitor.

One advantage of looking at the Resource Monitor is that you can see the listening ports quickly in a way that can be sorted. On top of seeing the applications that are listening and the information about the ports they are listening on, you can also see the firewall state. Windows has a built-in firewall and, though we haven't talked about it to this point, just because an application is listening

doesn't mean that anyone can get to the application. The Windows firewall can block traffic to that application, even based on the application, meaning all you need to do is tell the Windows Firewall that the application in question should not be using the network and it won't matter what ports are in use. Using the Resource Monitor, you can collect the listener ports and see whether anyone can reach the application as long as the Windows firewall is running. In other words, you can gather two pieces of information at the same time—whether the application has ports that are listening for traffic and whether the Windows Firewall is going to allow any remote system to communicate over those ports. The Windows Firewall will not prevent the application from listening on those ports, but it can prevent inbound communications from getting to the application.

ARP

The address resolution protocol (ARP) allows systems to perform lookups from IP addresses to MAC addresses. The MAC address is required for two systems to communicate on a local network. It is also required for a system to get something off the local network because the MAC address of the router is necessary as the destination address in the Ethernet header. Each system maintains a cache of the address resolutions to save time when it wants to send a message out on the local network.

Operating systems that have a TCP/IP protocol stack will typically have the `arp` utility installed, though `arp` is only useful if you are communicating using IPv4. The `arp` utility allows you to manage the ARP cache on the local system. Figure 3-17 shows the entire ARP cache on my system. By default, `arp` will attempt to perform reverse DNS lookups on the IP address, though since none of these addresses can be resolved, you will see a `?` in place of the hostname.

NOTE In IPv6, the Neighbor Discovery Protocol replaces the `arp` command. As we will discover in subsequent chapters, `arp` has a few security issues that IPv6 is attempting to remove from networks, including the ability to pretend to be someone else on the network in order to receive their messages.

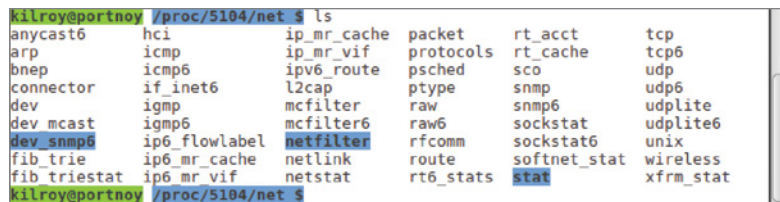
```
kilroy@oliver:~$ arp -a
? (169.254.136.145) at (incomplete) on en0 [ethernet]
? (169.254.196.202) at (incomplete) on en0 [ethernet]
? (172.16.144.138) at (incomplete) on vmnet8 ifscope [ethernet]
? (172.16.144.139) at (incomplete) on vmnet8 ifscope [ethernet]
? (172.16.144.255) at (incomplete) on vmnet8 ifscope [ethernet]
? (172.16.222.255) at (incomplete) on vmnet1 ifscope [ethernet]
? (172.30.42.1) at 44:e1:37:f4:f2:34 on en0 ifscope [ethernet]
? (172.30.42.3) at (incomplete) on en0 ifscope [ethernet]
? (172.30.42.5) at (incomplete) on en0 ifscope [ethernet]
? (172.30.42.7) at a4:5d:36:56:c4e on en0 ifscope [ethernet]
? (172.30.42.9) at (incomplete) on en0 ifscope [ethernet]
? (172.30.42.11) at (incomplete) on en0 ifscope [ethernet]
? (172.30.42.16) at f4:5c:89:b7:2c:89 on en0 ifscope permanent [ethernet]
? (172.30.42.19) at 38:1:95:6a:68:1f on en0 ifscope [ethernet]
? (172.30.42.23) at ac:87:a3:36:d6:aa on en0 ifscope [ethernet]
? (172.30.42.24) at 58:82:a8:44:a0:5f on en0 ifscope [ethernet]
? (172.30.42.25) at (incomplete) on en0 ifscope [ethernet]
? (172.30.42.255) at (incomplete) on en0 ifscope [ethernet]
? (224.0.0.251) at 1:0:5e:0:0:fb on en0 ifscope permanent [ethernet]
```

Figure 3-17: ARP cache.

Using the arp utility, you can display specific entries based on hostname instead of displaying all of the cache. You can also delete specific entries. If you run into a case where your local ARP cache is wrong, you can delete the entry and force the system to ask again. In most cases, you will get the MAC address, though if the system can't do a resolution, you will see `(incomplete)`. This is for addresses that the system knows about but the host at that address can't be reached. As an example, if you were to try to ping a system on your local network that isn't responding, that IP address will show up in the ARP cache because it was an IP address you tried to get to. Because it wasn't responsive, perhaps because it was down, there is no MAC address to provide. The entry is there but it's incomplete so the arp utility indicates that much.

/proc Filesystem

Linux systems have a pseudo-filesystem that is an interface to the kernel memory. The data structures associated with processes are available through the `/proc` filesystem. Every proc entry includes a large collection about the process and all of the resources that are available to it. In each case, you get access to the resources by going to the directory named for the process ID number. Underneath that, you will find a number of other directories. One of those directories is the `net` directory, the contents of which you can see in Figure 3-18. It contains all of the networking resources and statistics available to the process.



```
kilroy@portnoy /proc/5104/net $ ls
anycast6      hci          ip_mr_cache  packet      rt_acct      tcp
arp           icmp         ip_mr_vif    protocols   rt_cache     tcp6
bnep          icmp6        ipv6_route   psched      sco          udp
connector     if_inet6     l2cap        ptype       snmp         udp6
dev           igmp         mcfilter     raw         snmp6        udplite
dev_mcast     igmp6        mcfilter6    raw6        sockstat     udplite6
dev_snmp6     ip6_flowlabel netfilter     rfcomm      sockstat6    unix
fib_trie      ip6_mr_cache netlink       route       softnet_stat wireless
fib_triestat  ip6_mr_vif  netstat      rt6_stats   stat         xfrm_stat
kilroy@portnoy /proc/5104/net $
```

Figure 3-18: `/proc` entry for Nginx web server.

Only processes that use network resources will have much useful information in this directory. Each file will have data relevant to the name of the file. As an example, the contents of the `tcp` file are the details about the sockets in use with TCP. You can see the contents of the `tcp` file for the Nginx process in Listing 3-4. In the first line, you can see the socket listening on port 80. This may not be apparent immediately because the value representing that is 50. The 50 you see is in hexadecimal and converting that to decimal is as simple as multiplying 5 by 16, because the 5 is in the 16's place. When you multiply those two values, you get 80.

Listing 3-4: Contents of `tcp` File

```
sl local_address rem_address  st tx_queue rx_queue tr tm->when
retrnsmt uid timeout inode
```

```

0: 00000000:0050 00000000:0000 0A 00000000:00000000 00:00000000
00000000 0 0 36271 1 0000000000000000 100 0 0 10 0
1: 0101007F:0035 00000000:0000 0A 00000000:00000000 00:00000000
00000000 0 0 22036 1 0000000000000000 100 0 0 10 0
2: 00000000:0BB8 00000000:0000 0A 00000000:00000000 00:00000000
00000000 0 0 26025 1 0000000000000000 100 0 0 10 0

```

Summary

When you are looking at network information, you can get a lot of data from asking the host directly because every operating system keeps track of the network communications passing across its interfaces. In most cases, you can get access to that information by using system utilities but you may not want to use them because they may have been compromised. In that case, you can use third-party utilities, especially those that can be run without being installed. One source of those utilities on the Windows side is Microsoft, which provides the Sysinternals suite for free. The Sysinternals suite includes a lot of useful diagnostic utilities and programs. Some of the Sysinternals programs can be used to extract a lot of details related to the connections between the local system and remote systems.

Operating systems also include a number of utilities that are necessary for looking at network connections. The `netstat` utility, as an example, provides network statistics about listening ports and connections between two systems. The `arp` utility can be used to highlight systems on the local network that the system under investigation has received some communication from. ARP is the protocol used to identify and communicate with systems on the local network, so the `arp` utility provides information about the communication of that protocol, specifically the addresses that your system knows about.

The Windows Sysinternals team and your friendly, local operating system are not the only places to be finding tools that can be used to extract information about what the operating system knows about network connections and communications. In fact, as noted previously, your operating system utilities may have been corrupted by a rootkit that is altering what you see. If you can find tools that you can load up to a portable drive, like those provided by Sysinternals or even a vendor like Nirsoft, you have a way to corroborate what you are getting from your operating system.

4

Packet Capture and Analysis

In this chapter you will learn about:

- What a packet is
- How to capture packets
- Utilities used to capture packets like tcpdump and Wireshark
- How to use Wireshark to analyze captures
- Challenges associated with packet captures on networks

He has begun to move into the servers and away from the desktops, feeling like he has enough desktops from this particular network under his belt. The challenge with the servers, though, is that they are likely to be better monitored. This may mean that an administrator has a packet capture running. Even if he is encrypting his communication to the server, he realizes that anyone capturing packets would be able to see at least where he is coming from and that backdoor port is going to look a little suspicious over time. Fortunately, it isn't that hard to determine whether an administrator is capturing packets. Checking out the list of running programs will work, searching for well-known programs that will do that. On top of that, it's possible to see whether the network interface has been placed into the mode necessary to capture traffic.

Unfortunately, he is aware that just because no one is watching him from this system doesn't mean that no one is watching him from somewhere else. It could be that a network administrator is watching using a spanning port or a network tap. Nothing much to do about it, though. Encrypt and move fast. Hope any users or system administrators don't see it and shut everything down before he can get the good stuff and maybe even find another system to move off to.

One of the most important skills you can acquire when it comes to any sort of network analysis is capturing packets and performing analysis. When it comes to doing network investigations, the wire is your friend. Packet capture captures what happens on the wire, and what happens on the wire is true and accurate because there is nothing to get in the way. With applications, there are a lot of ways to get it wrong. Even the operating system can be tricked into providing you incorrect information if malware has infected the system. However, once you are down to reading electrical signals or light pulses off a cable, you have exactly what is happening. Of course, it is possible to perform this capture

on a system that may, in fact, be infected but so far there is no malware that has gotten so low that it can get between what is being transmitted out to the network and the capture of that information.

For the most part, this is information you can absolutely rely on. Nothing is hidden and there isn't a lot of poking and prodding needed to find additional information. Applications that are communicating across the network have to use the same protocols on both ends and they are generally well-known protocols. If they don't each use the same protocols, it would be like someone talking Japanese to someone else who is speaking Portuguese. Neither is going to be able to understand the other very well. From a networking perspective, you may as well simply not send anything out or just drop it on the floor, as we say in the networking world.

Fortunately for us, some excellent resources are available that can help us with both capturing and analyzing the network data. One that we will spend a lot of time with in this chapter, Wireshark, has made life much easier for anyone who wants to do anything with networks. Wireshark provides a lot of capabilities and does an incredible amount of analysis and decoding. This is especially amazing considering that it costs nothing and goes well beyond the capabilities of the early protocol analyzers that cost thousands of dollars.

While Wireshark is a graphical program, which makes visualization easier and more convenient, you won't always have the luxury of having a graphical interface. As a result, you will need to rely on console programs like `tcpdump` and `tshark` to do the collection of packets for you. If you aren't seeing what you need in the console output, you can then import a saved packet capture from them to perform analysis using Wireshark.

When it comes to capturing packets, you need to consider where you are actually going to capture the packets you need. You can do it on an endpoint device, like a desktop computer, but that requires installing software. You may not want to install this software on the endpoint you are trying to pay attention to, for a variety of reasons. Fortunately, you have other ways to capture those same packets on a separate device, while still getting all of the data that would be available on the target device. This is one of the great things about networks in general and network forensics specifically—you have a number of ways and places to get the same information and it will always be the same.

While Wireshark has a lot of capability when it comes to capturing and analyzing captures, when it comes to forensics, there are other tools that can do some of the heavy lifting for you. One of those is a program called NetworkMiner. Using NetworkMiner, you can capture traffic off the network, just as you can with Wireshark and `tcpdump`. Rather than showing you just the packets for you to analyze, NetworkMiner will pull useful files and other interesting evidence out of the capture. A tool like this makes life quite a bit easier for forensics practitioners.

However, no matter what tool you are using, or where in the network you are using it, we have to capture some packets, so let's start there.

Capturing Packets

While we call it capturing packets—and in the end, it's packets that we are looking at—in reality what we are capturing is *frames*. Remember that when we are talking about data structures at the

wire level, we are talking about frames. When you are capturing on a local area network (LAN), you are going to be looking at *Ethernet frames*. Ethernet, by default, has a maximum transmission unit (MTU) of 1500 bytes, including the headers. Any packet that is larger than that will be fragmented into smaller frames. Once you have the data captured, you can put it all back together, of course. However, each unit of data you are capturing is a frame.

Packet capture programs insert themselves into the network stack, meaning they are in the middle of the operating system, which is responsible for getting frames out onto the network. Prior to the frames being sent to the network interface to be converted to either an electrical signal or a radio signal, the packet capture program will grab copies of the frames and store them. This may just be long enough to display the header information, as may be the case with a command-line capture program, or they may write the frames out to disk in a specially formatted file that can be opened later on to display the frames for analysis. Similarly, on the way in, before the frames are handed up to the higher layers of the network stack, the packet capture program will take copies.

Because these programs are engaged in some of the input/output functions of the operating system, they require administrative privileges. Not every user is going to be capable of capturing packets. A Windows system may require that you elevate the privileges of the packet capture program or it may interface with a service that is operating with elevated privileges. Other operating systems like macOS and Linux will also require elevated privileges to capture the packets.

Another reason why the elevated privileges are necessary is because the network interface needs to be configured to operate in a special mode. By default, network interfaces will only respond to messages that are addressed directly to them or sent to a broadcast address. Because the network interface carries its own MAC address, it knows whether a frame coming in matches either its MAC address or the ff:ff:ff:ff:ff:ff address of a broadcast message. Any other messages are going to be dropped by default and not passed up to the operating system. To capture packets, the network interface needs to be told to capture everything that is seen and pass it up to the operating system. When a network interface is capturing all messages, it is said to be in *promiscuous mode*.

NOTE In a network that uses switches, only traffic destined for that system comes across the network interface, which is not how networks used to operate many years ago. As a result, a network interface that is set into promiscuous mode on a switched network won't see anything that it wouldn't normally see anyway. There are ways to get around this and they will be covered later.

There was a time when only specially constructed network interfaces were capable of this particular mode, which is one reason why network analyzers could be so expensive. These days, however, nearly all network interfaces can be placed into promiscuous mode. This means that any system can become a network analyzer using free software, which really changes the game when it comes to packet analysis.

There is another special mode that is necessary on wireless networks, referred to as *monitor mode*. With wireless networks being more commonplace, you can capture the wireless frames easily but what you won't see, unless you have a wireless interface that supports monitor mode, is the radio

headers. This is the communication between the client device and wireless access point. In most cases, this information isn't necessary but sometimes it's useful to see. Not all packet capture programs are capable of turning on monitor mode and not all wireless interfaces are capable of supporting it. If you need to capture the radio frames from a set of wireless communication, because those will show you authentication to the wireless network and other useful information, you need to make sure you have an interface that can be set into monitor mode.

A number of programs are available to do packet capture; this chapter covers the most common ones. Although others may be available, they will often behave very similarly, including the command-line parameters needed to make the program operate. As a result, we aren't going to be exhaustive in talking about different packet capture programs, but you will be able to get a good handle on the process.

Tcpdump/Tshark

Tcpdump is a program that has been available on Unix operating systems for decades. Various versions of the program existed in various states on different Unix-like operating systems for over a decade before they were finally collected into a single project in 1999. In the meantime, other packet capture programs were available, like snoop on the Sun Solaris operating system. While there may have been implementations of tcpdump available for Linux prior to 1999, tcpdump has been available as a package on most if not all Linux distributions. There has also been a port available for Windows called windump that runs on the same underlying packet capture library that tcpdump uses.

By default, tcpdump will print summary header details of each frame that it captures. This is all you get unless you tell it you are looking for something else. By default, without any additional options, Listing 4-1 shows what a packet capture looks like using tcpdump.

Listing 4-1: Packet Capture Using tcpdump

```
kilroy@oliver:~$ sudo tcpdump
tcpdump: data link type PKTAP
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on pktap, link-type PKTAP (Apple DLT_PKTAP), capture size
262144 bytes
20:42:04.498961 IP 172.30.42.19.vcom-tunnel >
st-routers.mcast.net.vcom-tunnel: UDP, length 182
20:42:04.503756 IP6 2601:19b:c601:ec00:703e:e7ce:d930:beb9.55010 >
cdns01.comcast.net.domain: 39462+ PTR? 19.42.30.172.in-addr.arpa. (43)
20:42:04.522661 IP6 cdns01.comcast.net.domain >
2601:19b:c601:ec00:703e:e7ce:d930:beb9.55010: 39462 NXDomain 0/0/0 (43)
20:42:04.524207 IP6 2601:19b:c601:ec00:703e:e7ce:d930:beb9.62817 >
cdns01.comcast.net.domain: 6428+ PTR? 7.0.0.224.in-addr.arpa. (40)
20:42:04.585214 IP6 cdns01.comcast.net.domain >
2601:19b:c601:ec00:703e:e7ce:d930:beb9.62817:
```

```

6428 1/0/0 PTR st-routers.mcast.net. (74)
20:42:04.586750 IP6 2601:19b:c601:ec00:703e:e7ce:d930:beb9.61338 >
cdns01.comcast.net.domain: 1541+ PTR?
9.b.e.b.0.3.9.d.e.c.7.e.e.3.0.7.0.0.c.e.1.0.6.c.b.9.1.0.1.0.6.2.ip6.
arpa. (90)
20:42:04.619422 IP6 cdns01.comcast.net.domain >
2601:19b:c601:ec00:703e:e7ce:d930:beb9.61338: 1541 NXDomain 0/1/0 (171)
20:42:04.620773 IP6 2601:19b:c601:ec00:703e:e7ce:d930:beb9.50271 >
cdns01.comcast.net.domain: 52826+ PTR?
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.d.e.e.f.8.5.5.0.1.0.0.2.ip6.
arpa. (90)
20:42:04.642004 IP6 cdns01.comcast.net.domain >
2601:19b:c601:ec00:703e:e7ce:d930:beb9.50271: 52826 1/0/0 PTR
cdns01.comcast.net. (122)

```

The summary information shows the time that the packet was captured, the protocol in use, the source and destination address, and then some additional information, based on the type of packet. As an example, the first packet shows that it's a UDP packet and the length is 182 bytes. The fourth packet down indicates that there was a DNS request for a PTR record. If you look at the end of each of the addresses that are shown, you will notice that there is a port number. However, it may not look like a port number. In the very first packet, the source address is 172.30.42.19.vcom-tunnel. The first part of that is the IP address but the second, vcom-tunnel, is the name for port number 8001. Some port numbers are considered well-known and registered. Tcpcat is capable of looking up the name associated with the port and printing that out. It does this by default, though it doesn't have to. Similarly, you will see that it prints out the hostname for each address that has a hostname. It has to do a lookup to print that out, however.

You can also ask tcpcat to provide more detail in the summary output. All you need to do is add a -v to the command line. This tells tcpcat that you are looking for additional verbosity in the output. Listing 4-2 shows a sample of tcpcat output with the additional verbosity. However, this is still just a summary of the header information and doesn't include any of the contents of the packet. There is no data shown here.

Listing 4-2: tcpcat Capture with Verbose Setting

```

kilroy@oliver:~$ sudo tcpcat -v
tcpcat: data link type PKTAP
tcpcat: listening on pktap, link-type PKTAP (Apple DLT_PKTAP), capture
size 262144 bytes
20:51:30.683950 IP6 (flowlabel 0x32bf0, hlim 64, next-header TCP (6)
payload length: 20) 2601:19b:c601:ec00:703e:e7ce:d930:beb9.62799 >
2606:2800:11f:17a5:191a:18d5:537:22f9.https: Flags [F.], cksum 0xb169
(correct), seq 1776269065, ack 1937819765, win 8192, length 0
20:51:30.687980 IP6 (flowlabel 0x249d4, hlim 255, next-header UDP (17)
payload length: 98) 2601:19b:c601:ec00:703e:e7ce:d930:beb9.58865 >
cdns01.comcast.net.domain: [udp sum ok] 55887+ PTR?

```

```

9.f.2.2.7.3.5.0.5.d.8.1.a.1.9.1.5.a.7.1.f.1.1.0.0.0.8.2.6.0.6.2.ip6.
arpa. (90)
20:51:30.753189 IP6 (hlim 57, next-header UDP (17) payload length: 98)
cdns01.comcast.net.domain > 2601:19b:c601:ec00:703e:e7ce:
d930:beb9.58865: [udp sum ok] 55887 ServFail 0/0/0 (90)
20:51:30.753466 IP6 (flowlabel 0x249d4, hlim 255, next-header UDP (17)
payload length: 98) 2601:19b:c601:ec00:703e:e7ce:d930:beb9.58865 >
cdns02.comcast.net.domain: [udp sum ok] 55887+ PTR?
9.f.2.2.7.3.5.0.5.d.8.1.a.1.9.1.5.a.7.1.f.1.1.0.0.0.8.2.6.0.6.2.ip6.
arpa. (90)
20:51:30.787209 IP6 (hlim 56, next-header UDP (17) payload length: 98)
cdns02.comcast.net.domain > 2601:19b:c601:ec00:703e:e7ce:d930:
beb9.58865: [udp sum ok] 55887 ServFail 0/0/0 (90)
20:51:30.787441 IP (tos 0x0, ttl 255, id 49528, offset 0, flags [none],
proto UDP (17), length 118)
    172.30.42.16.58865 > cdns01.comcast.net.domain: 55887+ PTR?
9.f.2.2.7.3.5.0.5.d.8.1.a.1.9.1.5.a.7.1.f.1.1.0.0.0.8.2.6.0.6.2.ip6.
arpa. (90)
20:51:30.853846 IP (tos 0x40, ttl 58, id 0, offset 0, flags [DF], proto
UDP (17), length 118)
    cdns01.comcast.net.domain > 172.30.42.16.58865: 55887 ServFail 0/0/0
(90)
20:51:30.854057 IP (tos 0x0, ttl 255, id 1754, offset 0, flags [none],
proto UDP (17), length 118)
    172.30.42.16.58865 > cdns02.comcast.net.domain: 55887+ PTR?
9.f.2.2.7.3.5.0.5.d.8.1.a.1.9.1.5.a.7.1.f.1.1.0.0.0.8.2.6.0.6.2.ip6.
arpa. (90)
20:51:30.887574 IP (tos 0x40, ttl 57, id 0, offset 0, flags [DF], proto
UDP (17), length 118)

```

What you get, as you can see in the output, is additional header information. In the first packet shown, you can see the payload length as well as the flags and the checksum. The sequence number and acknowledgment number that are in the TCP headers are shown here as well. What you can also see is that the checksums have been validated and have checked out. With IPv6, there is a next header field to indicate what the next protocol beyond IP is. The first two packets are both IPv6 and they both have different next header fields. One of them indicates that the next header is TCP and the other's next header is UDP.

So far, we are talking about just summary information. We can get some additional information out of tcpdump by adding an additional `v` to the command-line parameters. To get more of a protocol decode, you can use `-vv` with tcpdump. You can see more details in the packet capture in Listing 4-3.

Listing 4-3: Protocol Decode with tcpdump

```

20:20:27.806833 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has
172.30.42.1 tell 172.30.42.19, length 46
20:20:28.011816 IP (tos 0x0, ttl 128, id 57005, offset 0, flags [none],
proto UDP (17), length 328)

```

```

172.30.42.1.bootps > broadcasthost.bootpc: [udp sum ok] BOOTP/DHCP,
Reply, length 300, xid 0xaad02101, Flags [Broadcast] (0x8000)
  Server-IP 172.30.42.1
  Client-Ethernet-Address b8:27:eb:11:c2:5a (oui Unknown)
  Vendor-rfc1048 Extensions
  Magic Cookie 0x63825363
  DHCP-Message Option 53, length 1: NACK
20:20:28.243372 IP (tos 0x0, ttl 64, id 10615, offset 0, flags [DF],
proto TCP (6), length 64)
  172.30.42.16.56969 > 40.84.149.239.https: Flags [S], cksum 0xe9b9
(correct), seq 3595239578, win 65535, options [mss 1460,nop,wscale
5,nop,nop,TS val 662952483 ecr 0,sackOK,eol], length 0
20:20:28.319312 IP (tos 0x20, ttl 112, id 22932, offset 0, flags [DF],
proto TCP (6), length 60)
  40.84.149.239.https > 172.30.42.16.56969: Flags [S.], cksum 0x0ce2
(correct), seq 2509957292, ack 3595239579, win 8192, options [mss
1440,nop,wscale 8,sackOK,TS val 267796383 ecr 662952483], length 0
20:20:28.319361 IP (tos 0x0, ttl 64, id 13051, offset 0, flags [DF],
proto TCP (6), length 52)
  172.30.42.16.56969 > 40.84.149.239.https: Flags [.], cksum 0x4b46
(correct), seq 1, ack 1, win 4105, options
[nop,nop,TS val 662952559 ecr 267796383], length 0
20:20:28.320520 IP (tos 0x0, ttl 64, id 12940, offset 0, flags [DF],
proto TCP (6), length 271)
  172.30.42.16.56969 > 40.84.149.239.https: Flags [P.], cksum 0xbd8d
(correct), seq 1:220, ack 1, win 4105, options [nop,nop,TS val 662952560
ecr 267796383], length 219
20:20:28.327423 IP (tos 0x0, ttl 64, id 34667, offset 0, flags [DF],
proto TCP (6), length 295)
  172.30.42.16.56568 > edge-star-shv-01-lga3.facebook.com.https: Flags
[P.], cksum 0xfb46 (correct), seq 57304:57547, ack 10212, win 4096,
options [nop,nop,TS val 662952567 ecr 1157808256], length 243
20:20:28.327574 IP (tos 0x0, ttl 64, id 13595, offset 0, flags [DF],
proto TCP (6), length 1450)
  172.30.42.16.56568 > edge-star-shv-01-lga3.facebook.com.https: Flags
[.], cksum 0x7bdc (correct), seq 57547:58945, ack 10212, win 4096,
options [nop,nop,TS val 662952567 ecr 1157808256], length 1398

```

Much of this looks like what you have seen before. A couple of lines here, though, are a bit more telling in terms of the additional detail. The very first line is an ARP request. Tcpcap is actually decoding the packet for you. Rather than simply presenting the raw data, it's telling you what the packet is really doing. In an ARP message like the one shown, the raw data would be binary and you would have to do a lookup for what the operation code (opcode) means. This is what tcpcap has done. Tcpcap is telling you that this is an ARP request and the request is for the MAC address of 172.30.42.1. The system that has 172.30.42.1 should reply to 172.30.42.19.

Another packet that stands out is the next one down, which is a dynamic host configuration protocol (DHCP) message. To locate the individual packets in a capture like this, find the timestamp. That will indicate the start of each packet. This decode indicates that this has an option code of 53, which

indicates what type of message this is. In this case, it is a non-acknowledgment (NACK), which is typically used by the server to indicate there was a problem with a request. The message we are looking at has originated from 172.30.42.1, which is the IP address of the DHCP server on the network.

Other packets that have been captured in this sample provide varying degrees of useful information. In most cases, rather than just a summary of the headers, tcpdump is providing a decode of all of the header information. This means that tcpdump is not presenting the information to you as it would be if you were just to look at the data. Instead, it is converting the data to something that is more meaningful to us. The next-to-last packet shows that the Push flag is set, as an example. In the actual packet data, it doesn't say Push. It's a single bit that is set. Tcpdump also provides us with the checksum value and the fact that it is correct, meaning tcpdump calculated a checksum on the packet and it is the same as the one that was transmitted.

Again, header information is great. You will very often need data. This means that you need to grab the entire packet and you will also need to make sense of the data you have. As fond as I am of console-based (text-based) programs, this is a case for a graphical interface because it's easier to visualize and break the packet apart in ways we don't see in the console. However, tcpdump is a great way to safely capture data. It's also a good way to get captures from nearly any system. You can make use of a lightweight sensor or even a less powerful computer to capture packets using tcpdump. Graphical interfaces are far more complex. What we are going to do below is make sure that we are not only capturing all of the data, but we are also writing the results out to a file on the disk. You can see the necessary flags for tcpdump in Listing 4-4.

Listing 4-4: Capturing and Saving Packets

```
kilroy@oliver:~$ sudo tcpdump -s 0 -w capture.pcap
tcpdump: data link type PKTAP
tcpdump: listening on pktap, link-type PKTAP (Apple DLT_PKTAP), capture
size 262144 bytes
```

The `-s` flag tells tcpdump that we are using a snap length of 0, which really tells tcpdump to use the maximum value. By default, tcpdump will only capture 68 bytes, because that will generally ensure that you get the header information necessary for tcpdump to do the decoding that it is capable of. What we need is the entire packet and not just the first 68 bytes. The `-w` flag says to write out to the disk. The value after the flag indicates the filename that will be used to store the packet capture to.

One aspect of tcpdump that may be of some value, and that we will cover in more detail in the “Filtering” section later in this chapter, is filters. You will find that in the filtering section later in this chapter. You can add a filter onto the command line to tell tcpdump which packets you want to capture. As an example, you can specify a host. When you tell tcpdump that you only want to capture based on a host, the only packets that will be captured are ones where the source or destination addresses match the value that was provided to tcpdump with the host parameter. You can also tell

tcpdump which protocols you want to capture. These filters, called Berkeley Packet Filters, will help you limit the capture size so you can focus on the data that you are really interested in looking at.

One advantage of learning how tcpdump works is that other packet capture programs will use the same command-line parameters. All you need to do to get packets captured is to change the name of the program. One such program that works on the command line but makes use of the same command-line parameters as tcpdump is tshark, which is a command-line program that comes with Wireshark.

Wireshark

Many years ago, protocol analyzers were very expensive devices with special network interfaces and software that could communicate with the network interfaces to extract and present network data. In 1998, Gerald Combs was working for a small Internet service provider and he decided he was going to write his own protocol analyzer. What he wrote was originally called Ethereal, but eventually became what is now Wireshark.

The Wireshark project, which has undergone ownership changes, now has well over one thousand contributors listed on its website. As it has grown, it has continued to add a lot of functionality. Unlike tcpdump and tshark, Wireshark is a graphical interface. Where tshark will just present you a summary and the highlights of the important information, mostly from the packet headers, Wireshark includes a number of modules called dissectors. These dissectors are what make Wireshark so valuable.

NOTE There are issues running Wireshark to perform captures because it can require administrative privileges. This can lead to the potential to exploit vulnerabilities in Wireshark and provide administrative-level access to the attacker. This is one reason why it can be better to use tcpdump or tshark to capture packets and then use Wireshark to do the analysis. The problem tends to be in the protocol dissectors, which are modules that plug into Wireshark, and while tcpdump can do some protocol dissection, it doesn't do so with the use of plugin modules that allow for the extension of functionality like Wireshark does.

While it is not generally recommended to do this directly, Wireshark can be used to capture packets. In Figure 4-1, you can see Wireshark after it has been started. In this instance, I am using Wireshark 2.0.4, which has a different interface than the 1.x version, which can still be used. On the welcome page in Wireshark, you have quick links that can be used to start captures. Under the Capture header is a box where you can provide a capture filter, which can narrow the number of packets that will be captured. Below the capture filter box is the list of interfaces available to capture from. In this figure, it may be difficult to see but there is a very small graph indicating network usage next to interfaces that are actually sending or receiving network traffic. On this system, only the WiFi interface, en0, is engaged in network communication.

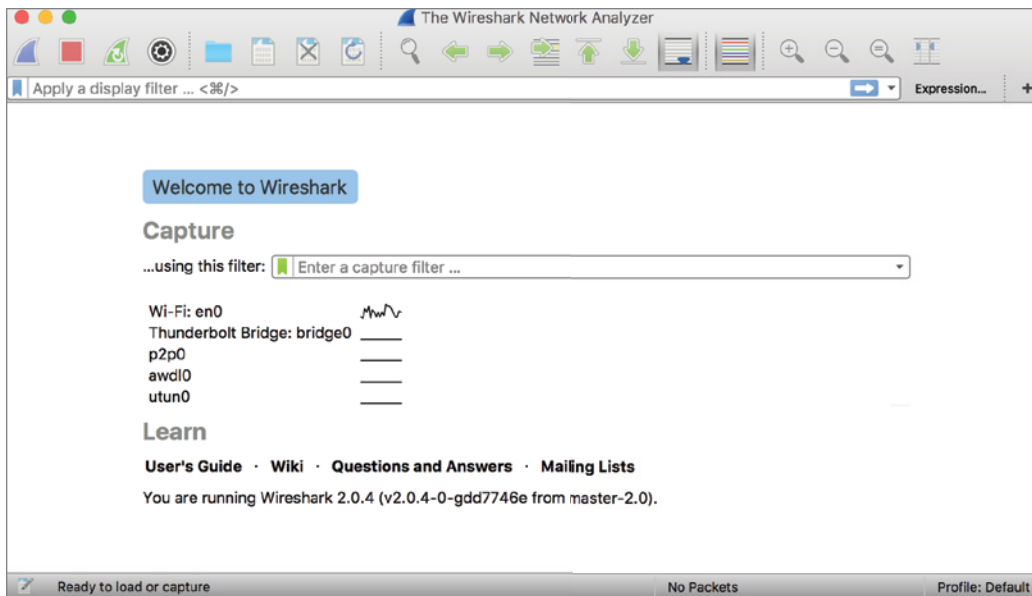


Figure 4-1: The Wireshark interface.

While using Wireshark, you can immediately start investigating traffic while it is in the process of capturing. In Figure 4-2, you see Wireshark in the middle of a capture. Three panes are worth looking at in the Wireshark window. The top is the list of frames that have been captured. The middle is the list of headers. This list can be broken out to have Wireshark explain everything in something closer to plain language that you can read. The bottom pane shows the raw data. This includes a hexadecimal dump of each byte as well as an ASCII decode of those bytes on the right-hand side. Not all of the bytes can be decoded to a printable ASCII value, which is why you may see a lot of dots (.). That just indicates that there is a byte there but it can't be printed.

Like tcpdump, Wireshark will put the network interface into promiscuous mode to capture all of the packets that pass by the interface. Putting the interface into promiscuous mode, as it is a hardware-related function, requires administrative access. Some systems will introduce a system service that handles the capture and then passes them up raw to Wireshark, which keeps Wireshark from needing to operate in administrative mode. Instead, the service that does the capturing runs in an administrative mode while Wireshark can run as any user. If Wireshark does have a vulnerability that has been exploited, the exploit only runs with the permissions of whatever user is running Wireshark. While it's still not ideal to have Wireshark exploited, it's better to do it with limited permissions rather than allowing the attacker to have complete run of the system.

No matter what program you use to capture packets, the issue is going to be how you get packets to the system. In a normal network configuration, what you are going to see is packets that are

destined for the system anyway as well as any broadcast messages. Though this may be all that you need, there will be times when you need to get access to all network traffic. This is not something that can be done using the packet capture software alone. To get everything, we need additional help.

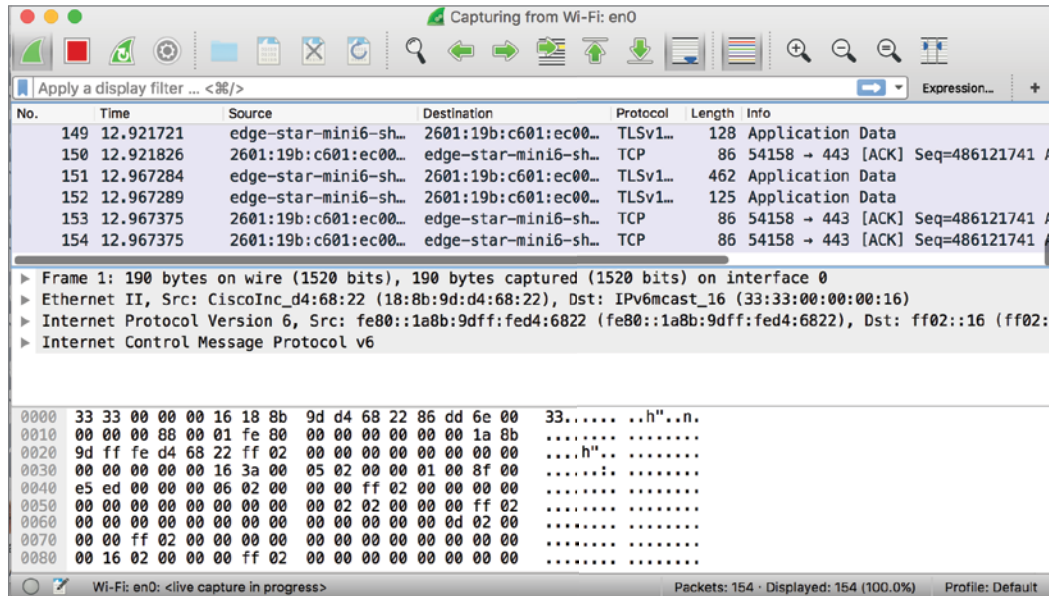


Figure 4-2: The Wireshark Interface.

Taps

In most modern networks, if a system is connected using a cable, it is with a multi-wire, copper Ethernet cable and the cable is plugged into a switch on the opposite end from the system. Years ago, systems were connected using hubs. A hub is a broadcast device when it comes to networking. Essentially, any signal that comes into a port in a hub is repeated back out to every other port in the hub. Because that consumes potential bandwidth and can impact performance on all systems on the network, switches have become far more common. A switch takes a look at the Ethernet header to determine what the destination MAC address is. Based on the destination MAC, the switch does a lookup using content addressable memory (CAM) to determine the port that the frame should be directed to. As a result, when switches are used, the only traffic that gets sent to a system is a frame that is addressed to the system in addition to broadcast traffic, which goes to all ports.

The reason for using switches is that hubs cause problems with performance. The network connection between you and the hub gets filled with network communications that aren't destined for you. You can alleviate that by using a switch, even though the hub guarantees that everyone on the

network will see what everyone else is doing. One way to get traffic from a particular network segment over a short period of time is to use a hub. It's not going to make any of the users on the other end very happy because they won't be able to use the maximum bandwidth on their connection. Instead, they will be sharing the overall bandwidth on the network with everyone else.

Another way to get traffic that is really destined to other systems is to use something called a tap, sometimes called a test access point (tap). The thing about copper network connections is that they consist of two pairs of thin copper wires. One pair is responsible for transmitting data and the other pair is responsible for receiving. Perhaps it's obvious but when two systems communicate with one another, it's necessary for the transmit wires to be copied over to the receive wires. This is done by the hub or the switch, depending on what you are using. To intercept communications over copper wires, the tap is inserted into the middle of what would normally be a single wire. As a result, you have the wire running from the computer into the tap and then another wire coming out of the tap going to the switch or router, depending on where you are capturing your traffic. The tap sits in the middle of what would otherwise be a single, uninterrupted cable, and includes a monitor port. The monitor port is where the traffic passing through the tap gets sent so it can be plugged into a device capturing traffic. You can see a basic diagram of what a tap would look like in Figure 4-3.

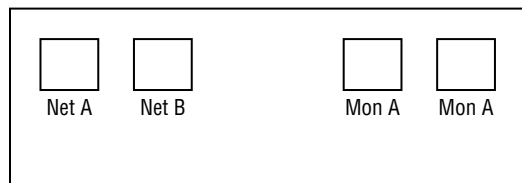


Figure 4-3: Copper network tap.

The reason for having two cables coming out is that it would be impossible to combine the two signals without causing corruption. Because both transmit pairs could be communicating simultaneously, trying to combine the two signals could end up having one signal cancelling the other out because you are talking about adding two electrical signals together. The waveforms from those electrical signals have the potential to either cancel each other out or amplify. Either way, you don't end up with two sets of data together; you end up with a single set of totally unusable data. Because you can't combine waveforms at the electrical layer, you have to have two cables that you can monitor.

In the case of copper connections that use electrical signals, the tap just repeats the signal across two separate cables. This is easy to do using basic electronics so copper taps are fairly simple. Not all connections these days are copper, however. Some connections make use of fiber cables that carry light pulses, either from a laser source or a light emitting diode (LED) source. Light is not as simple to duplicate, however. But there are other ways to capture light that you can't do with copper. Essentially, you split the signal by shaving a small portion of it off. This approach is referred to as a

passive tap because no active electronics are involved. You're just taking a portion of the light, as you might by using a mirror with the sun's rays, and diverting it somewhere else. This approach reduces the signal, because you are diverting a portion of it.

You don't have to use a passive tap, though, if you are concerned about inadequate signal at the far end. There are active taps that do re-create the light signal. This means you have 100% of the signal going to the far end. It also means that you need some additional electronics, though, to take the light signal and repeat it. As a result, active taps are more expensive than passive taps. They also require a power source because of the electronics.

Just like copper cabling, fiber also splits into transmit and receive because you can't send light in two different directions on the same fiber strand. Just as with the copper, you would end up with the potential for cancelling the signals out or amplifying them and in the end, you'd have an unusable signal. Unlike copper Ethernet cables, though, fiber cables are not combined into a single jack. Both the transmit and receive fibers would have their own terminal connector. As a result, fiber taps will have two connectors for both the entry and exit cables and then you would also have the monitor cables as well.

Port Spanning

Taps are not the only way to get traffic. With enterprise-grade switches, you may have the ability to have the switch copy traffic from one port to another. This is called *port spanning*. Cisco refers to this as using a Switch Port Analyzer (SPAN) port. Normally, you would mirror one port to another, though it's possible to mirror all of the traffic on a single network segment, a virtual local area network (VLAN), to a single port. The challenge with mirroring an entire network segment is that you are trying to cram multiple network signals down a single pipe. This is called *oversubscription*. If you have twenty 1-gigabit ports that you are mirroring to a single 1-gigabit port, you have the very real potential of losing a lot of data. If you have even a tenth of the bandwidth used on all of the connections, you have two times more data than you have the bandwidth to acquire.

NOTE Traditionally, networks are segmented physically. If you want to connect a number of systems on the same physical network, you connect them all to the same switches. The moment you connect a system to a switch with other systems, every system is visible to the others because they are all on the same layer 2 network. To segment systems using the same physical switches, you need to use logical separation. Virtual local area networks (VLANs) are a way of providing layer 2 separation within the same physical network equipment.

Not all switches are capable of supporting this sort of mirroring, but most businesses will have switches that can support this activity. Consumer-grade switches won't be able to support this, but this doesn't mean that you are out of luck if you don't have a tap or a switch that is capable of port spanning.

ARP Spoofing

On the local network, all communication is done using MAC addresses. The network itself is used to perform lookups on these addresses to resolve IP addresses to MAC addresses and vice versa. This means that every system is expected to respond when its number is called, so to speak. If one system is looking for the MAC address that belongs to an IP address, it will send out an Address Resolution Protocol request, sometimes referred to as a who-has request because that's how tcpdump and other packet capture programs render it in shorthand English. In some cases, a system that doesn't actually own that IP address will respond if it knows that the IP address will be reached by using it. This is called a proxy ARP, meaning that one system is acting as a proxy for another system, promising to pass the message along, much like kids in elementary school might.

To speed things up, every system will typically cache an ARP resolution in a table locally to prevent having to do a request the next time it needs to communicate with a particular address. A system may do this even if it just happens to notice an ARP message go by that it didn't request. This is commonly done in the implementation of the TCP/IP protocols on systems because it's efficient and can improve response times.

The problem with ARP is there is simply no way to verify that the messages being sent on the network are legitimate. I might, for example, send out a message in response to a request that I see go by. If my response beats the legitimate host, I will win and the message will come to me. Attackers may use this technique, called *ARP spoofing*, to get messages that are legitimately meant for someone else. It actually gets more complicated than this, though. I may not even bother waiting for someone to ask to know who a particular address belongs to. I may just send out a message telling everyone that a particular IP address belongs to my MAC address. When an ARP response is sent without any ARP request, the response is called a *gratuitous ARP*.

A gratuitous ARP is sent to everyone on the network using the broadcast MAC address ff:ff:ff:ff:ff:ff. Because systems are commonly set to just cache ARP mappings to save time later on, these messages are cached and the receiving systems have no way of knowing that they are bogus. An attacker can actually send these gratuitous ARP messages for any number of systems on the network. It may not just be a single host. Ideally, if I wanted to capture a particular communication flow, I would need to spoof both ends of the connection or else I'm only getting half of the conversation. I can easily go beyond just a pair of hosts, however. Using this technique, I could have all the traffic on the network sent to me. You can see this in Listing 4-5.

Listing 4-5: tcpdump Output of ARP Spoofing

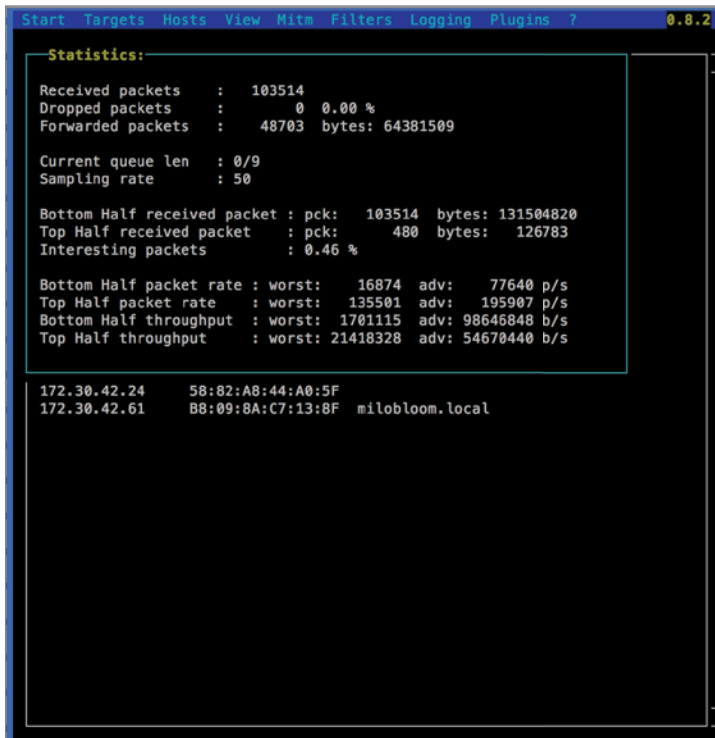
```
20:42:57.234477 ARP, Reply 172.30.42.23 is-at f4:5c:89:b7:2c:89
(oui Unknown), length 28
20:42:57.234591 ARP, Reply 172.30.42.1 is-at f4:5c:89:b7:2c:89
(oui Unknown), length 28
20:42:57.245649 ARP, Reply 172.30.42.20 is-at f4:5c:89:b7:2c:89
(oui Unknown), length 28
20:42:57.245780 ARP, Reply 172.30.42.1 is-at f4:5c:89:b7:2c:89
(oui Unknown), length 28
```

```

20:42:57.256978 ARP, Reply 172.30.42.19 is-at f4:5c:89:b7:2c:89
oui Unknown), length 28
20:42:57.257038 ARP, Reply 172.30.42.1 is-at f4:5c:89:b7:2c:89
(oui Unknown), length 28
20:42:57.267220 ARP, Reply 172.30.42.12 is-at f4:5c:89:b7:2c:89
(oui Unknown), length 28
20:42:57.267265 ARP, Reply 172.30.42.1 is-at f4:5c:89:b7:2c:89
(oui Unknown), length 28
20:42:57.278285 ARP, Reply 172.30.42.2 is-at f4:5c:89:b7:2c:89
(oui Unknown), length 28
20:42:57.278349 ARP, Reply 172.30.42.1 is-at f4:5c:89:b7:2c:89
(oui Unknown), length 28

```

Looking closely at this, you can see that several IP addresses have been said to belong to a single MAC address. Fortunately, this is not complicated. A number of programs are capable of performing this particular technique. The technique is sometimes called ARP poisoning because what you are doing is corrupting (poisoning) the ARP cache on different hosts on the network. This particular ARP poisoning was done using a program called Ettercap. Ettercap comes with different interfaces. Once of them is fully graphical. You can also run it using just the command-line interface, specifying your targets. In Figure 4-4, you can see the console-based interface, which can be used if you need to do ARP spoofing but you don't have the ability to use a graphical interface.



```

Start Targets Hosts View Mitm Filters Logging Plugins ? 0.8.2

Statistics:
Received packets : 103514
Dropped packets : 0 0.00 %
Forwarded packets : 48703 bytes: 64381509
Current queue len : 0/9
Sampling rate : 50

Bottom Half received packet : pck: 103514 bytes: 131504820
Top Half received packet : pck: 480 bytes: 126783
Interesting packets : 0.46 %

Bottom Half packet rate : worst: 16874 adv: 77640 p/s
Top Half packet rate : worst: 135501 adv: 195907 p/s
Bottom Half throughput : worst: 1701115 adv: 98646848 b/s
Top Half throughput : worst: 21418328 adv: 54670440 b/s

172.30.42.24 58:82:A8:44:A0:5F
172.30.42.61 B8:09:8A:C7:13:8F milobloom.local

```

Figure 4-4: Ettercap in Curses mode.

One of the challenges with ARP poisoning is that once you have the packet, it isn't going to the intended destination. This causes two problems. The first is that people will start to get suspicious that the network isn't working. Second, even if the users don't really catch on quickly, the systems that are communicating will because they will be expecting regular responses if they are communicating over TCP, which will commonly be the case. If they stop getting response messages, they will determine that the connection has failed and just tear it down. This means that to continue getting all the messages that you want, you need to find a way to make sure all the messages you are getting get forwarded to their intended target. This behavior is not common for most operating systems by default because it effectively turns your system into a router, since you are taking packets in and then forwarding them back out again on the same interface.

While ARP spoofing may not be the best approach to collecting network information, it is one approach to making sure you can collect network information. If you are using ARP spoofing, you can use any packet capture program to bring the packets in because they are being sent to you. Any ARP poisoning program can be used alongside a packet capture program to acquire packets from across the network, regardless of whether you are on a switched network, because all of the systems are being told to just send you everything and they will comply.

Passive Scanning

Another technique to keep in mind is passive scanning. This particular approach just watches the data that passes across the network and reports specific details to you. With this approach, you aren't gathering all of the information that you would using a full packet capture that you were analyzing in Wireshark. You also aren't getting just a summary of header information, as you would get from tcpdump or tshark. Instead, a passive scanner will present you with useful details from all of the different layers. A passive scanner will just run quietly, observing data that is passing across the network interface. A well-known scanner that uses this technique is p0f. Using p0f, you will end up with output that looks like the results shown in Listing 4-6. The output you receive will vary, of course, depending on what traffic it is seeing.

Listing 4-6: p0f Output

```
.-[ 172.30.42.16/55629 -> 199.58.85.40/80 (http request) ]-
|
| client    = 172.30.42.16/55629
| app      = Chrome 51.x or newer
| lang     = English
| params   = none
| raw_sig  = 1:Host,Connection=[keep-alive],
Upgrade-Insecure-Requests=[1],User-Agent,Accept=[text/html,
application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8],
?Referer,Accept-Encoding=[gzip, deflate, sdch],Accept-Language=[en-US,
en;q=0.8]:Accept-Charset,Keep-Alive:Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko)
```

```

Chrome/53.0.2785.116 Safari/537.36
|
|-----
|
|.-[ 172.30.42.16/55630 -> 199.58.85.40/80 (syn+ack) ]-
|
|  server    = 199.58.85.40/80
|  os        = Linux 3.x
|  dist      = 11
|  params    = tos:0x08
|  raw_sig   = 4:53+11:0:1460:mss*10,4:mss,sok,ts,nop,ws:df:0
|
|-----
|
|.-[ 172.30.42.16/55630 -> 199.58.85.40/80 (mtu) ]-
|
|  server    = 199.58.85.40/80
|  link      = Ethernet or modem
|  raw_mtu   = 1500
|
|-----
|
|.-[ 172.30.42.16/55629 -> 199.58.85.40/80 (uptime) ]-
|
|  server    = 199.58.85.40/80
|  uptime    = 26 days 9 hrs 36 min (modulo 49 days)
|  raw_freq  = 1000.00 Hz
|
|-----

```

What we can tell from this output is that there is a browser that is open and communicating with Google. The very first block of information tells us that the browser is Google Chrome, and includes the version number. We get the source and destination IP addresses and the source and destination port numbers. Looking more closely at the raw signature (`raw_sig`), it appears this browser is running on a Mac OS X system. This particular piece of information isn't that surprising because it was captured on the system that I am writing on and it happens to be a macOS system. Because it's providing information about the system where p0f is running, that's less interesting or useful. However, p0f isn't done.

Looking at the next block of information, it appears that the destination address belongs to a system that is running Linux and the kernel version is in the 3.x line. p0f was able to determine this based on the SYN/ACK message that it observed. The message type is in the very top line of the message block. Further down, p0f was able to determine that the remote system has been up for over 26 days. By really pulling all of the packets apart and doing deep investigation, it has been able to provide a lot of information that we may miss by simply doing packet captures and looking at them.

This is not to discount using packet capture programs, however. A good packet capture program, like Wireshark, will provide a lot of tools that we will be able to make use of to do network investigations.

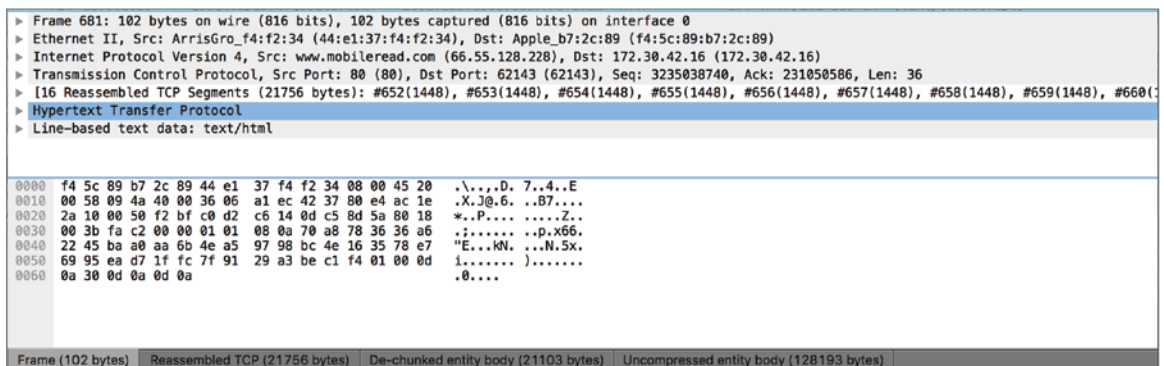
Packet Analysis with Wireshark

Ethereal, later Wireshark, started out as a decent, free alternative to very expensive packet capture software, and it has evolved into a very mature and useful program. Using Wireshark, we can extract a lot of information because Wireshark will do a lot of analysis for us without us needing to go digging into messages one at a time. Wireshark keeps track of a lot of information as it gathers each frame and it also does a lot of the decoding and dissection for us. Additionally, it will provide a lot of statistics about the capture, which can be very useful for certain types of investigations. While the capture is displayed as individual frames, Wireshark will also gather all of the related frames together for us, presenting us with a plaintext view of the data from the conversation. This can save us from trying to pick pieces out of individual frames to add them to information from other frames.

In some cases, the communication will have a number of files that are being transmitted. Consider a typical web page view. In addition to the text that you are seeing on the page, often graphics files and potentially other files are brought in to create the entire experience. You could extract the data from each individual frame and collect it all together or you could just let Wireshark do that for you. Wireshark has a number of other capabilities, and we will be spending some time in subsequent chapters going over some of those. For now, let's talk about some of the basics that we will need as we go forward.

Packet Decoding

Perhaps the most important feature that Wireshark offers is the ability to decode the packet, providing details in plaintext. This saves us from having to do decoding on our own. As an example, you can see part of the decoding that Wireshark does in Figure 4-5. At the top of the screen capture, Wireshark has broken the different layers of the packet into its components. Each line constitutes a different layer and set of headers or data. You may also note that I said packet and not frame. The reason is that Wireshark has very helpfully collected all of the frames to present it all together without much additional work on my part.



The first thing to notice, as mentioned previously, is that you get a line for each layer, starting with the physical layer. Wireshark provides a summary of how many bytes were captured and which interface they were captured on. When you get to the data link layer, you will see the MAC addresses but Wireshark has done us a favor by looking up the vendor from the first three bytes, which constitute the organizationally unique identifier (OUI). In the network layer, Wireshark has provided a lookup of the source IP address. What we don't see is Wireshark providing a lookup at the transport layer. This is configurable, however. Going to the View menu, as you can see in Figure 4-6, we can have Wireshark either provide us with the name resolutions, or not.

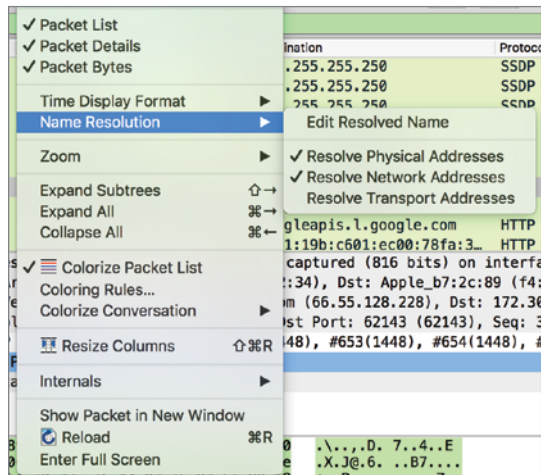
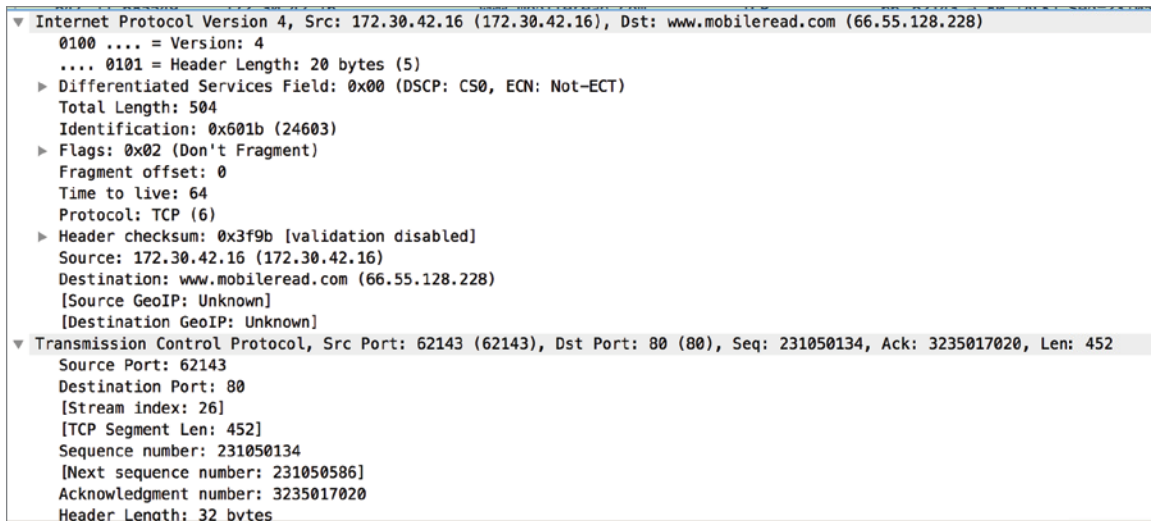


Figure 4-6: Name Resolution view.

One thing to keep in mind is that some of these name resolutions will require additional network traffic. In the case of the network layer, for example, it's not as though `www.mobileread.com` was provided in the packet that was captured. Instead, there was an IP address and Wireshark did the lookup. That lookup required Wireshark to initiate a DNS request. In the middle of the capture, you will see these DNS lookups as Wireshark observes traffic with addresses that need to be resolved.

TIP To limit the amount of traffic captured, disable name resolution while you are capturing, whether that's in Wireshark, tcpdump, tshark, or another capture program.

Digging a little more deeply into the packet, we can look at how Wireshark has broken out all of the pieces of the different headers. In Figure 4-7, you can see how each of the different fields in the IP header has a line telling you what the field is and the value of that field. This includes breaking out the value of each of the flag fields, as in the fragmentation flags. If you want to see the individual flag bit values, you can click the little arrow on the left-hand side of that field. Otherwise, Wireshark has provided the value of the byte and a brief explanation of what that value means. In the flags byte, we have a value of `0x02`, which means that the Don't Fragment bit has been set.



```

▼ Internet Protocol Version 4, Src: 172.30.42.16 (172.30.42.16), Dst: www.mobileread.com (66.55.128.228)
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ► Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 504
    Identification: 0x601b (24603)
  ► Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
  ► Header checksum: 0x3f9b [validation disabled]
    Source: 172.30.42.16 (172.30.42.16)
    Destination: www.mobileread.com (66.55.128.228)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▼ Transmission Control Protocol, Src Port: 62143 (62143), Dst Port: 80 (80), Seq: 231050134, Ack: 3235017020, Len: 452
  Source Port: 62143
  Destination Port: 80
  [Stream index: 26]
  [TCP Segment Len: 452]
  Sequence number: 231050134
  [Next sequence number: 231050586]
  Acknowledgment number: 3235017020
  Header Length: 32 bytes

```

Figure 4-7: Header field values.

Similarly, you can see the different header fields for TCP have been decoded for us. In this view, we can see the actual value of the sequence and acknowledgment numbers. In most cases, however, Wireshark will take care of those values for us, providing relative values instead. This means that Wireshark will keep track of the base value and present that to us as if it were 0. Every subsequent value will be the amount incremented over the base value. In some cases, it may be easier for you to see the actual values that are being used so you can tell Wireshark to provide the values that are sent rather than the values that Wireshark will compute. Because the values are so large, it's generally easier to just allow Wireshark to compute the value rather than trying to keep track of it manually.

Wireshark will also provide you with the checksum, which is the value that is computed across different sections of the packet to ensure it hasn't been corrupted. You can have Wireshark compute the checksum or not. Disabling checksum validation can improve performance very slightly. If checksum validation is enabled, Wireshark will be able to tell you whether the checksum is valid. By default, Wireshark will not compute the checksum for you because often modern operating systems along with the network drivers will offload the checksum computation to the network hardware. This can end up with it appearing that the checksums are incorrect.

In addition to all of the decoding, Wireshark will also provide the raw packet data. This is shown in the bottom pane of the Wireshark capture window. You can see in Figure 4-8 the raw capture data from the packet that was shown earlier. This provides the raw bytes on the left-hand side, displayed in hexadecimal. On the right, in a traditional hexadecimal dump format, is the ASCII decode. The ASCII decode shows any printable characters. Otherwise, all you will see is a dot because the character can't be printed. What we can see in this particular pane, though, in addition to seeing the raw

data, is where each value is located. As an example, the IP header checksum field has been selected. This is the 9th and 10th byte in the first line. At the very bottom, in the status line, Wireshark tells us the field that has been selected.

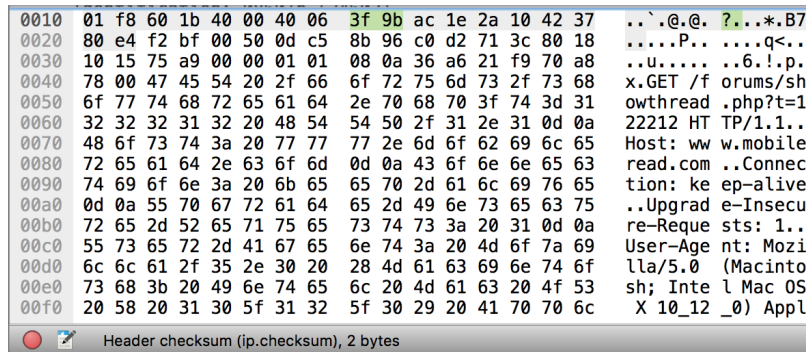


Figure 4-8: Raw packet data.

This is just a very quick overview of the different areas for which Wireshark does decoding of the packet for us. In addition to decode details, Wireshark keeps track of a lot of other useful information for us. We will take a look at some statistics that Wireshark provides a little later on in the Statistics section. One thing you may notice if you have started to do any captures on your own is that Wireshark captures a lot of information. The more information you capture, the more disk space you are using, if you are storing. Also, the more you capture, the more you have to wade through as you are looking for specific information. Having a way to narrow down whether what you are capturing or looking at is helpful. Wireshark offers a very rich filtering capability.

Filtering

Wireshark will allow for the use of both display and capture filters. You have a number of ways to perform filtering. In the display filter box above the list of frames captured, Wireshark will offer suggestions when you start typing. Filtering using protocols is easy. All you need to do is type the protocol, as in ICMP, HTTP, TCP, or UDP. Wireshark will then display all frames that match the protocol that you have specified. Beyond that, though, you can get into the specifics of the protocol. As an example, you can indicate that you want to just display packets where the IP address is 172.30.42.1. To display only those packets, you could type `ip.addr == 172.30.42.1`. This tells Wireshark that either the source or destination addresses should be the IP address specified. Note that Wireshark uses two equal signs to specify a match.

If you only want to see the packets where the source address is a particular address, you can specify `ip.src_host` as the parameter you want to match. Different protocols are capable of breaking out different parameters to filter on. As an example, if you are trying to filter through a lot of web

traffic, you can specify different parameters of HTTP. You may only want to look for POST methods, meaning you are looking for where the client is sending information to the server. To filter based on that, you would use `HTTP.request.method == POST`. This would show you only packets that include a message to a web server where the request is a POST rather than a GET or any other method.

The filter box is not the only place that you can specify a filter, however. You can automatically create a filter by right-clicking any packet in your list. You will get a context menu that allows you to choose either Apply as Filter or Prepare a Filter. In either case, you will be able to use that specific packet to either include or exclude. The difference between Apply and Prepare is that with Apply, the filter is immediately applied, whereas selecting Prepare places the prepared filter into the filter box so you can edit as you would like.

Wireshark keeps track of a lot of information and to perform some of these types of filters, it identifies communication streams. When system A starts to communicate with system B, Wireshark will know, based on the same source and destination IP addresses and port numbers, that a number of frames may belong to that particular stream. Because it keeps track of all of that information, Wireshark can present you with specific conversations without you having to do a lot of work.

Statistics

Wireshark digs through the entire packet as it is displaying and decoding packets. In the process, it learns a lot about the entire data set. Using the various statistics that Wireshark provides, we can get a much better look at the entire capture. You can see the entire Statistics menu in Figure 4-9. In the middle of the menu are a number of protocol-specific statistics. These may not be useful to you, depending on whether you have captured any of those protocols. As a result, we can start with a broader view of the entire capture.

Two statistics views display similar information: the Conversations view and the Endpoints view provide specific details about the number of packets that are being transmitted. In the Endpoints view, this is displayed solely by the address that is responsible for the traffic. In the Conversations view, you will see both endpoints. Figure 4-10 shows the Conversations view. The difference between Conversations and Endpoints is that the Endpoints view only shows the endpoint and not the aspects that are related to the conversation. In the Conversations view, you will see the counts in both directions with addresses associated with both ends. In the Endpoints view, you will see A ⇌ B and B ⇌ A counts without distinguishing between different B hosts. All those counts show is how many bytes are going out versus coming in.

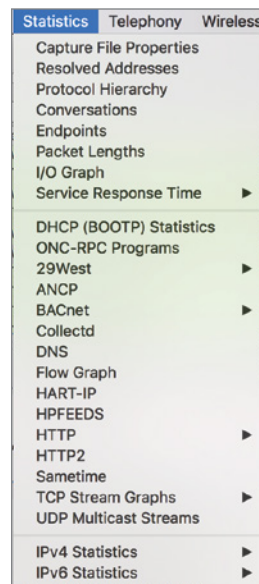


Figure 4-9: Statistics menu.

Wireshark · Conversations · wireshark_pcapng_en0_20161006204329_YTH1P1

Ethernet · 69 IPv4 · 84 IPv6 · 63 TCP · 145 UDP · 335

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration
SamsungE_86:2f:40	IPv4mcast_7f:ff:fa	21	9981	21	9981	0	0	19.251863000	60.012393
Parallel_9e:7a:88	IPv6mcast_16	45	4090	45	4090	0	0	4.915150000	81.513340
Parallel_9e:7a:88	IPv4mcast_fc	53	3532	53	3532	0	0	5.017684000	81.819588
Parallel_9e:7a:88	IPv4mcast_7f:ff:fa	2	120	2	120	0	0	5.017687000	0.409680
Parallel_9e:7a:88	IPv4mcast_fb	1	60	1	60	0	0	5.017828000	0.000000
Parallel_9e:7a:88	IPv6mcast_01:00:03	28	2592	28	2592	0	0	10.035642000	76.801446
Parallel_9e:7a:88	IPv4mcast_02	11	660	11	660	0	0	10.855118000	75.572953
Parallel_9e:7a:88	Broadcast	3	276	3	276	0	0	25.088955000	1.536024
IPv4mcast_01	ArrisGro_f4:f2:34	30	1320	0	0	30	1320	4.710396000	0.004427
IPv4mcast_02	Microsof_44:a0:5f	1	60	0	0	1	60	65.537425000	0.000000
IPv4mcast_02	Apple_8d:ad:ff	1	46	0	0	1	46	82.126463000	0.000000
IPv4mcast_07	SamsungE_6a:68:1f	46	10 k	0	0	46	10 k	0.000000000	92.469356
IPv4mcast_fa	Apple_36:d6:aa	1	60	0	0	1	60	5.120576000	0.000000
IPv4mcast_fb	Raspberr_11:c2:5a	1	46	0	0	1	46	4.812958000	0.000000
IPv4mcast_fb	Microsof_44:a0:5f	1	60	0	0	1	60	4.915299000	0.000000
IPv4mcast_fb	Apple_1e:6b:30	1	60	0	0	1	60	5.017565000	0.000000
IPv4mcast_fb	Apple_b7:2c:89	2	164	0	0	2	164	6.834826000	27.057593

☒ Name resolution ☐ Limit to display filter

Help Copy Follow Stream... Graph... Conversation Types Close

Figure 4-10: Conversations view.

Both of these statistics views provide both byte and packet counts. Because packets are variable length, just having a byte count won't tell you the number of packets. You similarly can't get a packet count based on the number of bytes. Both pieces of information can be important. You may also note that you can enable address resolution. This will provide address resolution of the vendor for the MAC address and if you click other tabs, you would be able to see names where Wireshark was able to look them up. As an example, Wireshark would provide the hostname that belonged to an IP address or the name of a port that had been used, assuming it was one of the well-known ports.

While you can sort the captured frames to organize the list by sender or recipient, using either the Conversations or Endpoints view is more convenient to be able to get a complete list of the endpoints and the conversations that are happening. One reason for that is the number of frames. Even if you are sorting your list of frames by the sender, each sender may have dozens or hundreds of frames. You would need to still read down through the entire list to make note of the different senders you have captured. Using the Endpoints statistics is much more convenient. Once you know who your endpoints are, you could move on to the Conversations view to determine who those endpoints are communicating with.

Using the Conversations view, you can get a look at the ports that are being used. If you click the TCP tab at the top, you will get a list of the conversations between source and destination and on that tab, you will see the ports on both the source and destination side. This may be able to tell you what is happening between these two systems, but it could also be that the ports don't tell you what you need to know. Another way of looking at the packet capture is to look at the Protocol Hierarchy view. This view, seen in Figure 4-11, can not only tell you where all of the data is being transmitted, but can also provide you with some outliers as well. Skimming through the packet capture may not tell you that there is an odd or unexpected protocol being used.

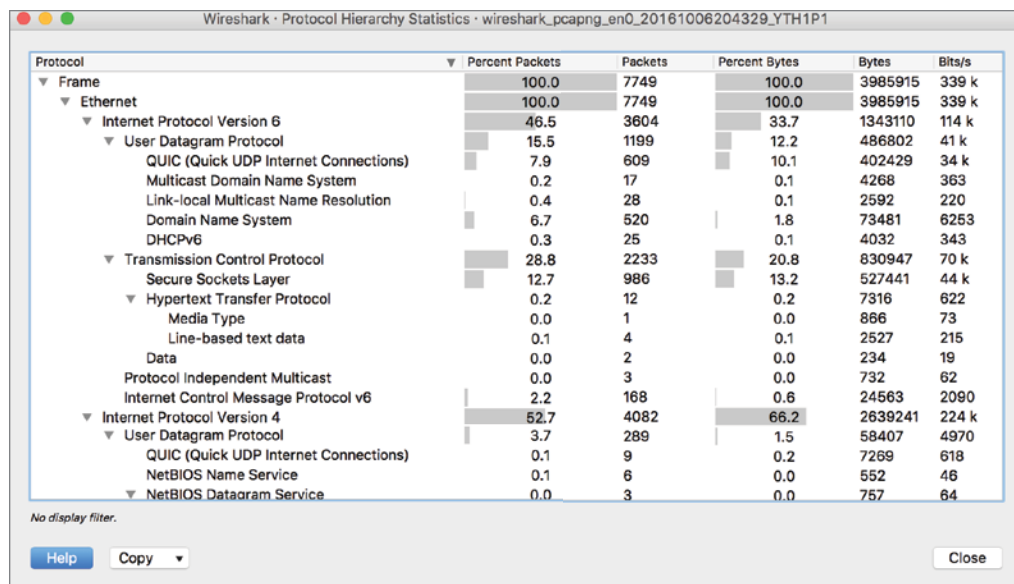


Figure 4-11: Protocol Hierarchy view.

As an example, scrolling down through the Protocol Hierarchy view, there was an entry for the Internet Control Message Protocol (ICMP). This seemed unexpected, so right-clicking that protocol allows me to create a filter. If you select Apply as Filter, Wireshark will prepare a filter for you based on the characteristics of what you have selected. Since I have ICMP highlighted, applying the filter based on that protocol places ICMP into the filter box at the top of the capture window. Wireshark will then only display frames that are using ICMP, as the display filter has indicated, and all other frames will be hidden.

Another statistics view that can be very useful is the IP statistics. As you saw in the Statistics menu, at the bottom there are entries for IPv4 Statistics and IPv6 Statistics. When you select one of those, you get a flyout menu offering the choice of All Addresses; Destinations and Ports; and IP Protocol Types or Source and Destination Addresses. Selecting Destinations and Ports shows you a view like the one in Figure 4-12. The advantage of this view is that you are able to see what the burst rate for the traffic is. The burst rate tells you what the peak speed is so you know how fast the traffic was moving at its fastest. This does not mean that it was sustained at that rate, just that it peaked there.

In the last column on the right, you will also see the time that the burst started. Looking at the capture window in Wireshark, the time, relative to the start of the capture, is in the second column from the left. Using this column, you can identify the point in the capture where the traffic burst started. Once you have identified the point in the capture where the burst starts, you may want to be able to identify the complete traffic stream. Fortunately, there are ways to do that using Wireshark.

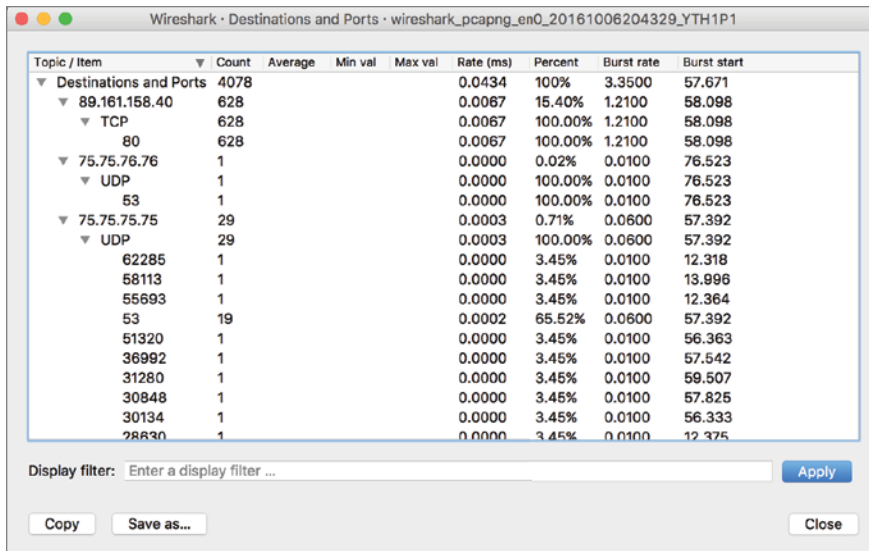


Figure 4-12: IPv4 statistics view.

Following Streams

As mentioned previously, Wireshark keeps track of the different streams that have been captured. To filter based on just one stream, all you need to do is right-click one of the frames from that stream and select **Apply as Filter using Selected**. That will create a filter based on specific details from the packet, which may include the IP address. That's not the only way, though, that you can filter on a stream. Additionally, while TCP is often referred to as a stream-based communication protocol, Wireshark also keeps track of the communications that happen over UDP and can identify a UDP stream as well. Wireshark will use internally stored information to do this filtering.

Right-clicking one of the frames that's potentially part of a stream you are interested in will allow you to select **Follow Stream** from the context menu. This will immediately apply a filter by identifying the stream that Wireshark has attached the frame to. In Figure 4-13, you can see the filter that Wireshark has applied is `tcp.stream eq 69`. This is not the sort of information that you will find looking at the packet itself without Wireshark. Only Wireshark knows that stream number. If you were to open up the TCP header information, you would find a value that Wireshark has applied indicating the stream index. In addition to applying a filter using the stream index, Wireshark will also collect all of the data from the packets and present it in a single view. You can see that in Figure 4-13 as well.

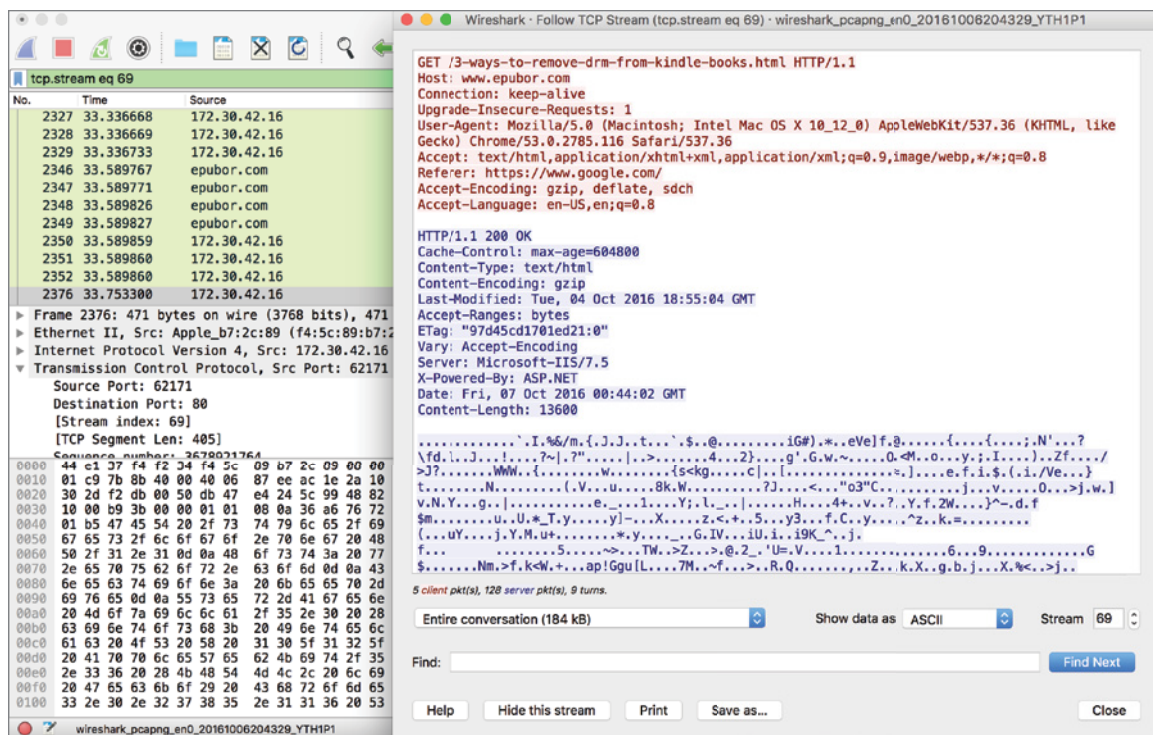


Figure 4-13: Follow stream.

While the plaintext of the capture here has been rendered by doing an ASCII translation, Wireshark is capable of performing a decode in other ways as well, including EBCDIC, C strings, hex dumps, UTF-8, Yet Another Markup Language (YAML), or raw. In this case, the web traffic was sent using ASCII for the encoding, but that won't always be the case depending on where you are in the world and the protocol that may be in use. Wireshark can help with just about any decode type that you may run across.

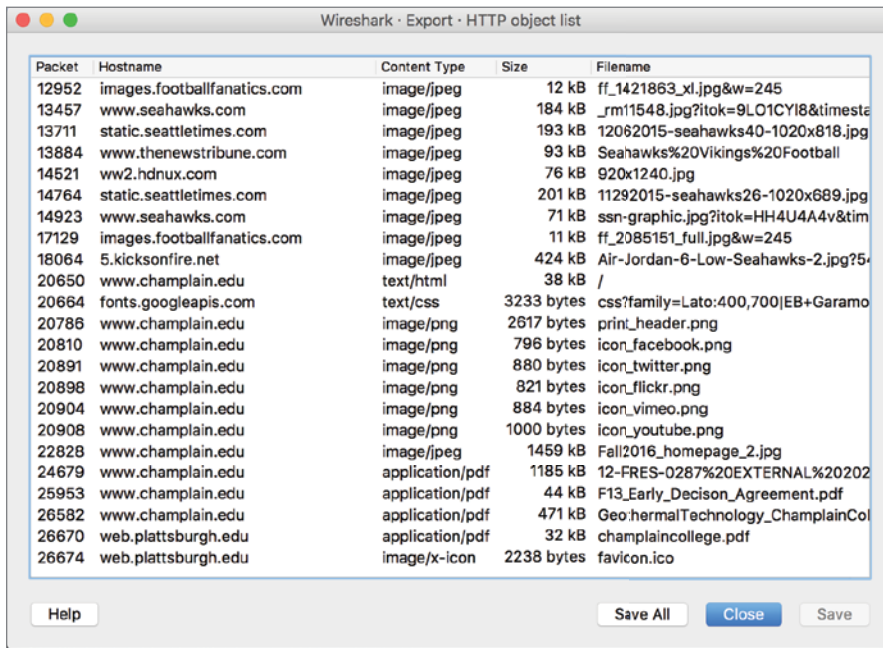
Following TCP streams works well if you have any plaintext that you want to be able to read across a number of frames or packets. It's considerably harder to try to extract all of that data manually. It's not impossible, because you can extract the data from each packet and put it together yourself, but it's quite a bit harder. If you have a text file that's being transmitted, you can still use the Follow Stream feature to get the contents. If you have binary content, that's harder to deal with. Wireshark can handle that, though.

Gathering Files

There likely will be a lot of files in a packet capture. This may not be immediately apparent, however. If you were just capturing web traffic, for instance, it may not occur to you that the data

that is coming across the wire is a collection of files. Every image is another file. A web page may also contain JavaScript files if they are imported into the page rather than being included directly into the Hypertext Markup Language (HTML). Often, pages will include a number of other files as well, including Portable Document Format (PDF) files or Java applets, Flash scripts, or a number of other file types. This is just from web traffic. On a corporate network, files are being sent back and forth on a regular basis between file shares or as print documents.

Using Wireshark, you can identify all of the files from a packet capture. You can do this by just going to the File menu and selecting Export Objects. That gives you the option to select protocols that you want Wireshark to look at, including HTTP or the Server Message Block (SMB), which is a protocol that Windows systems will use to share files. Once you select which protocol you want, you will get a window like the one in Figure 4-14, which shows a list of all of the files that have been discovered being transmitted over HTTP from this capture. This list is mostly images that were grabbed from a couple of web searches, but it also contains PDFs.

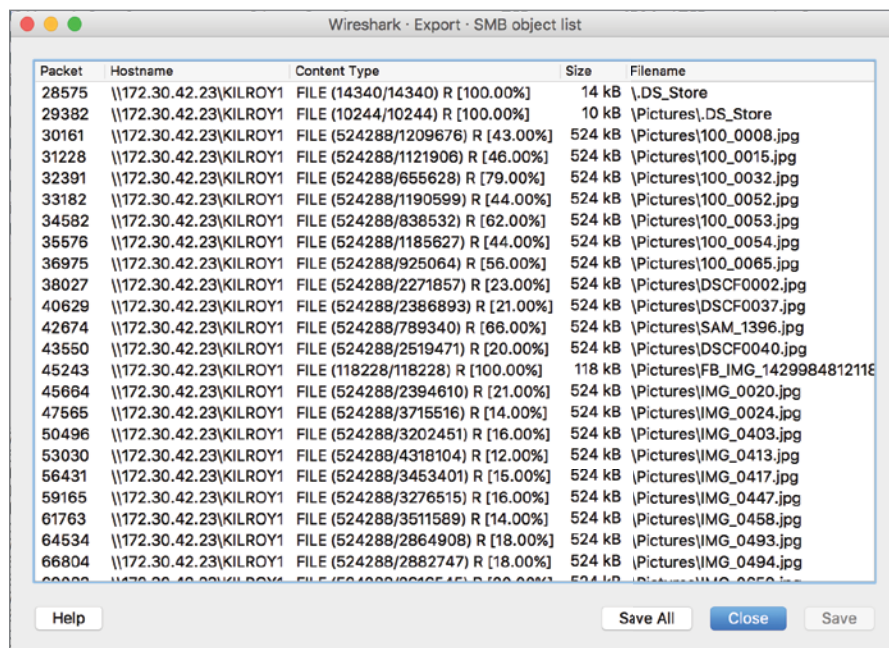


Packet	Hostname	Content Type	Size	Filename
12952	images.footballfanatics.com	image/jpeg	12 kB	ff_1421863_xl.jpg&w=245
13457	www.seahawks.com	image/jpeg	184 kB	_rm11548.jpg?itok=9LO1CYI8&time
13711	static.seattletimes.com	image/jpeg	193 kB	12062015-seahawks40-1020x818.jpg
13884	www.thenewstribune.com	image/jpeg	93 kB	Seahawks%20Vikings%20Football
14521	ww2.hdnux.com	image/jpeg	76 kB	920x1240.jpg
14764	static.seattletimes.com	image/jpeg	201 kB	11292015-seahawks26-1020x689.jpg
14923	www.seahawks.com	image/jpeg	71 kB	ssn-graphic.jpg?itok=HH4U4A4v&tim
17129	images.footballfanatics.com	image/jpeg	11 kB	ff_2085151_full.jpg&w=245
18064	5.kicksonfire.net	image/jpeg	424 kB	Air-Jordan-6-Low-Seahawks-2.jpg?5-
20650	www.champlain.edu	text/html	38 kB	/
20664	fonts.googleapis.com	text/css	3233 bytes	css?family=Lato:400,700 EB+Garamo
20786	www.champlain.edu	image/png	2617 bytes	print_header.png
20810	www.champlain.edu	image/png	796 bytes	icon_facebook.png
20891	www.champlain.edu	image/png	880 bytes	icon_twitter.png
20898	www.champlain.edu	image/png	821 bytes	icon_flickr.png
20904	www.champlain.edu	image/png	884 bytes	icon_vimeo.png
20908	www.champlain.edu	image/png	1000 bytes	icon_youtube.png
22828	www.champlain.edu	image/jpeg	1459 kB	Fall2016_homepage_2.jpg
24679	www.champlain.edu	application/pdf	1185 kB	12-FRES-0267%20EXTERNAL%20202
25953	www.champlain.edu	application/pdf	44 kB	F13_Early_Decision_Agreement.pdf
26582	www.champlain.edu	application/pdf	471 kB	Geo:hermalTechnology_ChamplainCol
26670	web.plattsburgh.edu	application/pdf	32 kB	champlaincollege.pdf
26674	web.plattsburgh.edu	image/x-icon	2238 bytes	favicon.ico

Figure 4-14: Exporting files.

Once you have the list of files, you can select a single file and export that from the capture or you could just export all of the files from the capture. Again, though, this is protocol specific. Using a different protocol, like SMB, we get a different list from the same capture. Figure 4-15 shows a list of files that were identified as being transmitted over SMB. Just as in the previous case, you can see

where the files came from. The hostname column shows the host and the file share that each file came from. You will also see the content type. In the case of web content, you will see the Multipurpose Internet Mail Extensions (MIME) type. SMB doesn't use MIME types, though, so all we see in the content type column is FILE. The filename itself is in the last column.



Packet	Hostname	Content Type	Size	Filename
28575	\\172.30.42.23\KILROY1	FILE (14340/14340) R [100.00%]	14 kB	_DS_Store
29382	\\172.30.42.23\KILROY1	FILE (10244/10244) R [100.00%]	10 kB	\Pictures_DS_Store
30161	\\172.30.42.23\KILROY1	FILE (524288/1209676) R [43.00%]	524 kB	\Pictures\100_0008.jpg
31228	\\172.30.42.23\KILROY1	FILE (524288/1121906) R [46.00%]	524 kB	\Pictures\100_0015.jpg
32391	\\172.30.42.23\KILROY1	FILE (524288/655628) R [79.00%]	524 kB	\Pictures\100_0032.jpg
33182	\\172.30.42.23\KILROY1	FILE (524288/1190599) R [44.00%]	524 kB	\Pictures\100_0052.jpg
34582	\\172.30.42.23\KILROY1	FILE (524288/838532) R [62.00%]	524 kB	\Pictures\100_0053.jpg
35576	\\172.30.42.23\KILROY1	FILE (524288/1185627) R [44.00%]	524 kB	\Pictures\100_0054.jpg
36975	\\172.30.42.23\KILROY1	FILE (524288/925064) R [56.00%]	524 kB	\Pictures\100_0065.jpg
38027	\\172.30.42.23\KILROY1	FILE (524288/2271857) R [23.00%]	524 kB	\Pictures\DSCF0002.jpg
40629	\\172.30.42.23\KILROY1	FILE (524288/2386893) R [21.00%]	524 kB	\Pictures\DSCF0037.jpg
42674	\\172.30.42.23\KILROY1	FILE (524288/789340) R [66.00%]	524 kB	\Pictures\SAM_1396.jpg
43550	\\172.30.42.23\KILROY1	FILE (524288/2519471) R [20.00%]	524 kB	\Pictures\DSCF0040.jpg
45243	\\172.30.42.23\KILROY1	FILE (118228/118228) R [100.00%]	118 kB	\Pictures\FB_IMG_1429984812116.jpg
45664	\\172.30.42.23\KILROY1	FILE (524288/2394610) R [21.00%]	524 kB	\Pictures\IMG_0020.jpg
47565	\\172.30.42.23\KILROY1	FILE (524288/3715516) R [14.00%]	524 kB	\Pictures\IMG_0024.jpg
50496	\\172.30.42.23\KILROY1	FILE (524288/3202451) R [16.00%]	524 kB	\Pictures\IMG_0403.jpg
53030	\\172.30.42.23\KILROY1	FILE (524288/4318104) R [12.00%]	524 kB	\Pictures\IMG_0413.jpg
56431	\\172.30.42.23\KILROY1	FILE (524288/3453401) R [15.00%]	524 kB	\Pictures\IMG_0417.jpg
59165	\\172.30.42.23\KILROY1	FILE (524288/3276515) R [16.00%]	524 kB	\Pictures\IMG_0447.jpg
61763	\\172.30.42.23\KILROY1	FILE (524288/3511589) R [14.00%]	524 kB	\Pictures\IMG_0458.jpg
64534	\\172.30.42.23\KILROY1	FILE (524288/2864908) R [18.00%]	524 kB	\Pictures\IMG_0493.jpg
66804	\\172.30.42.23\KILROY1	FILE (524288/2882747) R [18.00%]	524 kB	\Pictures\IMG_0494.jpg

Figure 4-15: Files shared over SMB.

The one case for which this won't work is where the content has been encrypted. Encryption may use the Secure Sockets Layer (SSL) or the Transport Layer Security (TLS) encryption mechanisms. Web traffic that has been encrypted can't be decrypted using Wireshark unless you have the keys. In the majority of cases, you won't be able to get the keys to perform the decryption. If you were able to easily obtain the keys, the encryption wouldn't be of much value.

In cases where files are being transmitted across different protocols, you may need to go through the process of exporting the data from the individual packets. Wireshark is not the only program that can extract a lot of useful information from a packet capture, however.

Network Miner

Other programs are capable of extracting information from a packet capture. One of these is Network Miner, by Netresec. You can buy a professional version, and there is also a free version. The professional version adds a number of additional features that may be useful if you end up using this a lot,

but for our purposes here, the free version will work fine. Because Network Miner is written in .NET, it can be run not only on Windows, but any operating system that can run the open source Mono platform. Network Miner will take the pcap file format that Wireshark writes and use that as input, extracting information like files, credentials, messages, keywords, and parameters.

Using Network Miner will save you a lot of time and effort that would be required to search for and extract information manually. Figure 4-16 shows a partial list of all of the files that were identified in the pcap. This includes the usual set of files that you would expect to see like image files, JavaScript files, cascading style sheet (CSS) files, or other documents. The list shown here includes certificate files that are associated with the encryption that is being used. When a client and server negotiate encryption, there is information that is shared. During the packet capture, files associated with that negotiation were captured.

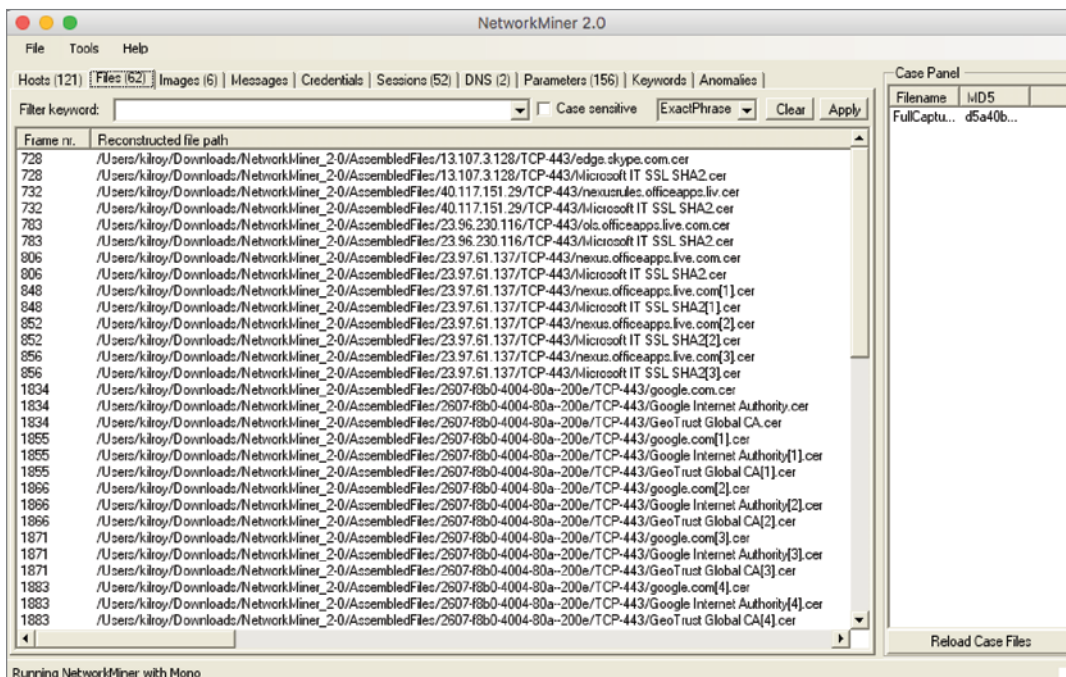


Figure 4-16: Files captured using Network Miner.

Looking at the tabs across the top, you can see other types of assets that have been captured. Wireshark can provide you a list of all of the endpoints and Network Miner will give you a list of hosts. Network Miner provides additional details about the host on the same tab, where in Wireshark you may need to look in more than one place to get this information. Each IP address shown in the hosts tab on Network Miner can be drilled open to get the MAC address, open ports, traffic statistics, and sessions. If available, Network Miner will tell you what the operating system is. Network Miner

will also provide you with the hop distance, meaning how many routers the packets had to traverse to get to the system where the packets were captured.

What you will also notice on the hosts page in Network Miner is that the MAC address associated with the IP address is shown. In most cases, the MAC address will be for the gateway device on the network. The MAC address shown in the hosts in Figure 4-17 is for the router on this network. The only way to have the real MAC address associated with the IP address is for the two systems to be on the same physical network; otherwise, you get the MAC that's associated with the gateway used to get to the destination network. Network Miner, like Wireshark, will also provide the decoding of the MAC address to indicate the vendor of the network interface.

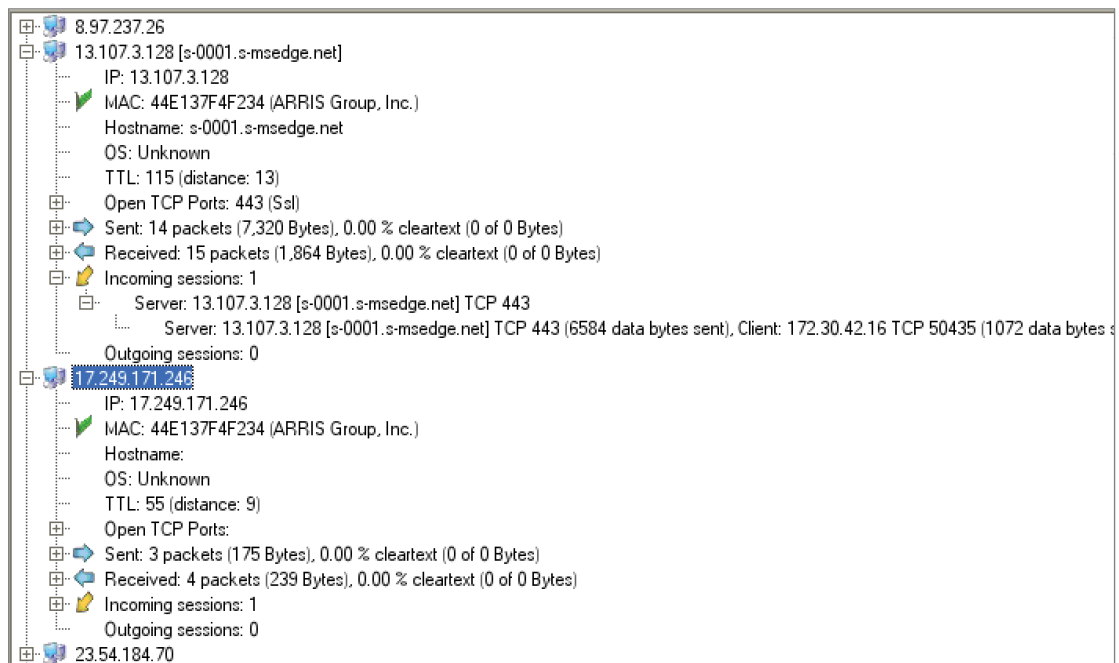


Figure 4-17: Hosts tab in Network Miner.

Summary

While there are certainly tools that will do a lot of the work for you in terms of decoding and analysis, it's still important to understand how to make use of those tools and also know what the information you are looking at means. There is no big report that will be created by these tools to tell you everything you need to know so you still need to know how to perform investigations of network attacks.

Much of what we do in the rest of this book will be based on the information in this chapter because network investigations will often require taking a look at the data being transmitted over the network.

To start with, you have to be able to capture network traffic. There are programs you can use for that. Although Wireshark will provide a graphical interface that will allow you to capture data, there are problems with using Wireshark for that task. To capture network traffic, the application doing the captures requires administrative privileges and once you have enabled those privileges, you run the risk of exposing your system. You may also try to capture data on systems without graphical interfaces, so in either case, `tcpdump`, `windump`, or `tshark` may be good options. They are free and they use the same command-line options, based on how `tcpdump` has functioned for years. Using one of these command-line options, you can store data into `pcaps` that can then be opened later on in a graphical interface for analysis.

One of the challenges of performing packet captures is that networks are currently designed to only send relevant packets to systems. This is done using a network switch that knows the physical (MAC) address of all of the systems on the network. Based on that, only packets that are destined for a host, whether directly or via broadcast, will be captured on end devices. To get more packets requires something like a port span (SPAN port) on a switch, a hub that sends all packets out to all ports, or another technique like ARP poisoning to get traffic to a device that is capturing the network traffic and storing it for future investigation.

Wireshark provides a lot of functionality when it comes to analyzing network traffic. At first glance, just looking at the packets that have been captured, you can see a lot of the decoding that Wireshark does for you. For a start, it provides readable explanations for all of the header information. It will also break out the different header layers, explaining them individually rather than just presenting them in a single incoherent chunk. Wireshark can also perform resolutions of names at the network and transport layers. Saving steps for name resolution as well as vendor lookups from MAC addresses can be very helpful as you are starting to look at a packet capture.

Statistics to get an overall look at the capture are helpful as well. When you start up a capture or load a saved capture, you may see just a collection of thousands and thousands of individual frames. You need to have a way to organize all of those frames into aggregated data to have a place to start. The various statistics views can be very helpful to show you the endpoints and conversations as well as the different protocols that are available in the packet capture. Once you see where the bulk of the information is, you may be able to start looking a little more deeply. You may also use the various statistics as a place to create filters of the data, which will allow you to only show the data you really want to see, hiding extraneous data.

Filtering is an essential skill to have when working with Wireshark. Wireshark allows for both display and capture filters, allowing you to only look at data that you really want to look at. The filters will provide you with functionality to narrow the data based on addresses, ports, streams, or another specific field from headers available from different protocols. Wireshark will automatically create some filters for you or you can just type in your own filters to the display filter box. Once you have the ability to filter, you can start to actually look at the data.

Wireshark will also allow you to extract files from the packet captures. Beyond files, though, a lot of other types of data may be of interest. Credentials may be one type of data that you want to collect. Network Miner will automatically extract credentials, files, session information, and other types of data from the packet captures. You could acquire a packet capture using tcpdump, tshark, or even Wireshark, save it as a pcap file, and then load that pcap into Network Miner. Network Miner will do a lot of automatic extraction for you, presenting the different data types in tabs.

Fortunately, programs like Wireshark, tcpdump, and Network Miner are not limited to a specific operating system. Instead, they run across multiple platforms. Wireshark has versions available for Windows, Linux, and macOS. Tcpdump will run on Linux and macOS with a port named windump available for Windows systems. Network Miner is portable due to the fact that it is written in .NET and there is an open source project called Mono that implements .NET on systems other than Windows. This enables you to become highly skilled as a network investigator, no matter what platform you are using as your primary operating system.

5

Attack Types

In this chapter you will learn about:

- Common attack types
- How to recognize different attacks
- Ways attackers can try to evade detection

The good news is that the tools are easy to come by. He had everything he needed and now, he had some systems he could use. The best part of this was having all of the tools that he needed available to just grab from whatever system he had compromised. They were so simple and yet so effective. One of the best parts was being able to launch one attack from one system in order to disguise an actual exploitation and compromise from somewhere else. One gets lost in the noise of the other. It's easy enough to blind those who may be paying attention by just giving them way too much to look at. This night, he was going to launch one attack that looked to be coming from one place while coming after a web application from somewhere completely different.

The best thing about the web attack is that for the most part it just looked like anyone else visiting the website. The other attack is just to be certain no one is really paying any attention since it's hard to be certain what sort of detection is in place within the victim network. It's best to just throw up a smoke screen. With so many compromised systems at his disposal from months and months of work, it's easy enough to do a little misdirection just to make sure that the important work happens without anyone really seeing it. Plus, why not? After all, it costs him nothing to just kick off a couple of programs on some systems that he has remote access to. Even if those systems are detected and somehow blocked from additional access, they are completely expendable. There are so many more where they came from.

There are a number of different ways to conduct attacks across the network. Some are, by their very nature, noisier and easier to see than others. Often, they have entirely different purposes. We are going to cover the different attack types, their purpose, and how you may be able to go about detecting them. This will include different tools that you may use to gather evidence that the attack occurred as well as tools you can use to dissect and better understand the attack.

When it comes to any incident response activities, application attacks have the potential for leaving lasting marks. They should leave logs behind somewhere. They will also potentially trigger any alerting mechanism that's in place. An application attack is all about trying to send something to a listening application that will provide the attacker some form of access or information he didn't have before. The objective is not always about system compromise or obtaining root-level (administrative) access. This is important to keep in mind. Not all attackers are interested in acquiring another system. Understanding the potential objective is useful in order to identify where the attack is headed.

If you know what to look for, attacks aren't usually that hard to spot, as long as you are looking for them. The problem is that often organizations aren't looking for them or they aren't looking in the right places. On top of that, attackers know that they can be spotted easily enough so they are often looking for ways to obscure their attacks. This is a process called *evasion*, meaning they are looking to evade blocking or detection on the part of the victim, and attackers may obscure their attacks in a number of ways.

Of course, the crown jewel of attacks, or at least the supposed crown jewel, is *system compromise*. This may come through an application attack by exploiting the business logic of an application. It may also come from exploiting a bug in the application program that can lead to access to the system. Not all exploits necessarily lead to system-level access. The exploits are targeted at applications but they aren't usually about taking advantage of the way an application works. Instead, they are usually about bypassing the application code in order to run code injected by the attacker. This may seem like a subtle difference, but the potential impact of the two paths can be significantly different.

One attack that's actually quite easy is the attempt to simply take a service offline. This doesn't require crashing a program since sometimes applications will remain running but unresponsive. No understanding of the application and how it works is needed. It's typically just ignorant brute force used in an attempt to cause an outage at the victim service. Because this is the easiest type of attack, we are going to start with it.

Denial of Service Attacks

We have previously talked a lot about *services*, which are just applications designed to provide something to the user. They run in the background on systems, and the services we are talking about are network-based for the most part. Other ways exist to take services down, but the attacks that are potentially the most dangerous are the ones that attack from the network. Any service that doesn't have a network component can be protected more easily than those that are exposed to the outside world. Strong authentication measures and physical protections by keeping unauthorized users from getting access to the systems and services can help to protect what system owners care about most.

NOTE A service can also simply be the functionality of the operating system and not a specific application. Attacking the network stack itself, which resides in the operating system kernel, can cause multiple applications to no longer be available. Causing the operating system itself to crash can also be an effective denial of service. DoS attacks don't have to be targeted solely at applications.

A denial of service attack is about one thing: making a service unavailable to a user. This could be making it so that customers can't get to a web-based shopping application, which will have an immediate impact to the business that owns the application because users will not be spending money there. Of course, the customers may come back at a later time so the money may not be completely lost, but it is, at a minimum, an annoyance to both the business and the customer. The service does not have to necessarily be taking out the application itself. If you take a system entirely offline by getting it to fail completely, you have done the same thing as knocking the application offline.

Keep in mind that the goal of any denial of service attack is just to deny service to the average user. In some cases, the attacks will look a lot like regular communications. It may simply be the volume that is different. Flooding, which is a volume-based attack, is a fairly common strategy. Its goal is just to utterly consume the available resources. However, there is a very easy solution to combat it—all you need to do is increase the resources you have. This doesn't make the attacks go away, but it does make mounting them more difficult.

Resource consumption attacks are not the only way to go about taking service away. Another strategy is to cause the service to fail or crash. You can do this, potentially, by triggering a bug in the application that causes it to fail or get hung. One way to do this may be to send it input that it doesn't expect. If applications are poorly tested, they may not respond well to unexpected input. As a result, the application may simply not know what to do and it may stop unexpectedly as a result.

SYN Floods

One very easy attack is a SYN flood. This is actually a two-pronged attack in some ways. Remember how the three-way handshake works: the first system sends a TCP message with the SYN flag set, and the receiving system sends back a TCP message with both the SYN and ACK flags set. In the midst of all of that, the sequence numbers are set, but for this attack, that's irrelevant. The attack could set the sequence number to any number because there is simply no intention to ever establish a connection. Once the attacker sends the initial SYN and the victim responds with the SYN/ACK, the connection is in a half-open state. While the system awaits the final ACK in order to establish the connection, it has to maintain that half-open connection.

Systems don't have unlimited resources. In most cases, at least when these attacks began to be popular, a system only had so many slots available for these half-open connections. Once the number of slots was exhausted, no one else could attempt to establish a connection. Think about a mail retrieval system with a collection of cubbyholes where mail goes. As messages come in, each recipient gets a temporary cubbyhole that is held until the message is retrieved. It's a problem if no one ever comes to collect the messages in those cubbyholes because soon there is no space to put any new messages coming in.

In the case of a SYN flood, legitimate messages come in looking exactly like the attacking messages. Everyone, legitimate user or attacker, sends a SYN message to begin a conversation. The legitimate users will respond to the SYN/ACK with an ACK, removing the "message" from the cubbyhole and

freeing that slot back up. The attackers won't respond and won't free up the slot. Eventually, when the cubbyholes, or slots, are entirely full, even legitimate users won't be able to establish connections with the system. This prevents the three-way handshake from completing, so no one gets access to the application because the connection won't get passed up to it until the three-way handshake is complete.

The objective of a SYN flood is to simply fill up the slots that the target system has available for half-open connections. Once the attacker has done this, no one else can start a connection, which means that no one can complete a connection and get access to whatever service is listening on that port. Of course, you might argue that all you need to do is increase the number of slots available to hold more half-open connections, especially since the half-open connections eventually time out and free the slot that was being held. That is one way of addressing the situation, and it does work, but only to a point. Because you can have an unlimited number of half-open connections, for this to be a permanent solution you would ultimately need to have unlimited available resources, which is probably not a realistic expectation.

This brings up the second prong of the SYN flood attack. Its initial target is the operating system, where the slots for the half-open connections are recorded. If the operating system can handle a large number of half-open connections, it takes many more SYN requests to get the application to shut down. At some point, the attacker may send in so many requests that he simply congests the network connection in to the target operating system and application.

Imagine a four-lane road with toll collectors taking money from cars as they zip by. Someone who wants to stop the cars from getting through can get a large number of cars together and send them to the toll booths. Because toll collectors can only work so fast, the heavy traffic begins to cause a backup. Even automatic toll systems will eventually get bogged down, resulting in a backup. Add more booths, more cars are necessary. Eventually, you simply run out of space to add more booths to take on the additional flood of traffic. This is the same thing with a SYN flood. What started out as a resource exhaustion attack on the memory slots available for half-open connections turns into a bandwidth exhaustion attack as more network traffic takes the operating system down with half-open requests.

Fortunately, SYN floods are easy enough to spot if you know what you are looking for. What you see, when you start looking at the network traffic, is a whole lot of SYN messages and the corresponding SYN/ACK messages without any ACK messages coming back. You can see a sample of a SYN flood captured using tcpdump in Figure 5-1. One thing to make note of is that the time remains identical for all of the frames seen in the capture. This is because the SYN messages were sent with a pause of only a microsecond between each frame, so the messages are coming very quickly. The moment a frame hits the wire, it's only a microsecond before the next one is sent. It can be challenging to read this because the received messages are being interleaved with the responses that are being sent.

```

[kilroy@milobloom:~$ sudo tcpdump -i en0 port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 262144 bytes
19:09:44.955415 IP 170.107.94.26.senomix03 > 172.30.42.12.ssh: Flags [S], seq 6
9199824, win 64, length 0
19:09:44.955447 IP 135.123.170.9.senomix04 > 172.30.42.12.ssh: Flags [S], seq 20
78484055, win 64, length 0
19:09:44.955542 IP 172.30.42.12.ssh > 170.107.94.26.senomix03: Flags [S.], seq 3
87686774, ack 369199825, win 65535, options [mss 1460], length 0
19:09:44.955545 IP 172.30.42.12.ssh > 135.123.170.9.senomix04: Flags [S.], seq 3
467874069, ack 2078484056, win 65535, options [mss 1460], length 0
19:09:44.959727 IP 51.240.135.243.senomix05 > 172.30.42.12.ssh: Flags [S], seq 1
19215757, win 64, length 0
19:09:44.959800 IP 172.30.42.12.ssh > 51.240.135.243.senomix05: Flags [S.], seq
860080834, ack 119215758, win 65535, options [mss 1460], length 0
19:09:44.959844 IP 141.163.174.218.senomix06 > 172.30.42.12.ssh: Flags [S], seq
1028021441, win 64, length 0
19:09:44.959891 IP 172.30.42.12.ssh > 141.163.174.218.senomix06: Flags [S.], seq
2756394574, ack 1028021442, win 65535, options [mss 1460], length 0
19:09:44.959980 IP 51.100.51.149.senomix07 > 172.30.42.12.ssh: Flags [S], seq 32
2567809, win 64, length 0
19:09:44.960052 IP 172.30.42.12.ssh > 51.100.51.149.senomix07: Flags [S.], seq 7
34173946, ack 322567810, win 65535, options [mss 1460], length 0
19:09:44.960171 IP 186-44-68-237.dynamic.tstt.net.tt.senomix08 > 172.30.42.12.ss
h: Flags [S], seq 535133466, win 64, length 0
19:09:44.960208 IP 172.30.42.12.ssh > 186-44-68-237.dynamic.tstt.net.tt.senomix0
8: Flags [S.], seq 3176046611, ack 535133467, win 65535, options [mss 1460], len
gth 0
19:09:44.960304 IP 252.7.69.141.8060 > 172.30.42.12.ssh: Flags [S], seq 21351890
00, win 64, length 0
19:09:44.960344 IP 172.30.42.12.ssh > 252.7.69.141.8060: Flags [S.], seq 8490193
04, ack 2135189001, win 65535, options [mss 1460], length 0
19:09:44.960444 IP host-197-162-65-19.static.link.com.eg.8061 > 172.30.42.12.ssh
: Flags [S], seq 627689716, win 64, length 0
19:09:44.960517 IP 172.30.42.12.ssh > host-197-162-65-19.static.link.com.eg.8061
: Flags [S.], seq 1907041502, ack 627689717, win 65535, options [mss 1460], leng
th 0

```

Figure 5-1: tcpdump of a SYN flood.

What you will see, though, in Figure 5-1 as well as any capture of a SYN flood, is the initial SYN coming in and the target system responding with a SYN of its own, as well as an acknowledgment number to indicate that it received the attacker's SYN message. Ideally what happens is that the attacker randomizes the source address, as happened in this case. With the source address randomized, the attacker doesn't get the SYN/ACK back. Some random system on the network receives a SYN/ACK and responds with an RST packet to indicate that the SYN/ACK is erroneous. There are no RST packets shown in this capture session. When using entirely random source addresses, the source address may simply not be reachable so the SYN/ACK may never reach its destination, meaning that the RST packet will never come. The SYN/ACK messages and any RST message end up further clogging the network connection. The more messages the merrier when it comes to resource congestion attacks.

In the case of an attempt to overwhelm the target operating system and prevent it from taking on any new connection attempts, the RST packets are mostly meaningless, though it does take a small

amount of processing to look at them and decide what to do. It may not be much, but it's also not nothing. A large number of RST messages coming back through the network is also an indication that something is going on. Similarly, if you are seeing a large number of SYN/ACK messages on the way in without any corresponding SYN messages on the way out, it's likely that your IP address is being used as a spoofed source of a SYN flood attempt. It doesn't much matter what your address is. When someone is just looking for an IP address that isn't theirs, any address will do. It may be yours.

Listing 5-1 shows a capture of some spoofed messages. The initial message comes in looking like a normal SYN message. The SYN/ACK that goes back will be sent to that IP address.

Listing 5-1: Spoofed Source Messages

```
19:09:44.959980 IP 51.100.51.149.senomix07 > 172.30.42.12.ssh:
Flags [S], seq 322567809, win 64, length 0
19:09:44.960052 IP 172.30.42.12.ssh > 51.100.51.149.senomix07:
Flags [S.], seq 734173946, ack 322567810, win 65535,
options [mss 1460], length 0
```

Spoofing addresses in network transmissions is easy to do. Because there are no protections in IP against spoofing the address, it falls to the other protocols to provide protections. This is part of the function of the three-way handshake. You can't create a complete connection with another system by pretending to be someone else using just TCP. Unfortunately, protecting against spoofing at the TCP layer doesn't protect against these attacks. Spoofed addresses, though, are not the only malicious packet manipulation that can cause problems.

Malformed Packets

Every protocol has a definition. This may be written up in a request for comments (RFC) document or in some other document managed by some group other than the Internet Engineering Task Force (IETF). The protocol definition is used to document expected behaviors of devices supporting the protocol. The moment you have behaviors that are off-book, the program responsible for taking the input and doing something with it may have problems. These issues can take place anywhere along the network stack.

Historically, there have been issues with large packets that require fragmentation, as just one example of malformed packets. In the case where the fragmentation offsets overlap, the target system may not be able to correctly assemble the packet. Let's say that you have packet sizes of 100 bytes but your fragmentation offsets indicate that the second packet should start at the 90th byte. The third packet indicates that it should be placed at the 150th byte, but of course the third packet really starts at the 201st byte. This particular attack was called a Teardrop attack. Some operating systems were incapable of correctly assembling the packets from the received frames, so they simply crashed (some versions of Linux and Windows were susceptible to this attack). Figure 5-2 shows a capture of a Teardrop attack.

This request was initiated manually in order to show how the request works at a very basic level without a lot of other headers that a browser might commonly add. After all of the connection information, you can see the request `GET / HTTP/1.1`. This says we are trying to `GET` the default index page at the top level. The `/` is the top level of the web server and if there is no specific page request, the web server is expected to just reply with the default page in that directory. After the Uniform Resource Identifier (URI) is the protocol and version. In this case, this is `HTTP/1.1`. Version 1.1 of the HTTP protocol definition requires that we specify a host because there may be many virtual hosts at a particular IP address. On the next line, the `Host:` field indicates which specific host the request is being addressed to. A blank line tells the receiving server that the request is complete and it can respond. After the blank line here, you can see the first two header lines that were sent back from the server in question.

By contrast, let's take a look at a malformed HTTP request. These may be very common attacks because a lot of important information is available over web servers that use HTTP to communicate with their clients, whether those are mobile applications or web browsers. Additionally, the means to perform these attacks are readily available. A number of open source programs are capable of generating these malformed requests. The practice of creating malformed requests for testing purposes is called *fuzzing*. Software vendors can use this sort of negative testing to ensure their software can handle unexpected input; but just as easily, attackers can use this testing software to determine whether or not a piece of software is vulnerable to a malformed packet attack.

One of these pieces of software, called Peach, was used to generate the capture shown in Figure 5-3. This shows the ASCII decode of the payload of the packets that were captured. This message should simply be ignored by the web server receiving it but some web servers may try to handle it and get hung up on the fact that it appears to be multiple incomplete requests in a single line. That could potentially cause a crash.

[illegible]

Figure 5-3: Malformed HTTP request.

This particular message is based on something that is legal. It's a long repetition of a legitimate GET request, which is the client attempting to retrieve a page or file. Because the server can't guess what is intended, all it knows is that the message isn't legitimate based on the protocol definition of HTTP.

Malformed requests can be created in a number of ways, and any of them can potentially cause problems. Ultimately, the best way to detect malformed requests is to better understand the protocol you are working with. In many cases, application-layer protocols use text-based commands to

exchange information between the client and the server. An example is shown in Listing 5-3. This is a manual exchange between a Post Office Protocol version 3 (POP3) server and a text-based network client where I was able to just issue commands to the server.

Listing 5-3: POP3 Server Interaction

```
Connected to localhost.
Escape character is '^]'.
+OK Dovecot ready.
user mailuser
+OK
pass P4ssw0rd!
+OK Logged in.
list
+OK 0 messages:
.
get A
-ERR Unknown command: GET
retr A
-ERR Invalid message number: A
```

Not all protocols are readable in this way. You can look at the request and determine what it is. The other way to develop a protocol is to use a binary protocol. Whereas you can read a text-based protocol, a binary protocol encodes information into the bytes. Rather than being able to read the information just by looking at it, a protocol may say that a `GET` request is really an operation code of 42. That 42 can be encoded into a byte containing the value of 42. Where `GET` would normally be three bytes with each letter taking a byte all to itself, we can simplify this and compress it. Binary protocols may take much less transmission space, but they are also harder to troubleshoot because you need special decoders to be able to translate the binary protocol into something that a support person can read.

NOTE Other ways of encoding data may not be binary. As an example, Abstract Syntax Notation One (ASN.1) uses numbers separated by dots as a way of encoding what may be hierarchical data.

In the POP3 example, it's plain what is happening. The very first command issued is `user`. This tells the server that we are passing in the username and that's the parameter that goes with the command. After that, we provide the password using the `pass` command. The server even provides usable and readable responses. It could just give back numeric status codes that the programs on either end would know about. Instead, it sends human-readable text. There are two errors in this session, either of which could be mishandled by a server. The first is trying to use `get`, which is an HTTP request, in order to retrieve a message. Because the server doesn't recognize the command, it rejects it. The second is using a letter in place of the expected numeric value.

The correct command to retrieve a message is `retr` so that is the next command tried. The correct way to use this command is to provide a message number that you want to retrieve. Normally, if there were messages, the `list` command would provide a message number followed by the number of lines in the message as an indication of the message size. As you see in the session, it's not a number but a letter that is passed in. This is illegal as far as the protocol is concerned. However, this particular server handles it correctly by generating an error message.

NOTE While the command `GET` would use 3 bytes if the message were encoded in ASCII, some languages use more characters than can be represented in a single byte. As a result, multibyte character sets are available as well. If you needed to ncode using multiple bytes, you would use Unicode rather than ASCII.

Malformed requests are not always malicious. Poorly written client programs may generate bad requests. In most cases, application software will correctly handle and generate errors on bad messages. However, there are still cases where server software, or even client software, may not be well-developed. Bad requests can cause conditions that are poorly handled. Whether it's intended or not, poorly handled errors can cause software to crash, and once the software has crashed, you have a denial of service condition.

Depending on the transport protocol used, the application may need to handle additional aspects of client control. Whereas TCP handles admission control, there is no such admission control using UDP. This can lead to more code that could go wrong on the server side. Additionally, it opens the door to more processing by the application.

UDP Floods

The purpose of a UDP flood would likely be to just consume all available network bandwidth. Because the operating system is doing none of the admission control that it does with TCP, the application does a lot more work, potentially, using UDP. This can run up the amount of processor resources being used. Send a large amount of spoofed requests into the application and the application will waste CPU cycles trying to determine whether the request means anything or if it's just a lot of garbage. Because of this, a UDP flood has the potential to be a problem for services.

At the operating-system level, we are looking at binary protocols where a TCP header is just a number of bits and bytes that are put together in well-defined way. Binary protocols require significantly less processing. As a general rule, the application knows how long the message coming in is going to be. It also knows where to look for the specific fields to determine what is being requested. This is not true when it comes to text-based protocols. At the application layer, these text-based protocols can require a lot of processing to handle. Requests can be variable length. Parameters are variable. String and character processing for operations like comparisons is more computationally intensive than that required for numeric processing, as would be the case in binary protocols.

If all an attacker is looking to do is flood the pipe, he can just send a large volume of UDP messages. You can see an example of this in Figure 5-4, which is a packet capture of a UDP flood. You may notice in this case that the source port number increments, which is a result of the tool that is being used, but the source IP address remains constant. With a UDP attack, it would be common for the application to not respond to improper requests so there would be no risk of the sending party getting responses, which would be potentially as problematic to the sender as the receiver in terms of bandwidth consumed. You will see from this example that all of the messages are one way. There are also thousands of UDP messages being sent per second.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.30.42.16	192.168.1.1	UDP	442	2407 → 10000 Len=400
2	0.000063	172.30.42.16	192.168.1.1	UDP	442	2408 → 10000 Len=400
3	0.000080	172.30.42.16	192.168.1.1	UDP	442	2409 → 10000 Len=400
4	0.000093	172.30.42.16	192.168.1.1	UDP	442	2410 → 10000 Len=400
5	0.000105	172.30.42.16	192.168.1.1	UDP	442	2411 → 10000 Len=400
6	0.000118	172.30.42.16	192.168.1.1	UDP	442	2412 → 10000 Len=400
7	0.000130	172.30.42.16	192.168.1.1	UDP	442	2413 → 10000 Len=400
8	0.000142	172.30.42.16	192.168.1.1	UDP	442	2414 → 10000 Len=400
9	0.000154	172.30.42.16	192.168.1.1	UDP	442	2415 → 10000 Len=400
10	0.000167	172.30.42.16	192.168.1.1	UDP	442	2416 → 10000 Len=400

▶ Frame 1: 442 bytes on wire (3536 bits), 442 bytes captured (3536 bits) on interface 0 (outbound) ▶ Ethernet II, Src: Apple_b7:2c:89 (f4:5c:89:b7:2c:89), Dst: ArrisGro_f4:f2:34 (44:e1:37:f4:f2:34) ▶ Internet Protocol Version 4, Src: 172.30.42.16 (172.30.42.16), Dst: 192.168.1.1 (192.168.1.1) ▶ User Datagram Protocol, Src Port: 2407 (2407), Dst Port: 10000 (10000) ▼ Data (400 bytes) Data: 5858585858585858585858585858585858585858585858... [Length: 400]

Figure 5-4: UDP flood packet capture.

The payload was generated by the attack tool. Each packet includes 400 bytes of payload. Because there is no expectation that the application will respond, it doesn't matter what the payload is. However, it appears that the payload isn't random either. Every packet has the same payload. This makes it easier to spot. A random payload with random source addresses may make it harder to identify other than simply looking for large packets to the same destination address and port. A lot of messages that have nothing to do with the application listening on the destination port will stand out, particularly if they are coming in at a very high rate of speed.

Bogus messages are not the only ones that can be problematic. Some UDP protocols use a type of encoding called Abstract Syntax Notation One (ASN.1). This type of encoding is represented as dotted numeric values, meaning that you have a series of numeric values separated by dots. You can see one of these values in Figure 5-5. This is an SNMP request. SNMP, as well as other protocols, makes use of ASN.1. Parsing ASN.1 requests can also be computationally costly. Not as costly, perhaps, as text-based transmissions, but still costly. As a result, rather than just sending bogus requests to a UDP port, if you were to find a port that is open that you wanted to target and it used a protocol that you knew, you could send legitimate requests to the port and let the requests eat up CPU cycles in addition to the I/O.

No.	Time	Source	Destination	Protocol	Length	Info
113	6.441897	172.30.42.16	172.30.42.1	SNMP	82	get-next-request 1.3.6.1.2.1
128	7.446874	172.30.42.16	172.30.42.1	SNMP	82	get-next-request 1.3.6.1.2.1
146	8.448885	172.30.42.16	172.30.42.1	SNMP	82	get-next-request 1.3.6.1.2.1
176	9.450949	172.30.42.16	172.30.42.1	SNMP	82	get-next-request 1.3.6.1.2.1
186	10.455929	172.30.42.16	172.30.42.1	SNMP	82	get-next-request 1.3.6.1.2.1
194	11.460539	172.30.42.16	172.30.42.1	SNMP	82	get-next-request 1.3.6.1.2.1

Figure 5-5: SNMP request.

From the perspective of the attacker, UDP requests are easier. It's considerably easier to spoof an address using UDP as the transport protocol because there is no verification of the source address at the operating system level as there is with TCP. If an attacker is going to be challenged using UDP as the transport protocol, it would have to be done by the application, and that just means more resources used. UDP also makes other types of spoofing attacks easier. We'll look at an example of another way UDP can be used in the next section.

Amplification Attacks

If you are trying to attack your target, it's easiest if you can make use of others to help you out in that effort. Let someone else do the work for you. This was the case many years ago with what was called a *Smurf attack*. A Smurf attack relied on spoofing a source IP address and sending an ICMP echo request, commonly known as a ping message, to the broadcast address of a network block. If the network was not correctly configured, every host on that network that received the echo request would send an echo response back to the spoofed source. This could potentially be thousands of responses to a single request and all of those responses would be returned to the spoofed source address, belonging to the attacker's target. Of course, protecting against this just means ensuring that responses don't go back to an address not on your network if the destination is to the broadcast address. In Figure 5-6, you can see the list of the top-ten Smurf amplifiers known on the Internet in January 2010. This was the furthest back I was able to go using the Wayback Machine, and since the cleanup efforts had been long underway at that point, the biggest network only shows 86 duplicate messages.

Current top ten smurf amplifiers (updated every 5 minutes) (last update: 2010-01-14 05:16:02 CET)					
Network	#Dups	#Incidents	Registered at	Home AS	
64.80.254.0/23	86	0	2009-03-13 12:08	not-analyzed	
212.1.130.0/24	38	0	1999-02-20 09:41	AS9105	
194.215.75.0/24	35	0	2000-09-18 21:11	not-analyzed	
137.229.212.0/24	33	0	2008-08-23 18:36	not-analyzed	
168.188.134.0/24	32	0	2009-04-19 20:44	not-analyzed	
64.141.51.0/24	28	0	2008-07-21 22:42	not-analyzed	
168.188.10.0/24	27	0	2009-04-16 07:01	not-analyzed	
204.158.83.0/24	27	0	1999-02-20 10:09	AS3354	
209.241.162.0/24	27	0	1999-02-20 08:51	AS701	
150.229.208.0/24	23	0	2006-05-26 20:21	not-analyzed	
2452459 networks have been probed with the SAR					
120 of them are currently broken					
193767 have been fixed after being listed here					

Figure 5-6: Smurf amplifier registry.

The more duplicate messages, the larger the amplification. When sending an ICMP echo request, a payload can be set and the payload will be included in the response. This is a good way to dramatically increase the amount of bandwidth being used. If you can get several hundred amplifiers all responding simultaneously with 1200 bytes each, that will consume a lot of bandwidth. When these attacks were most common, it was possible to get tens of thousands of hosts all responding to messages directed to the broadcast address. You can see an example of the multiple replies in Listing 5-4. In most cases, this particular attack won't work very well. Despite having a dozen or so hosts on this network, only two hosts are responding to ICMP echo requests to the broadcast.

Listing 5-4: Pinging a broadcast address

```
kilroy@oliver:~$ ping 172.30.42.255
PING 172.30.42.255 (172.30.42.255): 56 data bytes
64 bytes from 172.30.42.16: icmp_seq=0 ttl=64 time=0.074 ms
64 bytes from 172.30.42.23: icmp_seq=0 ttl=64 time=4.135 ms
64 bytes from 172.30.42.16: icmp_seq=1 ttl=64 time=0.110 ms
64 bytes from 172.30.42.23: icmp_seq=1 ttl=64 time=3.400 ms
```

Though this particular attack won't work very well, there are other amplifier attacks. Since low-cost attacks are economically advantageous, new amplifier attacks pop up periodically because there are adversaries who are looking for them. Another amplifier attack that became common in 2013 used the domain name system (DNS). Like the Smurf attack, this was also a reflected attack, meaning that the attacker spoofs a source so the attack is actually bounced off other hosts in order to create the effect. The attack is actually quite simple. Because DNS requests are commonly done over UDP, this very simple spoofing attack just requires an open DNS resolver somewhere. Fortunately, a number of those are available, including the Google server at 8.8.8.8.

Most DNS requests won't generate a lot of traffic in response to a request. As a result, it's important to select a type of request that will generate the largest response possible. Using the ANY request, you will get all available information back. However, even that may not be an enormous response. In Figure 5-7, you can see the result from a DNS ANY request for the DNS zone microsoft.com. The response is only 775 bytes. Other zones may yield more information.

Of course, considering that the query itself was only 31 bytes, there is a significant amplification achieved. A single host may not be able to really generate much in the way of attack traffic. The better approach is to get a lot of hosts involved to issue a number of DNS requests with the source address always the target. This way, you are guaranteed to generate a large amount of network traffic. The more systems you can have participating in the attack, the better chance you will have of knocking your target offline.

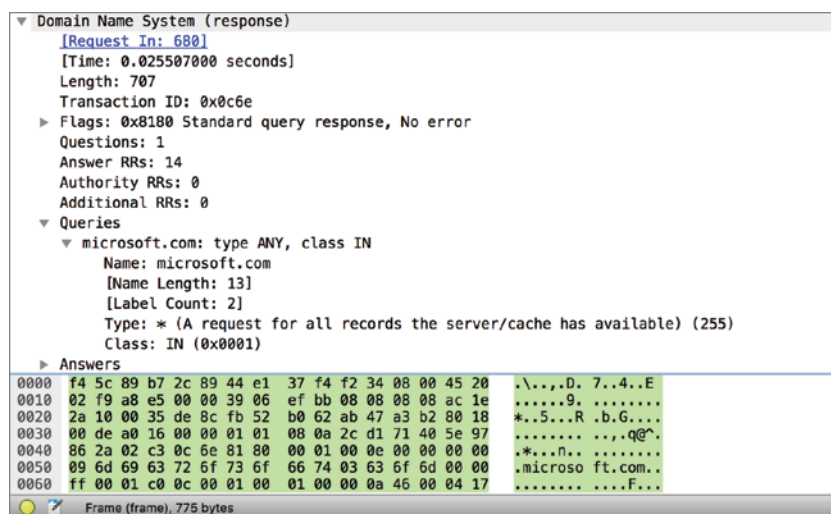


Figure 5-7: DNS ANY request result.

Distributed Attacks

Not all denial of service attacks are distributed, but with large quantities of bandwidth being the normal state for businesses and even many end users, it's quite a bit harder to generate enough attack traffic as a solo practitioner than it used to be. As a result, we have distributed denial of service attacks (DDoS). A distributed denial of service attack consists of multiple attackers distributed around the Internet. The primary reason to use attackers from a number of locations is to ensure they can generate a lot of outbound bandwidth in order to cause problems with the victim. Multiple attackers at a single site are going to be constrained by the amount of bandwidth available at that site. If the attacker can't generate enough outbound traffic, the victim will be able to mitigate the attack and it will fail because the services will remain up and functional.

A distributed attack needs to be coordinated. A number of attackers all generating attacks at vastly different times will not be enough to cause an outage at the victim. There are two ways to perform this coordination. The first is to use a common communication stream like an Internet Relay Chat (IRC) server, which allows all of the participants to be in one virtual location at the same time and receive the start message so everyone can go all at once. More common, though, is for the attack to be perpetrated by a number of systems that are connected through malicious software on each of them. This client software is generally installed without the user's knowledge and the system owner has no idea his system is being used to attack other systems.

When a computer system has been infected with this software, it gets attached to a botnet. The botnet is a collection of bots, which are computer systems owned by someone else but under

the control of the botnet owner. Botnets are complex systems that involve not just the bots but also an entire command and control infrastructure, also composed of infected systems.

NOTE A *bot* is a semi-autonomous entity that can perform work. This is probably a shortening of the word robot but bots have been around in different contexts, including outside the realm of computing, for many years. When you have a network of bots, you have a *botnet*.

The idea of a botnet began in the late 1990s with a piece of software written by a German computer security specialist who went by the name Mixter. Mixter wrote the Tribe Flood Network (TFN) as a denial of service attack tool. TFN became TFN2K. In early 2000, the world witnessed the very first large and public distributed denial of service attack. It was accomplished using the program Stacheldraht, which is the German word for barbed wire. Stacheldraht took elements of TFN2K along with another program named Trin00 and added in encryption. The Stacheldraht attacks were launched by a Montreal teen who went by the name MafiaBoy and were targeted at eBay, Yahoo, and other large and well-known companies on the Internet.

Since 2000, we have seen a number of other enormous botnet attacks against businesses. The largest attack known at this time was an attack against the British Broadcasting Corporation (BBC) and it measured more than 600 gigabits per second. What isn't clear is how long that attack was sustained. The attack of the Mirai botnet on the site belonging to Brian Krebs was also reported to be above 600 gigabits per second. Other attacks that took place over the course of several months in 2012–2013 were targeted at a number of US banks by hacktivists in the Middle East. These attacks were often several hundred megabits per second and were sustained attacks lasting weeks at a time.

It's easy enough to detect that these attacks are happening. What is considerably harder is knowing where they are coming from. Machines involved in the attacks are generally spread around the world, but they don't use their own IP addresses so you can't use IP identification mechanisms to determine who is involved. From the perspective of an enterprise, there is nothing that can be done to mitigate the attack because the goal is generally to overload the network connection from the Internet service provider (ISP) to the business. The business only has control over what happens on its end of the connection, and once the attack gets to any device that the business has any control over, the network connection is already overloaded. Firewall rules will stop the attack from getting into the business, but that won't stop the traffic from filling the connection to the ISP.

Even the ISP can likely only protect the business by attempting to set filters at its end of the connection. Because the attack traffic will generally look like real traffic—a SYN message to a web server, for instance—it can be difficult to distinguish a real message from an attack message. Commonly, legitimate messages will get caught in the crossfire and dropped, propagating the attack in the process of trying to mitigate it.

The best way to really protect against an attack like this is at the source. However, it can be challenging to determine where the attack originates. There can be a very large number of origin points

on large ISP networks. With millions of botnet clients around the world participating, it is difficult to block the messages as they enter the ISP's network. Not long after the Stacheldraht attacks, a pair of engineers at UUNet, an ISP at the time, developed a means of identifying the sources of attacks.

Backscatter

When spoofed messages hit a target, the target will respond to the IP address in the IP header, even though it's bogus. Because these IP addresses are typically random, the responses to the bogus messages will be spread all over the Internet, including to a number of IP addresses that are either not in use or are unreachable. The attack tools don't bother to see if the IP address is actually alive before using it as part of attacks, so you will see a large number of SYN/ACK messages that are sent to hosts that will never receive them because they are either not up, not existent, or behind firewalls and unreachable. These response messages are called *backscatter* and they can be plotted.

Backscatter can be observed using a *network telescope*, which is sometimes called a *black hole*. This is a system that is designed to observe network traffic. In particular, what it is looking for is traffic to unused IP addresses. Many of these unused IP addresses are from unallocated or reserved space. If traffic is destined to unallocated or reserved space, it is considered suspicious by default. With nothing on the other end to receive the message, there is no reason to have sent it to begin with. Much of this traffic likely comes from spoofed attacks, so it is useful to set up systems to observe this type of traffic.

Backscatter is the sort of network activity that is best seen in large networks that will see much of this activity and also be able to control where it goes. If, however, the local network starts to see a lot of messages either to or from unallocated network space, it is likely that there is an attack happening. Table 5-1 shows addresses that, if seen, may indicate something malicious is happening in your network.

Table 5-1: Martian Addresses

Network Range	Use
0.0.0.0-0.255.255.255	Unused network
10.0.0.0-10.255.255.255	RFC1918 Private addressing
100.64.0.0-100.127.255.255	Carrier grade Network Address Translation address
127.0.0.0-127.255.255.255	Loopback addressing
169.254.0.0-169.254.255.255	Zero configuration addressing
172.16.0.0-172.31.255.255	RFC1918 Private addressing
192.0.0.0-192.0.0.255	Internet Engineering Task Force Protocol assignments
192.0.2.0-192.0.2.255	Used for testing

Network Range	Use
192.168.0.0-192.168.255.255	RFC1918 Private addressing
198.18.0.0-198.19.255.255	Benchmark testing
198.51.100.0-198.51.100.255	Testing
203.0.113.0-203.0.113.255	Testing
224.0.0.0-239.255.255.255	Multicast
240.0.0.0-255.255.255.255	Reserved

This is a list of network addresses that messages should never have as a source address coming in from the Internet. Anything in this address range is considered to be a Martian because anything that comes from these blocks is coming from Mars, meaning from somewhere that a packet clearly can't come from. The multicast addresses may be a destination address but shouldn't be source addresses. If you are on a network that uses RFC1918 private addressing, you will of course see the IP addresses from that range on your local network. Because RFC1918 traffic is, by convention, non-routable on the Internet, you should never see those messages coming in from your Internet connection. Even if a message from the inside of your network made it to the outside with one of the private addresses in place as a source, there would be no way for the recipient to return the message so you'd never see it coming back in. Messages appearing to come from RFC1918 space would typically be attack traffic.

NOTE Unicast messages are those where the destination is a single host. Multicast traffic is where there are multiple destinations for the message but the recipients are still specific and they would have asked to be recipients. Broadcast traffic is being sent to everyone, regardless of whether they asked for it or not.

While none of these addresses should ever appear as source addresses coming into your network, this is not the entire collection of messages that would be unlikely to be seen. There are still collections of unallocated addresses scattered around the Internet and under the control of a large number of organizations. While all of the IP addresses have been allocated by the regional Internet registries, there are still companies that have unused blocks of IP addresses. Because of this, it's hard to generate a complete list of all of the addresses from which you would never see a message, but the ones just shown are a good start.

In order to capture these messages, the routers on the network could be configured to send all traffic destined for the Martian addresses off to a specific host for capture. Larger or growing volumes of traffic to your monitoring device may provide you a heads-up that a denial of service attack may be happening. In some cases, firewalls and routers are configured to simply drop or discard these messages. *Reverse path verification* is the term for routers ensuring that the packet should be coming

in on a particular interface. Although this keeps bad traffic from going any further along a network path, discarding the messages provides no way to analyze the events.

Cisco developed a feature in its routers named NetFlow. Using NetFlow, which has now been implemented in non-Cisco devices as well, you can get a view of what the traffic flows on your network look like. This may be a way to better observe the Martian traffic because the metadata about the messages are aggregated. You will be able to see the interface the traffic has arrived on, the source and destination IP addresses, the source and destination ports as well as the IP protocol (for example, TCP, ICMP, UDP, and so on), and IP type of service. You will also get the time and date that the flow started and the duration of the communication.

Vulnerability Exploits

Denial of service attacks are fairly easy and therefore common, but they are primarily annoying. They can cause outages, which can lead to a loss of business during the time that the attacks are underway. However, far worse than that is a system compromise. While it can be common for a system compromise to happen by sending users malicious software, there are still other attacks that can lead to system compromise. With network services allowing attackers to send data into an application, those attackers can send specially crafted messages that can allow them to gain access to the system, often without any type of authentication. As an example, Figure 5-8 shows an attack against a distributed program compilation service called distcc.

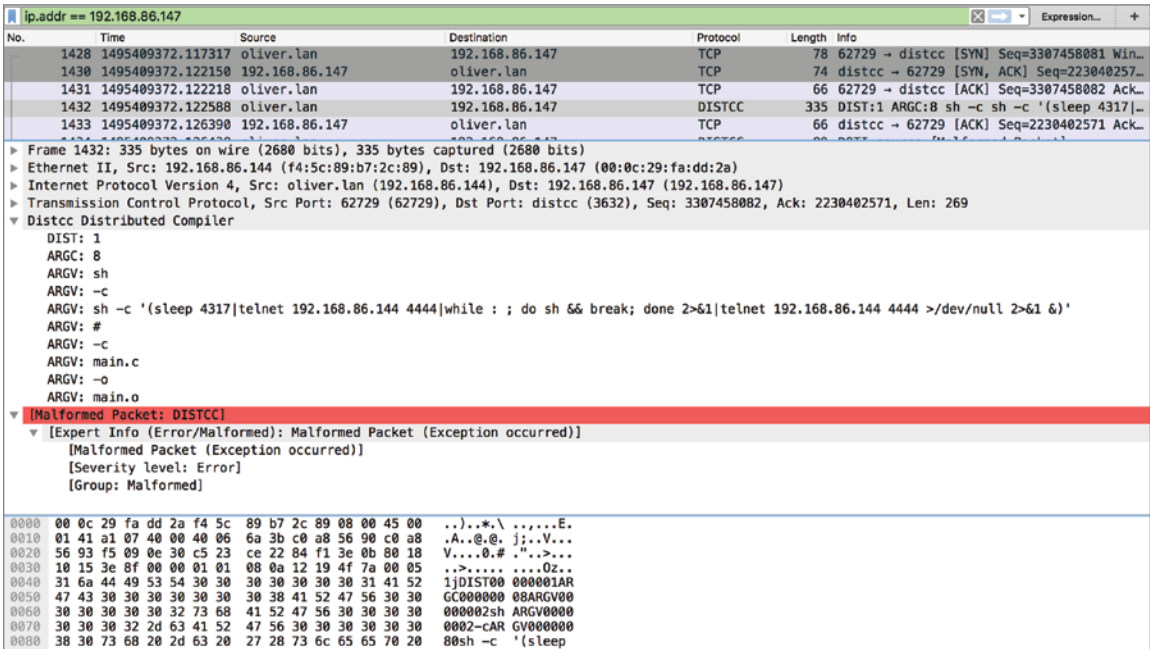
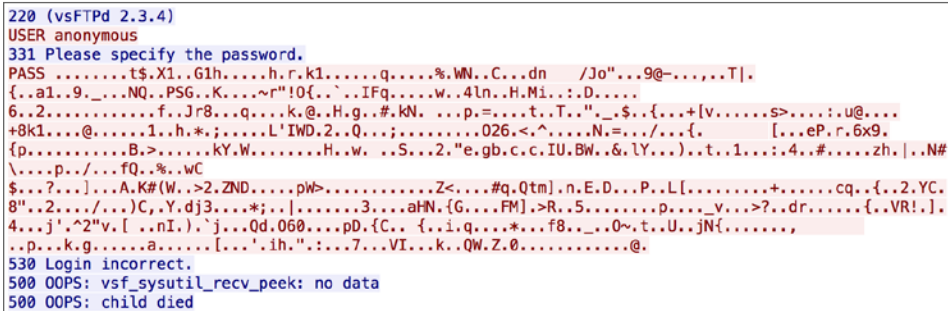


Figure 5-8: distcc attack.

This is a packet capture showing the attack, and Wireshark has flagged that there is a problem. If you look at the ASCII decode in the bottom, you can see a very short program that was sent to the distcc server. This particular distcc server has a vulnerability where the attacker can send command-line interface commands and have them executed by the server process. As a result, this particular payload (the portion of the attack that is actually being run on the server) contains a command to the server to initiate a network connection back to the attacker using the telnet client. The attacker has a process running waiting for the connection back from the victim server. Once the reverse connection is made, the attacker has a direct connection to target server. Using that connection, the attacker can issue shell commands, just as though he had a command-line session open physically on the target system.

Although this is an attack that is easy to see because it is in plaintext, not all attacks are like that. Perhaps more commonly, attacks that exploit vulnerabilities make use of the language that machines use. Instead of text-based commands, your computer uses binary operation codes that the processor can understand. One attack technique is a buffer overflow. The program takes in data into small memory chunks called buffers. The buffer is a fixed size and if the attacker can push more data into the buffer than it is intended to hold, the successive memory locations beyond the fixed-size buffer will be overwritten. This allows the attacker to inject code that the processor will execute. This code will be binary, which means it may not be values that can be represented as printable characters. You can see an example of an overflow attack in Figure 5-9.



```

220 (vsFTPD 2.3.4)
USER anonymous
331 Please specify the password.
PASS .....t$.X1..G1h.....h.r.k1.....q.....%WN..C...dn  /Jo"...9@-.....T|.
{..a1..9...NQ..PSG..K.....~"!!O{...`..IFq.....w..4ln..H.Mi...D.....
6..2.....f..Jr8...q...k.@..H.g.#..kN.  ...p.=...t..T..."_.$..{...+ [v.....s>.....u@....
+8k1...@.....1..h.*;.....L' IWD.2..Q...;.....026.<.^.....N=.../...{.  [...eP.r.6x9.
{p.....B>.....kY.W.....H..w.  ..S...2.."e.gb.c.c.IU.BW..&.lY...)..t..1...:4..#.....zh..|.N#
\.....p./...fQ..%.wC
$....?..]...A.K#(W...>2.ZND....pW>.....Z<...#q.Qtm).n.E.D...P..L[.....+.....cq..{..2.YC.
8"...2.../...)C..Y.dj3.....*;;|.....3...aHN.{G...FM].>R..5.....p....._v...>?.dr.....{..VR!..}.
4...j'^2"v.[  ..nI..).`j...Qd.060....pD.{C.  {..i.q...*.f8..._0~.t..U..jN{.....
..p..k.g.....a.....[...'.ih"...7...VI...k..QW.Z.0.....@.
530 Login incorrect.
500 OOPS: vsf_sysutil_rcv_peek: no data
500 OOPS: child died

```

Figure 5-9: Buffer overflow attack.

This particular attack is an overflow attack against the password input to a File Transfer Protocol (FTP) server. PASS is the command to the FTP server indicating that the subsequent parameter is the password for the previously submitted username. This attack overflows the buffer that has been set aside for the password field. What you can see in this screen capture is the binary data represented in ASCII characters. Most of them are not printable, and those characters are represented as just dots rather than a recognizable character. There are some recognizable characters there, but that's just because occasionally the binary data necessary to make the attack work aligns with an ASCII value for a printable character.

This particular attack is easier to see because FTP is a text-based protocol. The information that is being transmitted should be in plaintext and easily readable by people. In this case, we have a mixture of plaintext and binary data. This suggests that there is probably something that shouldn't be happening going on. Again, though, this requires some familiarity with the protocols that are being used. In order to see what shouldn't be happening, it's important to know what should be happening.

Other attacks are possible against vulnerable services. Buffer overflows continue to be possible, despite them being well-known attack vectors for about thirty years and well-documented as possible ways to break into systems for about the last twenty.

Another attack style that you may run across is called a *format string attack*. When a computer program is written in the C programming language, the programmer will use a collection of input/output functions that make use of something called a *format string*. The format string allows for a variable number of data parameters. As an example, the format string “%d %d %s %f” indicates that two decimal values, followed by a string and a floating-point value, are how the input or output should be formatted. The data parameters have a space between them. Another formatting value is a %x. If the user has control over input that will later be sent back to the user in some manner, there are sometimes ways that the format string can be entered. In some cases, programmers will make use of the `printf` output function without specifying a format string. In that case, if the attacker has specified the format string, the `printf` function will then have to find the actual values elsewhere. This type of attack targets a family of functions—`fprintf`, `printf`, and `sprintf`.

In place of the expected input provided by the user, since what the user presented is being used as a format string, the `printf` function makes use of the next values in memory. Using this technique, an attacker can extract the contents of memory using a specially crafted set of input. Fortunately, in this case, the attack is in plaintext since the format string is provided just as shown in the previous paragraph. If you see a collection of values going to a service where a letter follows the percent symbol, you may be looking at a format string. In most cases, the format string is a way to obtain information that may be used to attack the service. However, the %n format string value may allow a user to place a value into memory by specifying an address to write the data to.

These attacks can be used by unauthenticated adversaries from anywhere that can reach the service. However, this doesn't mean that the attackers are only on the outside. Services may still be available over the network but may only be available on the internal network, either because of firewalls blocking external access or because they are available only over private networks. The business can protect services from external attacks in a number of ways.

Insider Threats

It has long been considered a truism in the information security space that the majority of losses come by way of insiders. This may be changing in light of the attacks by criminal organizations looking to steal information being kept by the business. Additionally, there are some attackers from other countries looking to get intellectual property. No matter what the case is, though, not all attacks

originate from outside of the organization. When it comes to insider threats and attacks, the same attack types mentioned previously are still possible, but there are others as well.

It's not uncommon for users in an organization to continue to gather permissions for various resources on the network without ever dropping any of the permissions as they change jobs and need access to additional file shares and applications. Although new employees commonly go through a background check, circumstances change and some employees may make use of information they shouldn't have access to. They could sell the information or they could just corrupt it because they are dissatisfied with the company.

One of the challenges with data theft is that it can be difficult to determine that it's happened. It's not like the files just disappear. Someone can steal sensitive information and leave the information in place at the same time. A digital copy is just as good as the original because they are identical. This data loss is a serious problem for businesses.

Even if data is well-protected inside the organization and only the right users have access to it, it may not only be those users who get the data. When a user's system has been infiltrated by malware, the attacker who has control over the malware can also get access to the information. The malware may be running with the same permissions as the user of the system that has been infected. If the malware is a remote access trojan (RAT), the malware owner will have access to the system and any file shares it can get to that the user has permissions to. Once the attacker has a foothold on the network, he can start pulling any available data out and sending it to another system off the network where the attacker can store, sift, organize, and later sell or make use of it.

NOTE In its legitimate guise, a RAT is known as a Remote Administration Tool.

Because these sorts of attacks can appear to be legitimate, it is much harder to identify them. Currently, data loss prevention (DLP) software is available. External attackers will try to forward data out to external systems and that can be observed. Looking for specific keywords in data transmissions like "confidential" or "sensitive," or even patterns like those common for credit cards or Social Security numbers can help to detect an exfiltration, which is the process of sending data out of an organization.

In the case of an actual attack, like a denial of service or a vulnerability exploitation, businesses may not be looking inside their network. Firewalls and intrusion detection systems are common on the outside of the organization. Depending on the business and the architecture it has in place, the business may not be paying attention to what's happening on its local, internal network. In the case of a vulnerability exploit, this is going to be coming from an internal IP address. This makes it easy to track, assuming that facilities are in place to detect it even happening.

Denial of service attacks from inside the network are a little more problematic simply because they aren't passing through an external network connection. Instead, they are coming through the switches inside the network, and those physical links generally have more bandwidth capacity than the external connections. While spoofing is common in denial of service attacks, it may be less

effective in an internal attack. If the attack is against a system on the same network, meaning it is only passing through switches and not through a router, the frame will have the source MAC address on it.

NOTE Any network that can be reached strictly through layer-2 addressing is sometimes called a *broadcast domain*. It indicates that all of the hosts on the network can be reached with a layer-2 broadcast message. In networks that don't use switches, this is also called a *collision domain* (although because switches have generally removed the potential for collisions, the term “collision domain” is outmoded).

Because the MAC address is bound to the physical interface and isn't generally changed, it is possible to track the message back to its source by just identifying the owner of the MAC address. While there may not be one database anywhere that maps MAC address to a system name or location, it is possible to determine which switch port the MAC address is connected to. This will generally allow you to identify the physical location of the system that was involved in the attack. As soon as a frame hits a router, though, the layer-2 header with the MAC address is stripped off and the packet is forwarded out through another interface. This makes it slightly harder, though not impossible, to track the attack back through the network to the source.

Evasion

Attackers will generally want to evade detection and prevention. An attack won't be very successful if it is easily detected. A number of evasive techniques can make it more difficult to locate an attack or even identify it as an attack after the fact. As an example, let's look again at the distcc attack. This time, some evasive methods have been used. The first, which you can see in Figure 5-10, is fragmenting the packet into small frames. Where before you could see the entire attack in the same packet, the same is not true here. You can't see the shell commands in this particular frame. Just looking at this frame, it's hard to determine that anything bad is happening. Without the Wireshark warning that it is a malformed packet, you may not look any further.

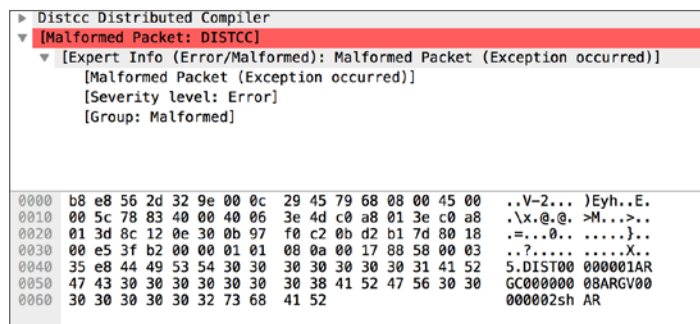


Figure 5-10: Attack evasion using fragmentation.

The next frame associated with this fragmented packet would have pieces of the shell commands that were sent with this attack. This particular fragmentation attack used 40-byte fragments, though an attacker could break the packet down to even smaller fragments, which could make it harder to detect. When it comes to detection efforts that are going on in real time, the system would have to reassemble the entire packet before doing any detection. Since waiting for all of the fragments is time consuming and it impacts the latency associated with the exchange, not all detection engines will spend the effort doing the recompile. This is especially true if the detection is happening in-line, meaning that it is between the user and the server he is communicating with. An attacker can take advantage of the unwillingness to impact the user experience by waiting for all of the fragments.

In addition to fragmentation, an attacker may reorder the fragments and also delay the different fragments. An attacker doesn't really care how long the entire attack takes but when it comes to protection, delays are not generally tolerated. Any impact to the user experience when business relies on speed and efficiency of communication is a no-go. As a result, these types of evasion attacks may be successful. All it takes is a piece of software that is capable of chopping up packets prior to sending them off to the network. In the case shown in Figure 5-10, the evasion was done by Metasploit, which is the software used to generate the attack. Evasion options are provided by Metasploit. If an attack isn't being performed from a tool like Metasploit that can handle the evasion natively, other tools like fragroute can perform the fragmentation after the attack tool has sent the message.

Other types of evasion are based on encoding the data in a way that is more difficult for some tools and humans to read correctly. As an example, Base64 is a type of encoding where any type of data can be converted to human-readable text strings. This is a way of taking non-printable data and making it printable so it can be manipulated. Binary data, depending on how it is represented, can't be copied and pasted, for example. However, if the binary data is converted to Base64, it can be copied and pasted from one place to another and then it can be reconverted back to the binary data that it originated as. Entire programs can be converted to Base64 and transmitted. To show you, Listing 5-5 is a text string that has been converted to Base64.

Listing 5-5: Base64 Encoded Data.

```
VGhpcyBpcyBiYXN1NjQgZW5jb2R1ZA==
```

The original string read "This is base64 encoded" though it didn't have the quotes. A number of other types of encoding exist. URL encoding takes non-alphanumeric characters and converts them to the hexadecimal value associated with the ASCII number for the character. To identify it as a URL-encoded character, it is preceded by a percent sign. You will often see %20, for instance, in URL-encoded strings. The hexadecimal value 20 is 32 in decimal and 32 is a space character in ASCII so if you see %20, what you are looking at is just a space. You may also run across HTML encoding, which takes some characters and converts them to their HTML notation. As an example, < becomes < because it is the less-than symbol.

Another type of encoding that isn't easy to read but is easy to identify and decode is decimal encoding. The string shown in Listing 5-6 reads "This is decimal encoding but it has been decimal encoded."

Listing 5-6: Decimal-encoded Value

```
&#84;&#104;&#105;&#115;&#32;&#105;&#115;&#32;&#100;&#101;&#99;&#105;&#109;&#97;&#108;&#32;&#101;&#110;&#99;&#111;&#100;&#105;&#110;&#103;
```

In decimal encoding, each character is preceded by an ampersand (&) and a pound sign (#) to indicate that it is a character that has been decimal-encoded. The numeric value is the decimal for the ASCII character that is being represented. The first character in Listing 5-4 is a T, which is decimal 84 on the ASCII table. When it comes to these encoded values, there are a number of ways to perform the decoding. Some browsers will allow extensions or plugins that may provide decoding capabilities either directly or through a toolbar. A number of web pages can also perform encoding and decoding—a quick Google search will turn up several pages. If you see anything in network traffic that appears to be encoded, you can extract the ASCII of the transmission and attempt to do the decoding. Each type of encoding looks different, but once you have seen enough of them, you will be able to differentiate them easily.

Application Attacks

We have talked about different application-layer attacks like fuzzing, vulnerabilities, and malformed packets, but other application attacks exist. Client vulnerabilities are an issue as well. These may be triggered by sending malicious files in the case of something like a vulnerability in a Portable Document Format (PDF) reader. Web browsers are also common applications that can be attacked. In many cases, the attacker will try to get the browser to open a malicious Java or Flash applet. It may also be possible to take advantage of a buffer overflow in either the browser itself or one of the add-ons, like the handler for PDFs or Flash.

Because these attacks are going after the browser, you will see them by looking through the HTTP messages going back and forth between the browser and the malicious server. In Figure 5-11, you can see an attack between a rogue server and a client. This particular attack was designed to go after an issue with the Firefox web browser. You can see the initial HTTP request from the browser going to the web server. Though it's just a standard GET request, the URL looks suspicious. It appears like a random string. On top of that, if you look at the `Host:` line, you can see that the port the request went to is port 8080. While it's not out of the realm of possibility since some web servers, and especially proxy servers, will listen on port 8080, it's also not common. The default port for web servers is port 80.

```

GET /R4muF1FBX2nbGgE HTTP/1.1
Host: 192.168.1.62:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.116 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: __ua=MtyKGWLaAGtMPVvGRBh;
Connection: Keep-Alive
Server: Apache
Content-Length: 39958

<script>
    // Base64 implementation stolen from http://www.webtoolkit.info/javascript-base64.html
    // variable names changed to make obfuscation easier
    var Base64 = {
        // private property
        _keyStr:"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/",

        // private method
        _utf8_encode : function ( input ){
            input = input.replace(/\r\n/g,"\\n");
            var utftext = "";
            var input_idx;

```

Figure 5-11: Attack against Firefox browser.

The entire communication is far too long to include here, but you can see that some JavaScript is being used for Base64 encoding. This is being sent to the web browser. At some point, the browser would probably run this function to encode some data. Though it's not included here, there would be a Base64 decode function as well. What also isn't included is the applet that was sent to the browser to be executed. Fortunately, if you have the packet capture, you can extract any applet that may be included in the attack. The applet can then be deconstructed to determine what it is trying to do.

Considering the prevalence of web applications and the usage of web browsers, it's not surprising that a large number of attacks are run against them. We'll take a look at some of them here. The first one that we'll look at makes use of the Structured Query Language (SQL), which is used to interface with a database server. Because web applications generally need to store data, from usernames and passwords to credit cards and retail products and a wide variety of data in between, the web application needs to be able to interact with the database server. This happens with SQL statements that are executed on the database server. These statements are commonly called *queries*.

A SQL injection attack is one where the attacker generates some SQL that is expected to be sent in to the application server and then passed through to the database server. Often, this would be a SQL fragment to take advantage of the fact that within the application is another SQL fragment that the user input is intended to be merged with to create a fully functional SQL query. In Figure 5-12, you can see a SQL injection attack that was used to get the list of users out of the database. The initial

SQL statement was intended to compare a user ID with the one from the database in order to print information about that user. By appending ' or 'a' = 'a we trick the SQL server to close the partial query it has, presumably checking the user ID column against the value we were supposed to put in. Once the initial query is closed with the ' we can append our own SQL fragment. What we are doing is saying that if the user ID is '', meaning blank, or as long as 'a' is equal to 'a', every row in the database would be considered a match and printed.

```
GET /dvwa/vulnerabilities/sqli/?id=%27+or+%27a%27+%3D+%27a6Submit=Submit HTTP/1.1
Host: 192.168.1.61
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.116 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://192.168.1.61/dvwa/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: security=low; PHPSESSID=1973a8571f26b1d0d8b9884c37e1d15d

HTTP/1.1 200 OK
Date: Sat, 17 Sep 2016 20:52:26 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Pragma: no-cache
Cache-Control: no-cache, must-revalidate
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Content-Length: 4665
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

Figure 5-12: SQL injection attack.

Because the request is being sent as part of the URL, you can see an example of URL encoding. Special characters and spaces are not allowed in URLs. Since this SQL query includes a number of these characters, the URL had to be encoded to be passed to the server. You can see the SQL attack at the top of the screen capture in the GET line. The response, which you can only see the HTTP header for, was a successful listing of all of the users that were stored in the database.

Another attack that you may see is a cross-site scripting attack (XSS). This is where the attacker sends JavaScript into the server so that it can be executed within a browser on the system of an unsuspecting user. The reason it's called cross-site scripting isn't because the JavaScript is spread across sites, but because using JavaScript, the attacker can obtain information about sites the user has visited other than the site the JavaScript came from.

There are two types of cross-site scripting (XSS) attacks. The first is a reflected attack. This means the attack has to be part of the URL. With a specially crafted URL, you can send it to users and get them to click on the link. For example, the URL `http://www.badtarget.com/index.php?name=guest<script>alert('attacked')</script>` would be an example of XSS embedded into the URL. The JavaScript embedded in the URL will then be executed by the target's browser. The other type of attack is a persistent XSS attack. With this attack, the attacker sends the JavaScript into a web application. The JavaScript is then stored in the database for the web application. Once it is in the

database, it is available for attacking any subsequent visitors. Consider a guestbook on a website, as is the case in Figure 5-13. The attacker sends in some JavaScript, and it gets stored so that every visitor that comes and visits the guestbook will cause the JavaScript to be executed.

```
POST /dvwa/vulnerabilities/xss_s/ HTTP/1.1
Host: 192.168.1.61
Connection: keep-alive
Content-Length: 94
Cache-Control: max-age=0
Origin: http://192.168.1.61
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.116 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://192.168.1.61/dvwa/vulnerabilities/xss_s/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8
Cookie: security=low; PHPSESSID=1973a8571f26b1d0d8b9884c37e1d15d

txtName=Me&mtxMessage=%3Cscript%3Ealert%28%27hi%27%29%3B%3C%2Fscript%3E&btnSign=Sign+GuestbookHTTP/1.1
200 OK
Date: Sat, 17 Sep 2016 20:53:23 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Pragma: no-cache
Cache-Control: no-cache, must-revalidate
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Content-Length: 4946
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

Figure 5-13: XSS attack.

Guestbooks are not the only types of web applications that can be attacked. Any web application that takes user data and stores it, then displays it back to other users is a good target for an XSS attack. Similarly, any page that takes input from the user and sends it into the server as part of the URL can also be attacked using the reflected style of XSS. A number of ways exist to protect against this attack, but XSS attacks are still very common, especially with lesser-known applications.

Another web application attack is command injection, where the attacker sends commands into a web server to be run by the operating system. In an attack like this, you will see the operating system shell command being sent along with the HTTP. XML attacks use a crafted Extensible Markup Language (XML) message to get sent into the web application that then parses the XML. This is a way to get data or even make changes from the operating system.

Web applications are also susceptible to fuzzing attacks. Any application that takes input from the user and acts on that data may be manipulated into doing something with that data. If the application is expecting a particular type of information, like a password as in the FTP attack shown earlier, and it gets something else, the application may mishandle it. This is just as true with web applications as it is with traditional programs. Fuzzing attacks try to get the web application to crash or to give up information that may allow the attacker to attack the server or application in a different way.

Though there are a number of attack types against web applications, the advantage to these attacks is that they are in a form that can be identified, as they are text-based. Since these are not binary

attacks, they can be read. While they may be encoded, the decoding will commonly be easy because the data will often be Base64 or URL encoded when the communication is taking place between a web server and a web browser.

Summary

When it comes to the networked world, attackers can cause damage to systems and networks in a number of ways. Some of them, like denial of service (DoS) attacks, require no sophistication or even much in the way of knowledge or expertise. A denial of service attack often simply requires brute force. This may mean just generating a large amount of traffic to be sent to the target. A SYN flood, for example, isn't even always about overloading the network connection. Initially, it was just about sending so many SYN messages that the server couldn't accept any additional half-open connections. This is not to say that a SYN flood can't also be about overloading the network connection so legitimate traffic can't get through. Any sort of flood, whether it's a SYN flood, a UDP flood, or an ICMP flood, can be cause for concern because these types of attacks require help from the Internet Service Provider (ISP) to mitigate. By the time the traffic hits a device on the enterprise network, it's too late because the network connection is already full. All of the inbound bandwidth is being consumed by the attack.

A denial of service attack like a flooding attack requires volume. Modern systems and network connections can take a lot, so it's unlikely a single system could generate enough volume itself to have much of an impact. This is where amplifiers are essential. A Smurf attack, for example, will make use of misconfigured networks and ICMP messages to generate a large number of duplicate messages targeted at the victim. DNS requests can also be used to generate amplification attacks.

With attacks like Smurf attacks, DNS amplification attacks, and even SYN floods, the attacker will spoof a source address. Any response to a spoofed source address will not return to the attacker, but instead will return to the owner of that address. In some cases, the address may be unallocated or unused. It may be part of a reserved block of addresses. When responses go back to systems that don't exist, this is called backscatter. Backscatter analysis can be used to obtain evidence that a denial of service attack is underway.

Although a denial of service attack is bad because it prevents legitimate users from gaining access to services offered, other attacks may be of even more concern. Vulnerability exploits can lead to system compromise, putting the system under the control of the attacker. This may result in data theft and corruption. Protecting against these sorts of attacks is a challenge that businesses generally take seriously. They introduce detection and prevention systems to counter these attacks. The attackers respond by utilizing evasion techniques including encoding and fragmentation to allow the attack to succeed and not be detected.

Vulnerability exploits against programs are a problem, because it can allow attackers to control the flow of execution of the program. There are other ways to attack applications, however. While these

are still vulnerabilities, it's not the same as attacking the application code to control the execution flow in order to get direct access to a system shell, which is a common objective. Web applications are popular ways to provide functionality to users. Attacking web applications requires skill, but not in the same way that other attacks might. Though these attacks may also use encoding techniques, they are primarily plaintext attacks, which can help with the detection of the attack.

Detecting these attacks requires understanding different encoding techniques as well as understanding some protocol basics to know when something isn't right. Fortunately, a lot of application protocols like SMTP, FTP, and HTTP are text-based, so may even be understood just by looking at the commands, since they are often some form of English word.

6

Location Awareness

In this chapter, you will learn about:

- The impact of time zones on investigations
- How to get location information from the network
- How location-based services work with Web applications
- How to get location information from WiFi

He sits in the dim glow of his laptop screen, knowing he is more than half a world away from the system he is really working on. It's late at night and the world outside is blanketed by darkness. He moves carefully on the system because, while it's late at night where he is and dark, it would be light and into the business day on the system he is connected to. Fortunately, he isn't directly connected to the system on the other end. Instead, he has bounced through a couple of intermediate systems. He knows that even if someone were watching, having those additional hops in between will make it harder to track him down.

The time difference is something that he always has to factor in to make sure he isn't being too noisy while the legitimate user of the system is trying to use it. If he is using too much network or too much disk, that may get noticed because it will cause performance problems and the user may well take notice of the changes on the system. As a result, he always has to be aware of where the system he has compromised is. He has a number of ways to know this but the easiest is just checking the time zone setting on the system. This isn't always accurate, however, since some servers use Greenwich Mean Time (GMT) as their time zone to be able to line up log files across an organization into a consistent timeline. It will also only give him a region and not a specific location, though he doesn't need that so much.

Systems can identify where they are located in a number of ways. Some of this information is available from the network, and can be as simple as just a time zone from a DHCP server. However, smartphone applications that became reliant on global positioning systems (GPS) to obtain a location have driven a need for devices to get locations in other ways. While mobile applications can acquire location information, they are not alone in this capability. If you visit particular websites, you may notice that your web browser asks if you want to provide a location to the website. Your computer

may use different strategies to acquire the location information and provide it to the server that is requesting it.

As more systems become mobile, whether they were designed to be permanently mobile like smartphones and tablets, or whether they are just sometimes mobile like a laptop, the device needs to be more aware of where it is in time and space. There are a number of reasons for this. One reason is that many applications want to know where you are in order to provide more accurate information. Not all devices have global positioning systems (GPS), however, so to provide the same level of service, there needed to be a means that would allow systems without GPS to know where they are.

Although databases are available that track information related to WiFi networks in order to provide location-based services, other ways exist to get information about where a system may be located. As a starting point, just knowing what public Internet Protocol (IP) address is being used can provide information about the location of the system. You can get this information in different ways with varying levels of accuracy.

NOTE Acquiring a physical location can provide a point of information that may be relevant to an investigation. While location information is important, because it can tie a particular computer to a physical location, that doesn't mean that the user can necessarily be tied to that same location.

Time Zones

When it comes to computers, time is relative. Every computer can be configured to know what time zone it is in. This allows computers around the world to correlate events across multiple systems because their timestamps can place events in a consistent time line. A time zone is a recognition that the Earth is a sphere that revolves in space, providing us with a way to measure the passage of time. Because it's a sphere, different parts of the globe are at different times of the day. This is because we use the sun's position in the sky to calculate time. When the sun is directly overhead, more or less, we consider this to be noon. Since the sun is more or less directly overhead at different moments (it would be directly overhead for me on the East Coast when it is nowhere near to being overhead in Los Angeles, for example), we use time zones so that time appears to be normalized. Noon is when the sun is essentially overhead.

The origin or reference time zone is based on the observatory in Greenwich, England. In the 1800s, in light of the importance of the Greenwich Observatory to astronomy and navigation, the prime meridian 0 was established to run through Greenwich. This means that the line of longitude with a degree of 0 is the line of longitude that runs through Greenwich. Every other line of longitude is calculated mathematically based on an origin of that prime meridian.

NOTE Longitude and latitude are ways of breaking the globe up into measurable units. They provide a way of location orientation at any point on the sphere we call Earth. Lines of longitude are those that run from one pole to the other and as such, the measurement is east and west. Where Greenwich, England is 0, anything to the west starts counting positively from there to the opposite side of the world at 180 degrees. Longitude measurements east of Greenwich are measured in negative numbers to -180. This means that in total, there are 360 degrees of longitude around the world.

It's necessary to keep time zones in mind as you are working with any piece of information that has a timestamp. You need to know the time zone the system is in so you can create a coherent understanding of when events happened. Coincidentally, if you are told the time zone, you have a better understanding of where the system is. This is not a guarantee, however, because many systems are configured not to provide that information in their network communications. As an example, Listing 6-1 shows a set of HTTP headers with a timestamp that shows that the time is set to be GMT, or Greenwich Mean Time.

Listing 6-1: HTTP Headers Showing Timestamp

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/7.5
X-AspNetMvc-Version: 3.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 07 Nov 2016 03:04:07 GMT
Content-Length: 38588
```

Operating systems handle time zones in different ways. In a Linux system, for example, there may be a file in the `/etc` directory that points to a file providing specific details about the time zone. You can see in Listing 6-2 that the `/etc/localtime` file points to a different file altogether, indicating that this system is on the East Coast. Not all Unix-like operating systems will use links to point to the zone file. Some will use a copy of the zone file to stand for the `/etc/localtime` file. While the time zone suggests it's in New York, New York is just one of the cities that has been designated to indicate what time zone the system is in. The properties of the location "New York" convey to the system that it is in the East Coast time zone and also adheres to daylight savings time. Although you can set the time zone using the graphical user interface components, ultimately what is happening is the time zone is set using the `/etc/localtime` file.

Listing 6-2: Time Zone

```
kilroy@oliver:~$ ls -la /etc/localtime
lrwxr-xr-x 1 root wheel 36 Sep 20 13:37 /etc/localtime ->
/usr/share/zoneinfo/America/New_York
kilroy@oliver:~$ date
Mon Oct 17 21:34:47 EDT 2016
```

The process is different on a Windows system, but just as with Linux, everything related to time is relative to where you are in the world in relation to Greenwich Mean Time. In Figure 6-1, you can see a partial list of the time zones that are available to be configured in Windows. According to documentation at Microsoft's Developer's Network, 75 possible time zones can be configured on a Windows system.

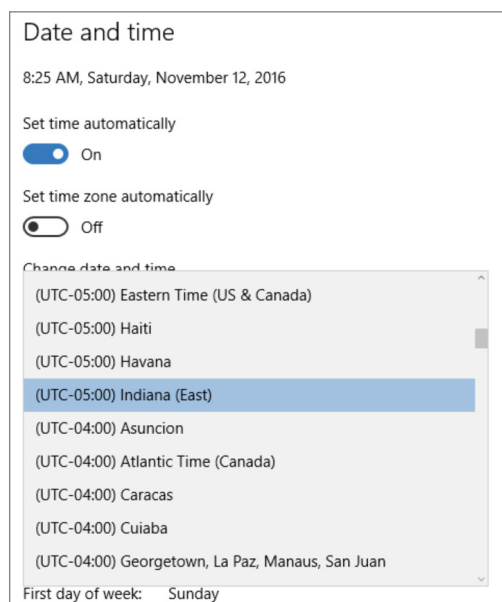


Figure 6-1: Windows time zones.

Unlike Linux systems where configuration files are typically stored in plaintext files in the `/etc` directory, Windows systems store their configuration in the registry. As you can see in Figure 6-2, the time zone setting on a Windows system is stored by name in the registry. The key holding this information is `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Time zoneInformation`.

Name	Type	Data
(Default)	REG_SZ	(value not set)
ActiveTimeBias	REG_DWORD	0x0000012c (300)
Bias	REG_DWORD	0x0000012c (300)
DaylightBias	REG_DWORD	0xffffffff (4294967236)
DaylightName	REG_SZ	@tzres.dll,-111
DaylightStart	REG_BINARY	00 00 03 00 02 00 02 00 00 00 00 00 00 00 00 00
DynamicDaylightTimeDisabled	REG_DWORD	0x00000000 (0)
StandardBias	REG_DWORD	0x00000000 (0)
StandardName	REG_SZ	@tzres.dll,-112
StandardStart	REG_BINARY	00 00 0b 00 01 00 02 00 00 00 00 00 00 00 00 00
TimeZoneKeyName	REG_SZ	Eastern Standard Time

Figure 6-2: Windows registry time zone settings.

Time zones are useful to know about and they can provide some general direction about where systems are located. The challenge, though, is that time zones are not always that reliable. Any user can set any time zone on their system. Additionally, when laptops or other mobile devices move around, the time zone typically remains unchanged, unless the user dislikes the clock on her computer being wrong for the duration of her stay in a different location. Some protocols will include the time zone that has been configured on the server sending the information. However, since this is configured, it may not provide an accurate physical location. What you have is whatever location has been configured on the server.

Using whois

The Internet registries can also provide a large amount of location-related information about IP addresses. When blocks of IP addresses are allocated, the information about the new owner is registered with one of the regional Internet registries. The same is true with domain names and other identifying information related to the Internet. Using this information can also help to provide location information, though, as suggested below, it may not be sufficient. One of the challenges with using the Internet registries is that IP address blocks are generally registered to a company and while the company's business information, including address and phone number potentially, may be available in the registry, there is no guarantee that the IP address you have identified is located at the address provided. Large companies commonly have a headquarters and a number of other locations. The IP address would probably be registered to the headquarters, and the physical address of the corporate headquarters is what you will be able to identify.

The IP address could very well be located in a satellite office somewhere because once IP addresses are allocated to a company, no one bothers to check to see where the addresses are being used. That is entirely up to the discretion of the company the addresses have been registered to.

It's also possible that the information provided within the Internet registry is the service provider that was originally provided with the IP address block. Service providers may hand out blocks for the use of their customers without actually assigning ownership of the block to the company that is using it. This may also provide you some location information, however. In the case of smaller service providers, which may be more likely to engage in this practice because of the limited number of address blocks they have been able to get, their customers are likely to be local to them. If you were a small business in Vermont, for instance, it would be highly unlikely for you to make use of a small service provider in Colorado. This means that if you find that the address you have is registered to a service provider in Colorado in, say, Durango, you have a good idea that the customer who is using the IP address is likely also located in or near Durango.

While none of this may be all that useful if you are thinking about just getting to an end goal, a lot of information can be obtained from a lookup at an Internet registry. Fortunately, you can perform these lookups in a number of ways. One way is to just use the `whois` command. You can see the use of a command-line version of `whois` in Listing 6-3.

Listing 6-3: whois Output

```
kilroy@oliver:~$ whois 4.2.2.1
# comment text removed from this output
NetRange:      4.0.0.0 - 4.255.255.255
CIDR:          4.0.0.0/8
NetName:       LVLT-ORG-4-8
NetHandle:     NET-4-0-0-0-1
Parent:        ()
NetType:       Direct Allocation
OriginAS:
Organization:  Level 3 Communications, Inc. (LVLT)
RegDate:      1992-12-01
Updated:      2012-02-24
Ref:          https://whois.arin.net/rest/net/NET-4-0-0-0-1

OrgName:       Level 3 Communications, Inc.
OrgId:         LVLT
Address:       1025 Eldorado Blvd.
City:          Broomfield
StateProv:     CO
PostalCode:    80021
Country:       US
RegDate:      1998-05-22
Updated:      2012-01-30
Comment:      ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
Ref:          https://whois.arin.net/rest/org/LVLT
```

Using `whois`, we can see that the owner of the IP address 4.2.2.1—a common DNS caching server that can be used by anyone on the Internet—is Level 3 Communications. Level 3 is located in

Broomfield, CO, though from personal knowledge I can tell you that 4.2.2.1 specifically is not located there. However, in smaller organizations that are not Internet service providers, this information may be useful.

If you are not comfortable with command-line utilities or you don't have a Unix-based system to run `whois` from, you can accomplish the same thing in other ways. For example, `whois` utilities are available for Windows. You can find them in places where you can get access to free utility software. Additionally, a number of websites will provide you the ability to do `whois` lookups. These sites work identically to the `whois` utility that was shown earlier. An example of one of these sites is shown in Figure 6-3. This particular site is at www.whois.com, though a number of other websites will also work. Some of these sites that offer `whois` lookups only work with domain names, so you have to be sure that you have a site that will do IP address lookups in addition to domain names.

Whois IP 4.2.2.2 Updated 1 day ago

```
#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/whois_tou.html
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/public/whoisinaccuracy/index.xhtml
#

#
# The following results may also be obtained via:
# https://whois.arin.net/rest/nets;q=4.2.2.2?
# showDetails=true&showARIN=false&showNonArinTopLevelNet=false&ext=netref2
#

NetRange:      4.0.0.0 - 4.255.255.255
CIDR:          4.0.0.0/8
NetName:       LVL3-ORG-4-8
NetHandle:     NET-4-0-0-0-1
Parent:        ()
NetType:       Direct Allocation
OriginAS:
Organization:  Level 3 Communications, Inc. (LVL3)
RegDate:      1992-12-01
Updated:      2012-02-24
Ref:          https://whois.arin.net/rest/net/NET-4-0-0-0-1

OrgName:       Level 3 Communications, Inc.
OrgId:         LVL3
Address:       1025 Eldorado Blvd.
City:          Broomfield
StateProv:     CO
PostalCode:    80021
Country:       US
RegDate:      1998-05-22
Updated:      2012-01-30
Comment:      ADDRESSES WITHIN THIS BLOCK ARE NON-PORABLE
Ref:          https://whois.arin.net/rest/org/LVL3
```

Figure 6-3: whois lookup.

This `whois` lookup is for a different IP address than the lookup done earlier. It's done from the same block of IP addresses, though. You may have noticed from the `whois` lookup that Level 3 owns the entire 4.x.x.x block of addresses, so anything else in that block of addresses will show Level 3

as one of the owners. Since IP addresses are handed out in a hierarchical fashion, you may see a chain of owners, depending on whether the block has been re-assigned or just temporarily assigned.

As noted before, you can also look up domain names using the same techniques. Domain names are less specific than IP addresses, though you can still obtain the same location information. As you see in Figure 6-3, physical addresses are provided. However, companies that own domain names may have multiple locations so this is just one piece of information. It may be necessary to locate additional pieces of information to be clearer about the location.

Related to using *whois*, you can also use DNS to obtain location information. At the moment of this writing, the public IP address of my cable modem is 73.219.13.135. Using DNS tools, I can obtain the hostname of that IP address. Though I could do this lookup in multiple ways, I am using the *host* utility provided in the Linux distribution I am using in Listing 6-4 to obtain the hostname.

Listing 6-4: Host Lookup of IP Address

```
kilroy@oliver:~$ host 73.219.13.135
135.13.219.73.in-addr.arpa domain name pointer
c-73-219-13-135.hsd1.vt.comcast.net.
```

Using the hostname, I can determine that the IP address is located in Vermont, which is correct. I can tell that by the portion of the hostname that says *vt.comcast.net*. This is a subdomain that Comcast uses to house IP addresses and other DNS resources for customers in Vermont. Not all organizations use their DNS hostnames to indicate where those hostnames are located, but generally Internet service providers do because it makes troubleshooting quite a bit easier. These hostnames can also be identified in more of a bulk fashion, so even if the hostname with the location isn't your target, it's possible to get a collection of hostnames that can point at a particular geographic region.

Traceroute

Traceroute is a diagnostic tool used by technical professionals looking to identify a problem with network routing. Traceroute works by making use of the time to live (TTL) IP header field. Normally, IP packets include a default IP header value. Every time a packet passes through a routing device, the time to live field is decremented. Once the TTL reaches 0, the device that decremented the field to 0 returns an ICMP error message to the source of the original message indicating that time to live has been exceeded in transit. Traceroute makes use of this capability. Traceroute will send a message out to a destination with increasing TTL values. The first packet being sent has a TTL of 1. When the very first router (the default gateway on your network) receives the message, it decrements the TTL to 0 and responds with the ICMP error message. Once the sending system receives the message, it has the IP address of the first router.

The sender only has the IP address, though, which means that the system running traceroute has to do a DNS lookup to get the hostname that is associated with the IP address. This is a reverse lookup and requires that whoever owns the IP address has the pointer (PTR) record configured in the DNS server. A reverse address, provided by a PTR record is good to have but it is not required. They are a convenience, and PTR records can be even less likely to be configured. Generally, however, service providers will keep their DNS records up to date. Because they are the ones who will commonly include location information in the hostnames, they are the ones we are going to be most concerned with.

To get an idea where something is located using traceroute information, you simply run a traceroute and look at the output. Once you get the hang of it, reading the location out of the traceroute is fairly easy. The example shown in Listing 6-5 was done from a Mac OS X system and the utility is named traceroute. On a Linux system, it will also be named traceroute. On a Windows system, because traceroute exceeded the 8-character limit of the 8.3 naming convention from the DOS days, the utility is named tracert. Even though the 8.3 naming restrictions no longer exist, the name of the utility has remained tracert, presumably for consistency's sake. Pathping is another Windows utility that can be used to identify a network path.

Listing 6-5: Traceroute Output

```
kilroy@oliver:~$ traceroute www.google.com
traceroute to www.google.com (172.217.1.68), 64 hops max,
52 byte packets
 1  172.30.42.1 (172.30.42.1)  1.397 ms  1.031 ms  1.509 ms
 2  96.120.70.65 (96.120.70.65)  9.103 ms  8.709 ms  8.679 ms
 3  ge-4-19-ur01.wolcott.ct.hartford.comcast.net (68.86.237.249)
   8.289 ms  8.348 ms  7.690 ms
 4  te-7-3-ur05.foxboro.ma.boston.comcast.net (68.86.224.201)  16.035 ms
   be-127-ar01.needham.ma.boston.comcast.net (68.86.225.113)
   17.467 ms  17.706 ms
 5  he-0-11-0-0-ar01.needham.ma.boston.comcast.net (162.151.112.17)
   17.431 ms
   be-7015-cr02.newyork.ny.ibone.comcast.net (68.86.90.217)  24.078 ms
   he-0-11-0-0-ar01.needham.ma.boston.comcast.net (162.151.112.17)
   15.502 ms
 6  be-7015-cr02.newyork.ny.ibone.comcast.net (68.86.90.217)  23.984 ms
   hu-0-10-0-1-pe02.11leighthave.ny.ibone.comcast.net (68.86.86.254)
   23.333 ms
   be-7015-cr02.newyork.ny.ibone.comcast.net (68.86.90.217)  24.226 ms
 7  hu-0-10-0-1-pe02.11leighthave.ny.ibone.comcast.net (68.86.86.254)
   22.760 ms  23.178 ms  22.462 ms
 8  209.85.245.116 (209.85.245.116)  23.303 ms  24.226 ms  25.214 ms
```

```

 9  209.85.245.181 (209.85.245.181)  25.803 ms
    209.85.245.116 (209.85.245.116)  24.113 ms
    209.85.245.181 (209.85.245.181)  24.221 ms
10  lga15s44-in-f4.1e100.net (172.217.1.68)  24.938 ms
    209.85.245.181 (209.85.245.181)  24.647 ms
    lga15s44-in-f4.1e100.net (172.217.1.68)  23.942 ms

```

As noted earlier, the very first thing I get in the output is the IP address of the default gateway on my local network. I am in the habit of using addresses from the 172.16.0.0/12 range on my networks. This is a range of private IP addresses, just as the 192.168.0.0 address range is, which is more common on home networks. Because I don't have a DNS server that includes any of my local systems in it, there is no reverse lookup to be had for that address. The first place we get a real hostname is on line 3. You can see the hostname listed as `ge-4-19-ur01.wolcott.ct.hartford.comcast.net`. This is a port on a network device in Hartford, CT. The `ge` indicates that this is a gigabit Ethernet (ge) port. The numbers after that could indicate slot and port in a large chassis. The `ur01` indicates a router. Service providers will sometimes use short names to indicate the type of router within the network. If you see `cr`, it is probably a core router, meaning a device in the core or deep inside the network. An `ar` router would be an access router, where customers may commonly connect.

NOTE A couple of notes on the traceroute responses. First, the times shown are the round trip times to the individual host in the path. Second, in cases where you see multiple entries associated with a particular hop, it means that there are multiple pathways that are the same network distance to the destination.

In general, you will see the type of interface followed by the slot and port numbers, if they exist, in the first part of the hostname. After that, you may well see the location information. In some cases, as in lines 3–5, the name will be pretty straightforward. You are seeing multiple entries on those lines because traceroute sends three messages. If there are multiple paths through the network to get to a particular location, each successive message may hit a different router in the network. That appears to be the case here. It may also indicate some routing distribution or load balancing, depending on where the message is located.

In some cases, the hostnames are even more specific, depending on the provider. The Comcast entries that include `ibone` indicate there are routers at 111 8th Avenue in Manhattan. This particular building is owned by Google and has a meet-me room where multiple carriers get together and hand off traffic to one another. The traceroute goes through a few hops in that building before departing to a number of IP addresses that don't have reverse lookups associated with them. Because of that, we don't really know where they are located. However, the traceroute terminates at the hostname `lga15s44-in-f4.1e100.net`.

The domain name `1e100.net` is a domain name that Google uses to identify servers within its network. This particular hostname enables us to identify another way of looking at locations within service provider hostnames. For a long time, it was fairly common for service providers to identify

locations within their network by the code for the airport in the city where the devices are located. If you see a three-letter indicator in a hostname, it may well be an airport code. LGA is LaGuardia Airport, located on Long Island. LGA provides services to Manhattan, so the servers that we have terminated at are located in New York. This doesn't necessarily provide us with a building address, though. Google owns only a small number of buildings in New York City and of those buildings, not all of them house data centers. There is a good chance that the building we have terminated at is also located in 111 8th Avenue.

Traceroute can provide a lot of details that are not only useful for network engineers, but also can provide some location information for investigators, once you learn how to read the output. While IP addresses do map to hostnames, you can get locations from IP addresses in other ways. There is nothing about an IP address that inherently provides a location, but with a little help, we can get fairly specific about where the IP address is located, if you know how to read the hostname that is associated with the IP address.

Geolocation

When Voice over IP (VoIP) services became commonplace, there was a challenge. Federal regulations require telecommunications providers to be able to support enhanced 911 (E-911) services to phone subscribers. Anyone dialing 911 should be able to be located by the phone network. In a traditional phone network, this is easy because the phones are hard-wired to the central office and each subscriber has an address associated with it. If a call comes from a particular phone number using a wired line, it's guaranteed that the call has come from a specific physical address because hard-wired lines can't be moved. When the caller dials 911, the central office knows which public service access point (PSAP) to route the call to.

VoIP, though, uses interface devices that convert traditional phones and the signals they use to IP. These devices can be taken anywhere. As long as they can get an IP address and can communicate with the servers within the VoIP provider network, there is nothing to prevent the service from being used. That, however, causes problems for the service providers because they are required to be able to hand off location information for their subscriber. As noted earlier, there is nothing inherent about an IP address that can provide physical addresses, and while it is possible to read hostnames and network paths to get some location out of them, the hostname and network path don't have nearly the specificity required by E-911.

At a minimum, the service provider needs to be able to know which PSAP to route the call to. There are a number of ways to do this, including just hard-coding the subscriber into a database associated with a particular PSAP. VoIP services are not the only ones where location information from IP addresses is important or at least very useful. As a result, there are databases that will keep track of that information, as well as web interfaces that can perform lookups from IP addresses. In fact, some of these websites will tell you where you are based on your IP address.

As it turns out, a number of geolocation providers and some of the websites that you can do lookups from will provide information from the different databases. Just to demonstrate some of the challenges associated with looking up geographic location from an IP address, you will sometimes get different locations. To highlight that point, Figure 6-4 shows location information related to an IP address belonging to Google. This is information from three different databases, though the site in question, www.iplocation.net, provides results from many other databases. While two of them appear to show the same location, when you look at the latitude and longitude, they are quite different. The two showing the same city will map to very different locations.

Geolocation data from IP2Location (Product: DB6, updated on 2016-11-1)			
IP Address	Country	Region	City
172.217.1.68	United States 	California	Mountain View
ISP	Organization	Latitude	Longitude
Google Inc.	Not Available	37.405990500586	-122.07851409912
Geolocation data from ipinfo.io (Product: API, real-time)			
IP Address	Country	Region	City
172.217.1.68	United States 	California	Mountain View
ISP	Organization	Latitude	Longitude
Google Inc.	Google Inc.	37.4192	-122.0574
Geolocation data from EurekAPI (Product: API, real-time)			
IP Address	Country	Region	City
172.217.1.68	United States 	Georgia	Mcdonough
ISP	Organization	Latitude	Longitude
Google	Google	33.4514	-84.187

Figure 6-4: Geolocation lookup.

The third location is not only in a different city and state but most of the way across the United States. A fourth database shows New York and the fifth shows Mountain View again. As a result, you have a start on a location from the IP address but it is by no means definitive. In some cases, all the lookup service is doing is running a `whois`, getting the owner of the IP address, and providing the city for that owner. As previously discussed, that's not always that useful.

One of the databases at `db-ip.com` is not only more accurate but will also use IPv6 to perform a lookup. Some of the backbone providers are using IPv6 to communicate back and forth. In Figure 6-5, you can see a lookup of my external IPv6 address. While the address belongs to Comcast, `db-ip.com` isn't just providing the location of the IP address according to *whois* because that would be based on Comcast's address, which is not in Vermont. However, while we are very close to a real location, the database maps this address to a town that is nearby rather than the town I am actually located in.

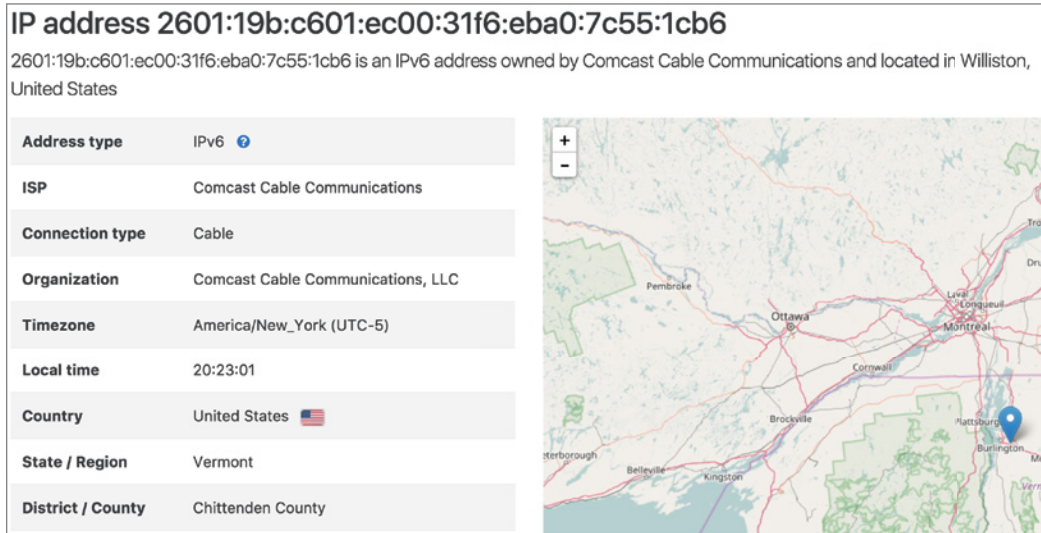


Figure 6-5: `db-ip.com` lookup.

The company MaxMind maintains several databases related to location information and mapping network information. These databases can be integrated with Wireshark to save the effort of performing multiple lookups using a web interface. You can download lite versions of the databases from MaxMind and then tell Wireshark where the databases are using the preferences settings. There is a configuration setting for the locations of GeoIP databases. MaxMind provides databases for both IPv4 and IPv6 as well as information about the autonomous system (AS) number used by service providers for routing purposes, the city where the address is located, and the address in longitude and latitude form.

Once you have a packet capture, you can look at the Endpoints dialog box in the Statistics menu. This collection of information will give you IP addresses that were found in your packet capture, and if there are entries in the MaxMind databases, they will display the information. You can see an example of this in Figure 6-6.

Country	AS Number	City	Latitude	Longitude
United States	AS8068 Microsoft Corporation	Redmond, WA	47.680099	-122.120598
United States	AS8068 Microsoft Corporation	Redmond, WA	47.680099	-122.120598
United States	AS714 Apple Inc.	Cupertino, CA	37.323002	-122.032204
United States	AS714 Apple Inc.	Cupertino, CA	37.323002	-122.032204
United States	AS714 Apple Inc.	Cupertino, CA	37.323002	-122.032204
United States	AS714 Apple Inc.	Cupertino, CA	37.323002	-122.032204
United States	AS714 Apple Inc.	Cupertino, CA	37.323002	-122.032204
United States	AS6185 Apple Inc.	Cupertino, CA	37.323002	-122.032204
United States	AS6185 Apple Inc.	Chicago, IL	41.848301	-87.651703
United States	AS14618 Amazon.com, Inc.	Ashburn, VA	39.048100	-77.472801
United States	AS1273 CW Vodafone Group PLC	Cambridge, MA	42.362598	-71.084297
Hong Kong	AS8075 Microsoft Corporation	-	22.250000	114.166702
United States	AS8075 Microsoft Corporation	-	32.778702	-96.821701
United States	AS7015 Comcast Cable Communications, LLC	Cambridge, MA	42.362598	-71.084297
United States	AS7016 Comcast Cable Communications, LLC	Cambridge, MA	42.362598	-71.084297
United States	AS7016 Comcast Cable Communications, LLC	Cambridge, MA	42.362598	-71.084297
United States	AS14618 Amazon.com, Inc.	Ashburn, VA	39.048100	-77.472801
United States	AS8075 Microsoft Corporation	San Antonio, TX	29.424101	-98.493599
Ireland	AS8075 Microsoft Corporation	Dublin, 07	53.338902	-6.259500

Figure 6-6: GeoIP lookup using Wireshark.

Wireshark provides fields for the country the IP address appears to be located in as well as the AS number associated with the service provider, which also yields the name of the service provider. Finally, you can also see the city, longitude, and latitude columns that are associated with the IP address. Not all IP addresses will be able to be looked up in the database. This is especially true in the case of private addresses, because a private address can be associated with multiple networks around the world. As a result, any packets captured from a system on my local network will have no entries in the location columns.

Location-Based Services

Laptops and other mobile systems that don't have the capability to use GPS still have a need for location-based services. As web applications get more functionality and have to provide the same or similar services as truly mobile devices like smartphones, semi-mobile devices like laptops, or even immobile systems like desktop computers, there is a need for the application provider to obtain location-based information. The World Wide Web Consortium (W3C) has developed an application programming interface, called the Geolocation API, and a set of specifications that will allow devices that don't have GPS capability to also provide a location.

This interface is commonly provided in web pages using JavaScript. The JavaScript makes calls to a navigator object looking for the GeoIP information. This may simply be based on information about the IP address that is known using techniques referenced earlier. Other ways exist to obtain location information, however, and there is a good chance that this will continue to change over time. When your browser asks if it is okay to provide location information to the website you are visiting, it is probably using this W3C location interface.

WiFi Positioning

One way to get information about where people may be is to get someone to report on those people. This may be a self-check-in where the user provides information about himself in one form or another. However, it may also be that other people are collecting information and sharing it with a public database. This is partly the case when it comes to the WiFi Positioning System (WPS). WPS is an attempt to provide a way to locate systems using the wireless networks they are connected to. Databases are available to locate WiFi networks, and some of these databases are populated by users who collect the information and submit it to the database provider.

One of these database providers is WiGLE, which is a database for wireless hotspots around the world. Using WiGLE, you can view maps of locations and see the different WiFi networks that may be available within a particular geographic area. WiFi networks not only have a Service Set Identifier (SSID) associated with them, which is the network name, but they also have a Basis Service Set Identifier (BSSID). This looks like and often is a MAC address. The wireless access point, as a network device, has a MAC address associated with the network interface. This MAC address may become the BSSID for the wireless network to provide a layer-2 addressable identifier for the network.

WiGLE and other similar databases will not only store SSID information, but also store BSSID information. You can see an example of both BSSIDs and SSIDs in Figure 6-7.

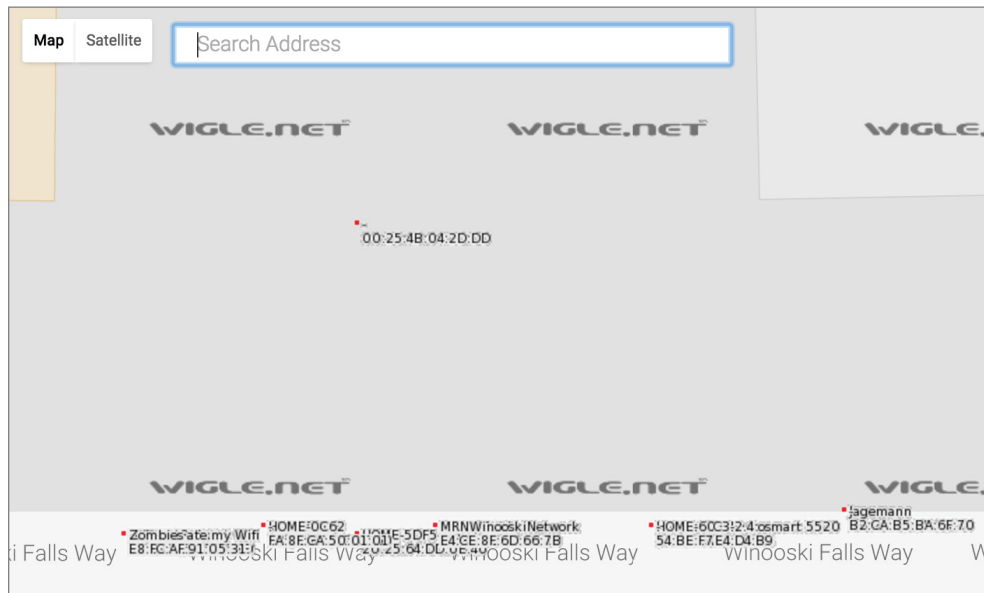


Figure 6-7: Geolocation lookup.

The map shown is a part of the website at wgle.net, and is a location nearby where I am writing this. You can search locations and zoom in on the map. You can see SSIDs like Zombies ate My WiFi, as well as BSSIDs, which just appear to be MAC addresses, and they are quite likely to just be MAC addresses. Because urban or suburban areas are likely to have large numbers of WiFi networks in close proximity, all of these WiFi networks just get overlaid on top of one another on the map. It takes zooming in very closely to be able to differentiate one network from another. Of course, by that point, you may have lost some context.

This is one way that systems can obtain information about their position. Locating systems in physical space can be challenging, and using volunteers to provide information about WiFi networks helps with that effort.

Summary

Locating addresses on the Internet is challenging. You can use a number of tools to help narrow the scope of a search, but very little is highly accurate. You can start with something very broad like using time zone information in network transmissions, if a time zone has been transmitted that is useful. As we have seen, it is not uncommon for servers to simply use Greenwich Mean Time as a time zone because it saves on calculating a relative address with other servers within the infrastructure. This may be more commonly the case with large providers that would have systems scattered across multiple time zones.

The Internet registries can be used to locate information about addresses and domain names using tools like `whois`, but even that isn't going to be very accurate. You may be able to get a location from the IP address of small businesses, but with larger businesses or service providers, the best you are going to be able to get is the address of the headquarters. This may be nowhere near where the IP address is actually being used. Fortunately, in the case of service providers, DNS information may be useful in providing a location. However, at best, this may provide a city. Rarely will a hostname provide a specific address. As we look at DNS hostnames, we can make use of the `traceroute` utility to identify the path that packets take through the network, and since service providers will often use location information in the hostname, we can see a geographic path using `traceroute`.

Databases can provide more specific information about IP addresses, but even those databases, providing AS numbers, longitude and latitude, countries, and service provider names, are often just using information about the owner of the IP address, which may have nothing at all to do with the user of the IP address. However, you can make use of these databases with programs like Wireshark to perform lookups so you can identify locations from within Wireshark. This is far easier than trying to do manual lookups one address at a time. Wireshark will also map all of your addresses, placing dots on a world map indicating where the addresses you have seen are located.

Browsers can provide location information to web applications to provide services similar to the global positioning system. However, even these are not always highly accurate, as they rely on volunteers to obtain the information and submit it to the database provider. A number of databases will track information about WiFi networks around the world, and this can be useful in providing location information, but it's not a guarantee.

7

Preparing for Attacks

In this chapter, you will learn about:

- How to prepare for attacks
- How to acquire and manage NetFlow data
- How to ensure adequate logging is in place
- The value of collecting information in a Security Information and Event Management system

He sits at his keyboard, making notes about targets. He knows, based on the size of his target, how much work he will need to do to be harder to spot as he takes on another target. Targets that are less likely to have any detection in place. These same targets may also be less likely to have any ability to look back at historical data. Targets that aren't doing much in the way of detection or logging are much easier because it means he will have to do much less work to avoid detection and also very little work to clean up after himself.

He connects to a fresh target and immediately checks to see whether there is anything running locally that might send any data anywhere, whether it's a host-based intrusion detection system or just log data. Desktop systems are far less likely to have these protections on them and because they often have permissions to where all of the interesting data is, they make great systems to attack. Easier to gain and retain access without anyone being aware and once he has access, he can continue to extract data and gain access to additional systems on the network without worrying about covering tracks or cleaning up anything. Life is so much easier when no one is paying any attention to what you're doing.

You won't always be able to rely on having a packet capture to look at when something happens that you want to investigate. In most cases, you will have to rely on a number of other artifacts as a starting point; these artifacts may be your only shot at determining what happened, so it's important that you have access to them. This may not be your call, unfortunately. Setting up incident detection systems, logging, antivirus, and other systems that will be useful for you is not typically a forensic investigator's responsibility. Of course, if you are the sole system and network administrator, you may also be expected to perform investigations.

Although this is data you will need to investigate later on, you need to have it configured up front. If you are on an incident response team for a company, you will need to make a case for having these

systems and configurations in place ahead of time. You will also need to know whether you need them, based on the circumstances, before you can expect an organization to invest in systems that will support these capabilities. Though having the data is essential when it comes to performing an investigation, businesses don't exist to perform investigations, so someone needs to justify the expenditure for systems or storage. Being prepared for investigations can be expensive.

There are many ways you can prepare for incidents ahead of time. Your resources and network may support most of what will be covered in this chapter. Depending on the likelihood of attack against your infrastructure, your organization may be okay with spending what is necessary to implement the storage required to have information on hand to respond to an incident. You will need to do the work to determine the likelihood as well as the costs that may be associated with response without having data to determine root cause and initial point of entry. Without this information, you may be susceptible again if the organization is incapable of closing the holes.

Network devices are capable of generating a fair amount of data, including the flows of traffic passing through them. This can be captured using Cisco's NetFlow protocol, even if you aren't running Cisco devices. Additionally, enabling logging on network devices will help to provide information. All of these logs and other pieces of data, of course, will require storage space. As part of determining costs to present to management, you will need to understand how long you will want to store your data for and how much space you will need, based on the record size and volume.

NetFlow

NetFlow is a protocol that was developed by Cisco Systems as a way of providing data that could be used to troubleshoot a network. Although this capability was developed primarily for network administrators to use, it also provides a lot of capability for a network forensic investigator. Like the other capabilities discussed in this chapter, it has to be enabled ahead of time to provide any value. If it isn't enabled and the architecture necessary to collect and store it isn't in place, you won't have any NetFlow data to look at.

Speaking of architectures, you will need systems in place in addition to the device that will generate the data for you. The device that will generate the data will commonly be a router. Though Cisco developed this particular protocol, it has been adopted by other vendors as well, like Nortel and Juniper. Other developers that have not adopted NetFlow specifically may have developed something similar, so just because the company you are working with doesn't have NetFlow doesn't mean you are out of luck. It may require a little more investigation to determine what is in place so that functionality can be enabled. Each device, whether it's a router or a switch, would need to have NetFlow enabled. You can see a diagram of a simple NetFlow collection architecture in Figure 7-1.

Once you have devices generating NetFlow data, you need a system in place to handle the collection. This system is called a NetFlow Collector. In Figure 7-1, the Collector is in the middle of the diagram. Each device that creates NetFlow data forwards it to the Collector. The Collector stores the data, and different types of Collectors may store their data in a different format. A Collector is just software that can run on any system. The only thing that is guaranteed with NetFlow is what

data is gathered and how it is transmitted. Even that is based on the protocol version because that determines what is transmitted. For that reason, the device generating the data needs to know what version is being used.

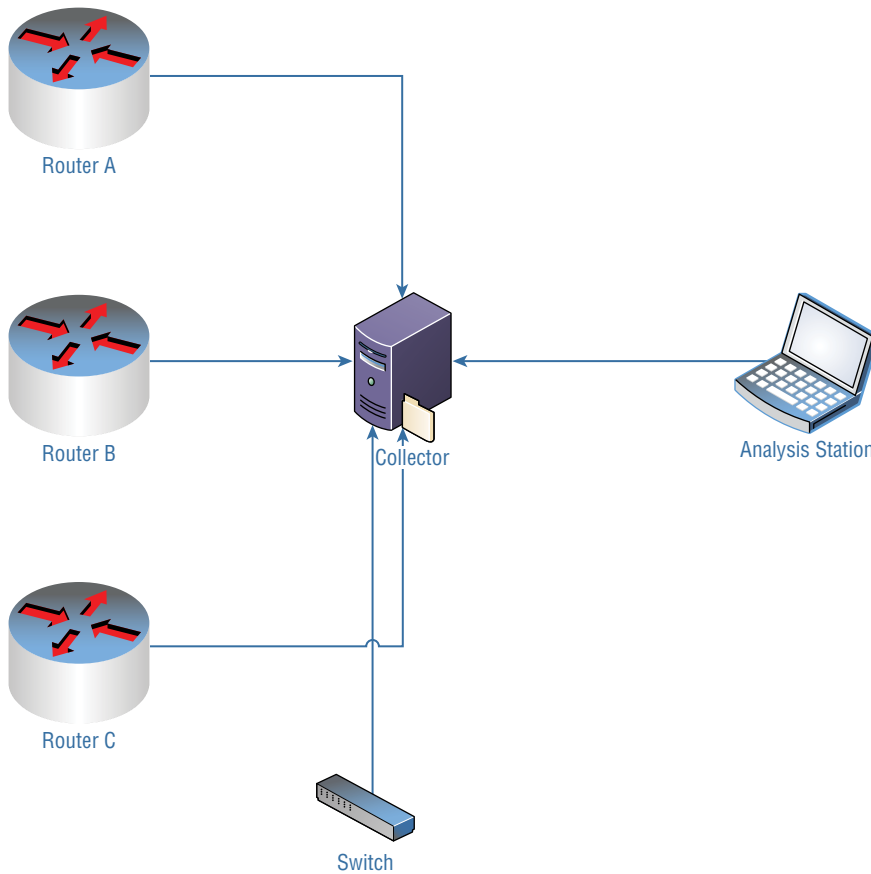


Figure 7-1: NetFlow diagram.

The Collector stores the NetFlow data in whatever format the software that handles the collection chooses to store it in. What you need then is an analysis station. In Figure 7-1, you can see this on the right-hand side of the diagram. Though the diagram depicts this as a laptop, indicating that it is probably on the user side, the analysis can be done in a variety of ways. The simplest analyzer is a program that will take the collected NetFlow data and dump it out in a text-based format, as you can see in Figure 7-2. This was done using the flow-tools package on a Linux system, which includes a number of utilities for collecting, reporting, and manipulating NetFlow data. You can see the source and destination interface as well as the source and destination ports in the output in Figure 7-2. You can also see the protocol, in hexadecimal, abbreviated to Pr.

Sif	SrcIPaddress	Dif	DstIPaddress	Pr	SrcP	DstP	Pkts	Octets
0000	172.30.42.16	0000	239.255.255.250	11	bd4d	3cf0	1	63
0000	172.30.42.21	0000	239.255.255.250	11	db39	76c	4	004
0000	172.30.42.16	0000	172.30.42.255	11	8488	3cf0	1	63
0000	172.30.42.21	0000	224.0.0.252	11	e106	14eb	2	114
0000	172.30.42.21	0000	224.0.0.252	11	cs10	14eb	2	114
0000	172.30.42.16	0000	239.255.255.250	11	80b4	3cf0	1	63
0000	172.30.42.16	0000	172.30.42.255	11	82fb	3cf0	1	63
0000	172.30.42.9	0000	224.0.0.251	11	14e9	14e9	12	3132
0000	172.30.42.16	0000	239.255.255.250	11	e839	3cf0	1	63
0000	172.30.42.16	0000	172.30.42.255	11	82be	3cf0	1	63
0000	172.30.42.21	0000	224.0.0.252	11	d314	14eb	2	114
0000	172.30.42.16	0000	239.255.255.250	11	d35d	3cf0	1	63
0000	172.30.42.21	0000	224.0.0.252	11	fcba	14eb	2	114
0000	172.30.42.16	0000	172.30.42.255	11	bd18	3cf0	1	63
0000	172.30.42.21	0000	224.0.0.253	11	e7df	dd8	2	136
0000	172.30.42.16	0000	172.30.42.255	11	881b	3cf0	1	63
0000	172.30.42.16	0000	239.255.255.250	11	9a8b	3cf0	1	63
0000	172.30.42.16	0000	172.30.42.255	11	dec7	3cf0	1	63
0000	172.30.42.21	0000	224.0.0.252	11	e788	14eb	2	114
0000	172.30.42.21	0000	224.0.0.252	11	cfe5	14eb	2	114
0000	172.30.42.16	0000	239.255.255.250	11	8060	3cf0	1	63
0000	172.30.42.16	0000	172.30.42.255	11	c00d	3cf0	1	63
0000	172.30.42.16	0000	239.255.255.250	11	d333	3cf0	1	63
0000	172.30.42.16	0000	172.30.42.255	11	8057	3cf0	1	63
0000	172.30.42.21	0000	224.0.0.252	11	e942	14eb	2	114
0000	172.30.42.16	0000	239.255.255.250	11	ea97	3cf0	1	63
0000	172.30.42.21	0000	224.0.0.252	11	fd09	14eb	2	114
0000	172.30.42.16	0000	172.30.42.255	11	bef6	3cf0	1	63
0000	172.30.42.16	0000	239.255.255.250	11	9aa6	3cf0	1	63
0000	172.30.42.16	0000	239.255.255.250	11	bd4d	3cf0	1	63

Figure 7-2: NetFlow output.

This is just a simple text-based, tabular format, but other tools can be used to present information in different ways. The diagram indicates a single laptop, but some analysis applications can be implemented in multiple systems. This may be done using a web-based interface, which would require an application to decode the stored data and also a web application that will take the decoded data and present it in a web browser in the way the user has requested. Other analysis applications may be native applications that need to be installed on the operating system you are performing the analysis on. You may also be able to look at NetFlow data using a mobile application on a phone or a tablet, depending on what operating system you are using.

An advantage to using tools like the flow-tools package is the ability to take the format the data is stored in and generate different types of output. One of those is comma-separated value (CSV) files. Using CSV output, you would be able to easily import the data into a spreadsheet program like Microsoft Excel or Google Sheets and manipulate it. This includes sorting, which will help to combine all of the source or destination IP addresses together since, by default, the output will be based on time with the earliest flows at the beginning of the output and the latest at the end. Being able to sort will help to bring a better sense of what may be happening. Using a spreadsheet program will also help you to search through the data better. Figure 7-3 shows sorted output, with one row highlighted because it lists a large number of packets. Following up on this, it may be useful to re-sort based on the number of packets or the number of octets, which may yield something else of interest.

If you are familiar with database programs, you can export to a database like MySQL using flow-tools. This would allow you to generate queries to pull out specific data from the entire flow. Spreadsheet programs will also allow some of this functionality, but database programs are designed for easy and efficient retrieval. Large quantities of flow data may be difficult to search and manipulate in a spreadsheet program, which may force you to use something like MySQL or SQLite.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	#unix_secs	unix_nsecs	sysuptime	exaddr	dpkts	doctets	first	last	engine_type	engine_id	srcaddr	dstaddr	nexthop	input	output	srcport	dstport
2	1483563901	34410000	1228330	192.168.139.1	7	532	483330	888376	0	0	172.30.42.36	132.163.4.102	0.0.0.0	0	0	123	1
3	1483563901	34410000	1228330	192.168.139.1	907	25396	441641	908012	0	0	172.30.42.36	172.30.42.1	0.0.0.0	0	0	0	20
4	1483563781	14935000	1108311	192.168.139.1	1	63	752388	752388	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	34201	156
5	1483563781	14935000	1108311	192.168.139.1	1	63	758430	758430	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	45704	156
6	1483563781	14935000	1108311	192.168.139.1	1	63	764367	764367	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	54849	156
7	1483563781	14935000	1108311	192.168.139.1	1	63	770409	770409	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	42297	156
8	1483563781	14935000	1108311	192.168.139.1	1	63	776451	776451	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	51723	156
9	1483563781	14935000	1108311	192.168.139.1	1	63	782390	782390	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	51060	156
10	1483563781	14935000	1108311	192.168.139.1	1	63	788433	788433	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	50588	156
11	1483563781	14935000	1108311	192.168.139.1	1	63	794371	794371	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	52579	156
12	1483563781	14935000	1108311	192.168.139.1	1	63	800414	800414	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	55565	156
13	1483563781	14935000	1108311	192.168.139.1	1	63	806453	806453	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	43080	156
14	1483563841	25389000	1168321	192.168.139.1	1	63	812393	812393	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	37901	156
15	1483563841	25389000	1168321	192.168.139.1	1	63	818434	818434	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	37601	156
16	1483563841	25389000	1168321	192.168.139.1	1	63	824374	824374	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	46313	156
17	1483563841	25389000	1168321	192.168.139.1	1	63	830417	830417	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	55627	156
18	1483563841	25389000	1168321	192.168.139.1	1	63	836457	836457	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	48375	156
19	1483563841	25389000	1168321	192.168.139.1	1	63	842397	842397	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	57847	156
20	1483563841	25389000	1168321	192.168.139.1	1	63	848438	848438	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	50774	156
21	1483563841	25389000	1168321	192.168.139.1	1	63	854377	854377	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	44672	156
22	1483563841	25389000	1168321	192.168.139.1	1	63	860419	860419	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	52759	156
23	1483563841	25389000	1168321	192.168.139.1	1	63	866460	866460	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	50018	156
24	1483563901	34410000	1228330	192.168.139.1	1	63	872396	872396	0	0	172.30.42.16	172.30.42.255	0.0.0.0	0	0	59327	156

Figure 7-3: NetFlow output in Microsoft Excel.

It's important to note that the output isn't always in decimal form. In Figure 7-2, the output is in hexadecimal. As noted above, the Pr column is the protocol and although it shows 11, that's really 16+1 or 17, which is the protocol number for UDP. Figure 7-4 has similar output as that shown earlier but the values are in decimal, rather than hexadecimal. The protocol is shown to be 17, indicating UDP, and the port values are also in decimal. It's important to know which base number system you are looking at. There is a significant difference between 50 and 80, for instance. If you were to look up 50 as the port number, thinking it were decimal, you'd come up with the Remote Mail Checking Protocol, which is a protocol you may never have heard of before. In fact, 50 is the hexadecimal value of 80 decimal. You may be more familiar with 80 as being the TCP port number for HTTP (web) traffic.

NOTE If you are unfamiliar with hexadecimal, it is a number system based on 16 rather than 10. Rather than counting each position from 0–9, you count from 0–F. 10 in hexadecimal is equivalent to 16 in decimal, which means F is equivalent to 15. 100 would be the same as 256, leaving FF as 255.

You may also note that Figure 7-4 lacks the Sif and Dif columns. Because NetFlow data often comes from routers or switches, where packets or frames are forwarded from one interface to another, it is useful to know what the source and destination interfaces are. This will tell you where the packet

or frame originally came from, which can be essential in looking at spoofing cases where the source IP address (or MAC address) is different from the legitimate address. In combination with a routing or ARP table, you may be able to spot instances of spoofing that would otherwise be difficult to see.

```
kilroy@oliver:~/Downloads$ flow-print -f3 < ft-v05.2017-01-04.050001-0700
```

srcIP	dstIP	prot	srcPort	dstPort	octets	packets
172.30.42.16	239.255.255.250	17	49342	15600	63	1
172.30.42.8	239.255.255.250	17	60020	1900	606	3
172.30.42.16	172.30.42.255	17	34201	15600	63	1
172.30.42.21	224.0.0.252	17	65092	5355	114	2
172.30.42.21	224.0.0.252	17	53160	5355	114	2
172.30.42.16	239.255.255.250	17	48467	15600	63	1
172.30.42.16	172.30.42.255	17	45704	15600	63	1
172.30.42.16	239.255.255.250	17	43009	15600	63	1
172.30.42.16	172.30.42.255	17	54849	15600	63	1
172.30.42.21	224.0.0.252	17	61091	5355	114	2
172.30.42.16	239.255.255.250	17	35302	15600	63	1
172.30.42.21	224.0.0.252	17	57924	5355	114	2
172.30.42.16	172.30.42.255	17	42297	15600	63	1
172.30.42.16	239.255.255.250	17	38700	15600	63	1
172.30.42.16	172.30.42.255	17	51723	15600	63	1
172.30.42.16	239.255.255.250	17	50644	15600	63	1
172.30.42.16	172.30.42.255	17	51060	15600	63	1
172.30.42.21	224.0.0.252	17	56242	5355	114	2
172.30.42.21	224.0.0.252	17	56123	5355	114	2
172.30.42.16	239.255.255.250	17	36240	15600	63	1
172.30.42.9	239.255.255.250	17	63121	1900	615	4
172.30.42.16	172.30.42.255	17	50500	15600	63	1
172.30.42.9	239.255.255.250	17	61137	1900	615	4
172.30.42.16	239.255.255.250	17	52662	15600	63	1
172.30.42.16	172.30.42.255	17	52579	15600	63	1
172.30.42.21	224.0.0.252	17	58158	5355	114	2

Figure 7-4: NetFlow output with decimal values.

As an example, let's say that you have a router with two interfaces where interface 0 is on the outside, facing the Internet, while interface 1 is on the inside. You are using a private address space on the inside of your network—172.20.100.0/24. If you were to see that address coming in on interface 0 while looking at NetFlow data, you would know that it's not a legitimate flow. In fact, the reverse flow would never reach the router so you would only ever see one direction of flow. These sorts of discrepancies can be isolated using NetFlow data.

One of the challenges with NetFlow data is the amount of storage space it can consume if you are running it full time. Even though you are capturing just a summary of the headers and not the entire packet, it can still be space consuming over a period of time. This is especially true if you have a significant number of devices generating NetFlow data and your network is busy. There are trade-offs, of course. As an investigator, you want as much data as you can get your hands on. The network administrative team and the system administrators, however, may have something else to say about this. NetFlow data causes more network traffic running through the network since it needs to be transported from the device generating it to the device storing it. The Collector is another system that needs to be maintained, and if there is a large amount of storage, it is not only a cost to the business but is also a storage device that needs to be maintained and monitored. These are all factors that need to be taken into account when thinking about whether to turn NetFlow on.

It may be possible to turn NetFlow on but then establish a rotation for older NetFlow files so they are either discarded or pushed off to near-line or off-line storage rather than taking up space on a live disk. These are challenging decisions to make, though, because you may have no idea how far

back you need to go to isolate the source of an infection or intrusion. The NetFlow data will help you to do that, but only if you have access to it. This is where having other data in place will be helpful because you may be able to use it as a fallback or, better, to correlate with.

Logging

You may initially think about logging from an operating system perspective, where it is commonly discussed, since operating systems create and store logs. However, it's more complicated than that. As noted in the previous section, it is important to have either data to replace primary sources like NetFlow or data to correlate with primary sources like NetFlow or packet captures during a network investigation. But from a networking perspective, there is a lot more to look at, including not only the operating system and application logs, but also any network devices like routers and switches. They can often have important information when it comes to investigating network incidents. This is where logging comes in.

While logging systems often store the log data on the device where it is generated, larger enterprises may be more likely to store their logs on centralized logging systems. This makes a lot of sense. For a start, trusting logs on any endpoint is probably a mistake. Any device that interacts with users in some way or is exposed to a larger network has the potential to be compromised. Once a system has been compromised, everything on it is suspect, including any log files that may be stored there. An attacker could easily manipulate the log files once he gains the necessary access, which may include administrative privileges. This could include wholesale deletion of the logs or even manipulation of individual log entries.

Centralized log servers also provide a backup to anything stored locally. Administrators may choose to store logs more permanently on central log servers than on the endpoint systems. The log server would be configured with larger storage so the logs on each server could be rotated to leave the disk space there for where it is needed—serving the application that the server was implemented for. The central log server may be considered to be the primary source for log data, even though I referred to it as a backup. This simply means that it's a place to archive logs but since the logs have been moved off of potentially untrustworthy endpoint systems, they may be considered a primary source.

Using a centralized log server also allows log watching to be done in one place rather than taking up processing on every server. With all the logs being forwarded to a single server, that server can become the place where logs are watched for anomalies. This is common in intrusion detection systems, where logs may be monitored for specific events in order to trigger alerts based on those events. There may be some latency involved in the detection because it requires the log entry to be forwarded off the server that generated the log, but in practice that shouldn't be very long, assuming the logs are not batched up for sending, which can slow down delivery. If each entry is forwarded off as it is created, the delay is milliseconds for the log to be sent to the log server and then it is dependent on the log watching service and how regularly it is watching the logs.

A number of systems can do centralized logging, including both commercial and open source offerings. Depending on your needs, you may use a unified offering that can do log gathering as well as monitoring and alerting. You may also choose to use separate applications for the log hosting and the monitoring or alerting.

Syslog

Syslog is an old Unix-based logging system. It was initially developed as part of the mail server Sendmail. Since that time, it has become the predominant way of logging on Unix-like systems. This includes Linux, Solaris, AIX, HP-UX, and macOS, to a degree. There have been other implementations aside from the original syslog, including rsyslog and syslog-ng, but the specification for how syslog functions remains the same. For consistency, most of the references from here will be to the standard or generic implementation: syslog. If there is a reference to a specific implementation, the name of that program will be used. Syslog originally functioned as a de facto standard until it was eventually fully standardized by the Internet Engineering Task Force (IETF).

Syslog specifies different facilities, which indicate what type of event the log entry is. Using these facilities, the syslog server can determine how the log entry is disposed of. Each facility may be sent off to a different log file. The latest request for comment (RFC), document, RFC 5424, specifying how syslog is to function, defines the facilities shown in Table 7-1.

Table 7-1: Syslog Facilities

Facility code	Keyword	Description
0	kern	kernel messages
1	user	user-level messages
2	mail	mail system
3	daemon	system daemons
4	auth	security/authorization messages
5	syslog	messages generated internally by syslogd
6	lpr	line printer subsystem
7	news	network news subsystem
8	uucp	UUCP subsystem
9	-	clock daemon
10	authpriv	security/authorization messages
11	ftp	FTP daemon
12	-	NTP subsystem

Facility code	Keyword	Description
13	-	log audit
14	-	log alert
15	cron	scheduling daemon
16	local0	local use 0 (local0)
17	local1	local use 1 (local1)
18	local2	local use 2 (local2)
19	local3	local use 3 (local3)
20	local4	local use 4 (local4)
21	local5	local use 5 (local5)
22	local6	local use 6 (local6)
23	local7	local use 7 (local7)

In addition to facilities, the syslog standard specifies levels of severity. This helps the applications to generate a log entry for a wide range of events and the system administrator can then determine which levels of those log entries they care about. As an example, a system administrator may only care to write out logs with a severity of Critical or higher. While the system and applications may generate log events for other severities, the syslog server, which is responsible for determining the disposition of each log event, may choose to discard those other log events. The latest specification for syslog, defined in RFC 5424, defines the severities shown in Table 7-2.

Table 7-2: Syslog Severity Levels

Value	Severity	Keyword	Description
0	Emergency	emerg	System is unusable
1	Alert	alert	Should be corrected immediately
2	Critical	crit	Critical conditions
3	Error	err	Error conditions
4	Warning	warning	May indicate that an error will occur if action is not taken
5	Notice	notice	Events that are unusual, but not error conditions
6	Informational	info	Normal operational messages that require no action
7	Debug	debug	Information useful to developers for debugging the application

You will notice that higher severities have lower numbers. The most severe log entry will have a severity level of 0. This may seem counterintuitive unless you are just used to that being the way it is. You might think that a higher number may correspond to a higher severity. Starting at 0 follows a common computer programming numbering scheme and it allows you to begin with what you most care about, with other numbers added later on as desired. The way it is currently defined allows all potential severities to be stored in three bits because the values of severity run from 0 to 7, and those values can all be represented with just three bytes.

A common syslog configuration file will include lines specifying what to do about log events based on facility and severity. In Figure 7-5, you can see a sample syslog configuration. This is from an Ubuntu Linux installation, using the rsyslog package. You should be able to see that entries include both the facility as well as the severity. This is done by specifying the facility followed by a dot (.) and then the severity. In some cases, if you want to indicate that every severity should be included, you would use the asterisk (*).

```
kilroy@quichelorraine: /etc/rsyslog.d$ cat 50-default.conf
# Default rules for rsyslog.
#
# For more information see rsyslog.conf(5) and /etc/rsyslo
g.conf
#
# First some standard log files. Log by facility.
#
auth,authpriv.*          /var/log/auth.log
*.*,auth,authpriv.none   -/var/log/syslog
#cron.*                  /var/log/cron.log
#daemon.*                -/var/log/daemon.log
kern.*                   -/var/log/kern.log
#lpr.*                   -/var/log/lpr.log
mail.*                   -/var/log/mail.log
#user.*                  -/var/log/user.log
#
# Logging for the mail system. Split it up so that
# it is easy to write scripts to parse these files.
#
#mail.info               -/var/log/mail.info
#mail.warn               -/var/log/mail.warn
mail.err                 /var/log/mail.err
#
# Logging for INN news system.
#
news.crit                /var/log/news/news.crit
news.err                 /var/log/news/news.err
news.notice              -/var/log/news/news.notice
#
# Some "catch-all" log files.
#
#*.debug;\
#   auth,authpriv.none;\
#   news.none;mail.none  -/var/log/debug
```

Figure 7-5: Sample syslog configuration file.

Each entry indicates what to do with the log events, typically by specifying a file to write the events out to. If you don't care as much about whether a particular event makes it to the log file, you can add a dash before the filename, as you will see in some cases in Figure 7-5. This tells syslog to just not bother writing the file out to disk after each log event. Instead, it will retain the log entry in memory until the buffer needs to be synced to disk.

As mentioned earlier, you can use one log setup for centralized logging. A syslog server can be used as a centralized log server as well as having the syslog server forward log entries off to a centralized log server. Syslog was initially designed to use UDP as the transport protocol, but since UDP doesn't have guaranteed delivery, you may want to make use of TCP. The rsyslog server allows you to use either TCP or UDP. The configuration setting for rsyslog to act as a syslog server is shown in Listing 7-1.

Listing 7-1: Rsyslog Configuration for Listening

```
# provides UDP syslog reception
#module(load="imudp")
#input(type="imudp" port="514")

# provides TCP syslog reception
#module(load="imtcp")
#input(type="imtcp" port="514")
```

In addition to listening, rsyslog can be configured to forward log entries off to another server. This is a different set of configuration entries and, just as with listening, you can forward with both TCP and UDP. The entries in Listing 7-2 show how you would forward syslog entries off to a remote server. There are two entries here for both TCP and UDP. Commonly, syslog uses the well-known port 514 but it is possible to use a different port. You can see that in the first configuration file entry where the port 10514 is used instead of the default port. The first line also uses TCP, indicated by the @@ before the IP address. In order to forward using UDP, you would use the second example with a single @. You will also see *.* , which mirrors the way the log file entries look. What this means is every criticality and every severity will be forwarded off to the central log host. It is then up to the log host to determine what to do with those entries, based on its configuration file.

Listing 7-2: Forwarding syslog Entries

```
# This uses TCP and a non-default port to forward to
# the port specified must match where the server is listening
*.* @@192.168.86.34:10514
# This uses UDP for the transport protocol and the default port
*.* @192.168.85.10
```

Once you have a syslog server established, you will then have log entries—you may have several log files to look at, depending on what your syslog configuration looks like. In Figure 7-6, you can see a sample of one of the logs. This particular log is the auth.log that stores information about authentication events. You can see entries for sudo, which is a way for a user to gain temporary superuser or administrative privileges. Additionally, there are entries for logins to an FTP server, pure-ftpd. Logs from services like this one can be used to correlate the NetFlow data we looked at before.

```

Jan 7 11:19:29 quichelorraine sudo: pam_unix(sudo:session): session opened for
user root by kilroy(uid=0)
Jan 7 11:19:30 quichelorraine sudo: pam_unix(sudo:session): session closed for
user root
Jan 7 11:20:03 quichelorraine sudo: kilroy : TTY=pts/0 ; %WD=/etc/rsyslog.d ;
USER=root ; COMMAND=/usr/bin/apt-get install pure-ftpd
Jan 7 11:20:03 quichelorraine sudo: pam_unix(sudo:session): session opened for
user root by kilroy(uid=0)
Jan 7 11:20:09 quichelorraine sudo: pam_unix(sudo:session): session closed for
user root
Jan 7 11:20:27 quichelorraine sudo: kilroy : TTY=pts/0 ; %WD=/etc/rsyslog.d ;
USER=root ; COMMAND=/etc/init.d/pure-ftpd start
Jan 7 11:20:27 quichelorraine sudo: pam_unix(sudo:session): session opened for
user root by kilroy(uid=0)
Jan 7 11:20:27 quichelorraine sudo: pam_unix(sudo:session): session closed for
user root
Jan 7 11:20:45 quichelorraine pure-ftpd: pam_unix(pure-ftpd:session): session o
pened for user kilroy by (uid=0)
Jan 7 11:20:45 quichelorraine systemd-logind[4438]: New session c1 of user kilr
oy.
Jan 7 11:20:45 quichelorraine pure-ftpd: pam_unix(pure-ftpd:session): session c
losed for user kilroy
Jan 7 11:20:54 quichelorraine systemd-logind[4438]: Removed session c1.
Jan 7 11:21:06 quichelorraine pure-ftpd: pam_unix(pure-ftpd:session): session o
pened for user kilroy by (uid=0)
Jan 7 11:21:06 quichelorraine systemd-logind[4438]: New session c2 of user kilr
oy.
Jan 7 11:21:06 quichelorraine pure-ftpd: pam_unix(pure-ftpd:session): session c
losed for user kilroy
Jan 7 11:21:13 quichelorraine systemd-logind[4438]: Removed session c2.

```

Figure 7-6: syslog entries.

Each entry includes a date and time stamp indicating when the event was generated. This is followed by the name of the system that generated the entry, quichelorraine in our case. After that is the name of the program that created the entry, followed by the message that was created by the application. Up through the name of the program is all created by the syslog server. The message originates with the application. Though the date and time stamp, the system name, and the name of the program are all going to be accurate based on the system configuration, the message from the application is only going to be as good as the programmer who wrote the program. If the programmer(s) was less inclined to be specific, the message won't be very useful. Good programmers recognize the importance of being clear and specific in log messages to help with troubleshooting, though, so most programs writing to syslog will generate useful entries.

Each syslog entry may only be useful if you understand the application that created it, however. As an example, if you didn't know what sudo was, you may not understand the entry about a session being opened. In this case, what that means is the user used sudo to gain temporary elevated permissions. Just having the logs alone may not be as helpful. In addition to having the logs, it may be necessary to have an understanding of the different applications that are logging.

Different services may generate their own logs and they may be stored in the same location as the logs syslog creates. As an example, the Apache web server will commonly store its logs in `/var/log` in a directory specific to Apache. On the Ubuntu system we have been looking at, the Apache logs are in `/var/log/apache2`, but on other systems, they may be in a directory named `/var/log/httpd`. Because it's the Apache server that writes these logs out, it's necessary to look at the Apache configuration settings to determine where the logs for that service are located. These logs can be

essential to correlate with any network information that is gathered, whether it's packet capture data or NetFlow data.

You would use syslog on Linux systems and other Unix-like systems, and it's possible to forward logging events on a Windows system to a syslog server. On Windows systems, however, you would primarily use the Windows Event Log to look for Windows logs.

Windows Event Logs

The Windows Event Log has been around since Windows NT was released in 1993. Since then, the Windows Event Logs have gone through different iterations. In the most recent iterations of Event Logs, Microsoft has moved toward structured XML for storing log information on disk. Most of the time, users and administrators will use the Event Viewer, as seen in Figure 7-7. One advantage of the Event Viewer is that it collects everything into one interface. As you can see, Windows has a lot of Application logs in addition to the Windows logs. You can see this on the left-hand side of the interface. If you expand the Applications and Services section, you'll see that each vendor has its own folder for logs. Under the Microsoft folder, each system service has its own place to store logs. This is an extensive list, so having one place to look at all of them without having to wade through a single file for specific applications is helpful.

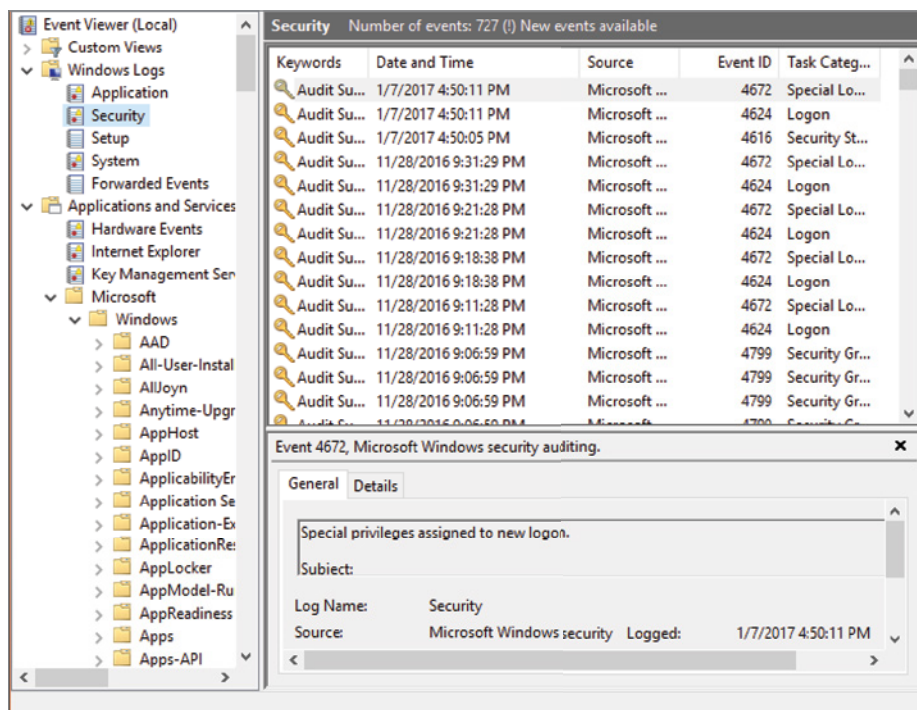


Figure 7-7: Windows Event Viewer.

Since the underlying storage of the events is not just plaintext lines as in syslog, but instead in an XML format, using the Event Viewer will help you extract all of the relevant information and present it in a meaningful way so it's easier to quickly parse the event. You don't have to read through the entire line, as you would with syslog. Instead, you can just look at the section of the event that most interests you. As an example, in Figure 7-8, you can see a single event selected in the top from the Security log. This is an audit event indicating that special privileges have been assigned to a login. At the bottom of the interface, each part of the event is broken out so you can look directly at the section you are most interested in. You may want to find the Event ID so you can look it up with Microsoft on their TechNet pages to learn more about the event. You can find that in the bottom section with its own header. This ability to quickly parse an event can be helpful.

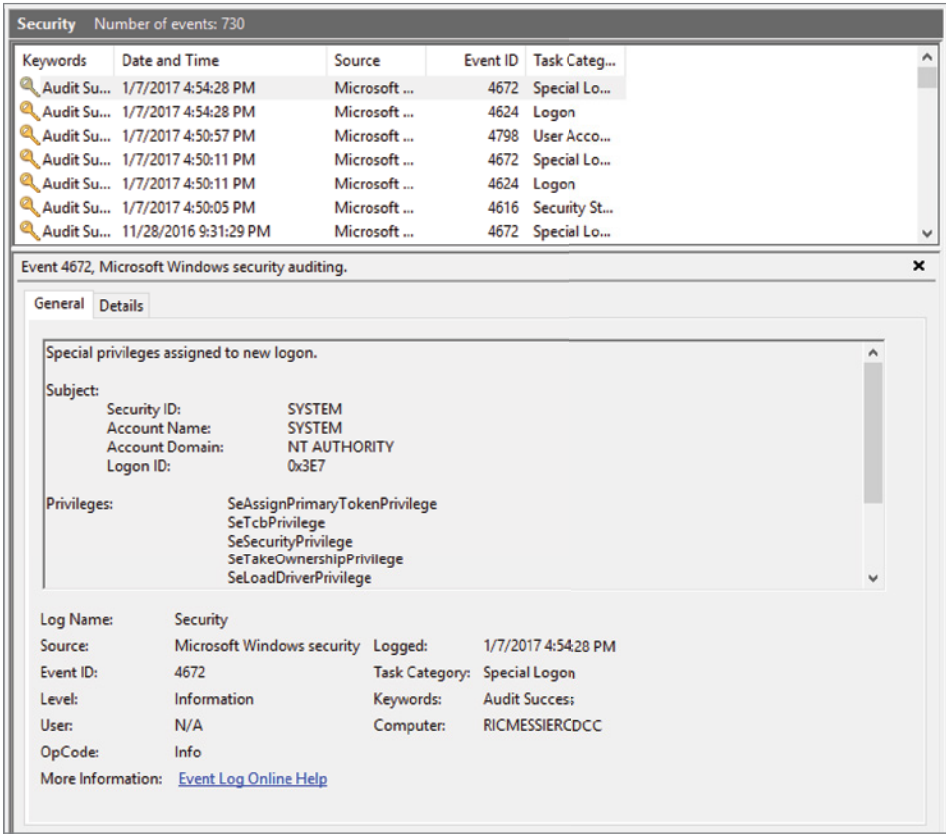
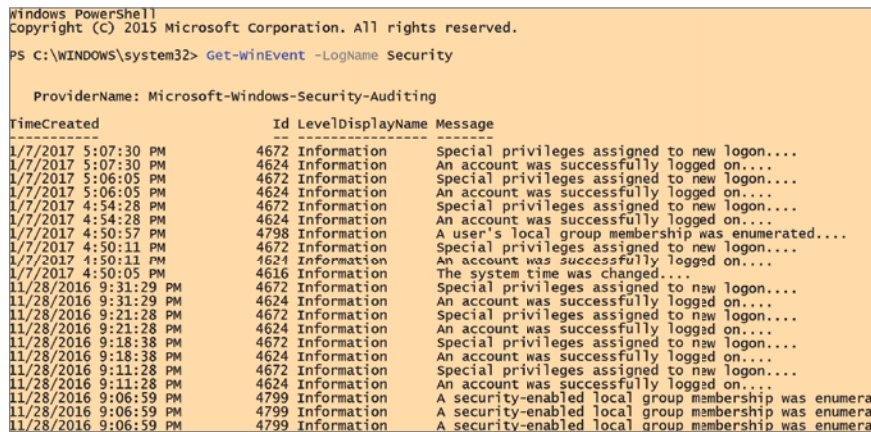


Figure 7-8: A Windows event.

While the Event Viewer will also allow you to filter and search, there are other ways to look at the Event Logs on Windows. One method is to use the PowerShell, which implements cmdlets to

perform functions. One cmdlet that can be used to get access to the logs is Get-WinEvent. Using this cmdlet, you can get text-based representations of the events. You can see an example of its use in Figure 7-9. You need to specify which log you want to look at and in this case, we are looking at the Security log, just as we did earlier. You don't get the detail here as you did in the Event Viewer, but you can get a complete look at all of the events. Get-WinEvent has a number of other parameters that can be used to get more detail and change the way the data is extracted. For example, you may only be interested in 50 events. If that's the case, you would add -MaxEvents. You can also specify Xpath filters to get a limited view of the events.



```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\WINDOWS\system32> Get-WinEvent -LogName Security

ProviderName: Microsoft-Windows-Security-Auditing
-----
TimeCreated          Id LevelDisplayName Message
-----
1/7/2017 5:07:30 PM 4672 Information Special privileges assigned to new logon....
1/7/2017 5:07:30 PM 4624 Information An account was successfully logged on....
1/7/2017 5:06:05 PM 4672 Information Special privileges assigned to new logon....
1/7/2017 5:06:05 PM 4624 Information An account was successfully logged on....
1/7/2017 4:54:28 PM 4672 Information Special privileges assigned to new logon....
1/7/2017 4:54:28 PM 4624 Information An account was successfully logged on....
1/7/2017 4:50:57 PM 4798 Information A user's local group membership was enumerated....
1/7/2017 4:50:11 PM 4672 Information Special privileges assigned to new logon....
1/7/2017 4:50:11 PM 4624 Information An account was successfully logged on....
1/7/2017 4:50:05 PM 4616 Information The system time was changed....
11/28/2016 9:31:29 PM 4672 Information Special privileges assigned to new logon....
11/28/2016 9:31:29 PM 4624 Information An account was successfully logged on....
11/28/2016 9:21:28 PM 4672 Information Special privileges assigned to new logon....
11/28/2016 9:21:28 PM 4624 Information An account was successfully logged on....
11/28/2016 9:18:38 PM 4672 Information Special privileges assigned to new logon....
11/28/2016 9:18:38 PM 4624 Information An account was successfully logged on....
11/28/2016 9:11:28 PM 4672 Information Special privileges assigned to new logon....
11/28/2016 9:11:28 PM 4624 Information An account was successfully logged on....
11/28/2016 9:06:59 PM 4799 Information A security-enabled local group membership was enumera
11/28/2016 9:06:59 PM 4799 Information A security-enabled local group membership was enumera
```

Figure 7-9: Using PowerShell for Windows events.

As with syslog, there are reasons you may want to look at the Windows Event Logs as either a starting point for a network forensic investigation or a way to correlate on the operating system side what was seen in the network. While the network never lies, in the sense that what you see on the wire is exactly what was sent on the wire, you may not have the entire picture without correlating with the operating system.

Firewall Logs

Firewall logs are another important tool when it comes to looking into network incidents. A number of different types of firewalls exist, of course, and we're going to look at a couple of them. Regardless of what firewall you may have access to, firewalls will commonly log what they are doing. This may include how the firewall is operating or the logs may relate to specific rules that have been configured. As an example, the logs shown in Listing 7-3 resulted from an iptables rule on a Linux system indicating that all communications to port 80 be logged.

Listing 7-3: iptables Log Entries

```

Jan  7 14:13:22 quichelorraine kernel: [12167.418837] IN=ens33 OUT=
MAC=00:0c:29:bd:3d:3e:f4:5c:89:b7:2c:89:08:00 SRC=192.168.86.21
DST=192.168.86.34 LEN=52 TOS=0x00 PREC=0x00 TTL=64 ID=44256 DF
PROTO=TCP SPT=53542 DPT=80 WINDOW=4096 RES=0x00 ACK URG=0
Jan  7 14:14:07 quichelorraine kernel: [12212.670359] IN=ens33 OUT=
MAC=00:0c:29:bd:3d:3e:f4:5c:89:b7:2c:89:08:00 SRC=192.168.86.21
DST=192.168.86.34 LEN=40 TOS=0x00 PREC=0x00 TTL=64 ID=65033
PROTO=TCP SPT=53542 DPT=80 WINDOW=4096 RES=0x00 ACK URG=0
Jan  7 14:14:53 quichelorraine kernel: [12258.205046] IN=ens33 OUT=
MAC=00:0c:29:bd:3d:3e:f4:5c:89:b7:2c:89:08:00 SRC=192.168.86.21
DST=192.168.86.34 LEN=40 TOS=0x00 PREC=0x00 TTL=64 ID=6319
PROTO=TCP SPT=53542 DPT=80 WINDOW=4096 RES=0x00 ACK URG=0
Jan  7 14:16:29 quichelorraine kernel: [12354.182134] IN=ens33 OUT=
MAC=00:0c:29:bd:3d:3e:f4:5c:89:b7:2c:89:08:00 SRC=192.168.86.21
DST=192.168.86.34 LEN=64 TOS=0x00 PREC=0x00 TTL=64 ID=27954 DF
PROTO=TCP SPT=53625 DPT=80 WINDOW=65535 RES=0x00 SYN URG=0
Jan  7 14:16:29 quichelorraine kernel: [12354.186483] IN=ens33 OUT=
MAC=00:0c:29:bd:3d:3e:f4:5c:89:b7:2c:89:08:00 SRC=192.168.86.21
DST=192.168.86.34 LEN=52 TOS=0x00 PREC=0x00 TTL=64 ID=28090 DF
PROTO=TCP SPT=53625 DPT=80 WINDOW=4117 RES=0x00 ACK URG=0
Jan  7 14:17:00 quichelorraine kernel: [12385.337712] IN=ens33 OUT=
MAC=00:0c:29:bd:3d:3e:f4:5c:89:b7:2c:89:08:00 SRC=192.168.86.21
DST=192.168.86.34 LEN=52 TOS=0x00 PREC=0x00 TTL=64 ID=23849 DF
PROTO=TCP SPT=53625 DPT=80 WINDOW=4117 RES=0x00 ACK URG=0
Jan  7 14:17:13 quichelorraine kernel: [12398.025122] IN=ens33 OUT=
MAC=00:0c:29:bd:3d:3e:f4:5c:89:b7:2c:89:08:00 SRC=192.168.86.21
DST=192.168.86.34 LEN=52 TOS=0x00 PREC=0x00 TTL=64 ID=56388 DF
PROTO=TCP SPT=53625 DPT=80 WINDOW=4117 RES=0x00 ACK FIN URG=0
Jan  7 14:17:13 quichelorraine kernel: [12398.029197] IN=ens33 OUT=
MAC=00:0c:29:bd:3d:3e:f4:5c:89:b7:2c:89:08:00 SRC=192.168.86.21
DST=192.168.86.34 LEN=52 TOS=0x00 PREC=0x00 TTL=64 ID=64227 DF
PROTO=TCP SPT=53625 DPT=80 WINDOW=4117 RES=0x00 ACK URG=0

```

While these logs may look obtuse and arcane at first glance, they are easier to understand if you carefully review them. Each entry here indicates the IN and OUT interfaces, which would be relevant if this host were configured as a network firewall, forwarding packets from one interface to another, rather than just an individual Linux host with iptables running on it. Additionally, you can see the MAC address, the SRC and DST (source and destination) IP addresses, as well as statistics related to the packet such as the length (LEN) and the type of service (TOS). You can also see the source and destination ports (SPT and DPT) Using this knowledge, the log entries should make much more sense.

Different firewalls will generate different looking logs, of course. You may also get different statistics from different firewalls. As an example, the PFSense firewall, based on FreeBSD, will show

you the state of network connections. Providing a connection state may be a common thing across stateful firewalls, which have to pay attention to the state of each connection in order to determine whether or not connections are allowed. This means that the firewall knows whether the message is new—it hasn't been seen before.

A SYN message, the first message in a TCP three-way handshake, is recognized as a *new* message by a stateful firewall. An *established* connection is one that has been seen and been previously allowed. This means the firewall is aware that there has been previous communication and there is an entry in the state table. A *related* connection is one that is associated to but not directly part of an established connection. This may be something like an FTP transmission where a second connection is established from the server back to the client to transmit files.

In Figure 7-10, you can see a section of the state table from the PFSense firewall. This shows the different flows, meaning the connections from one system to another, as well as the state they are in. Additionally, you can see the number of bytes that have been transmitted over the course of that communication stream. The state table will also indicate which interfaces the communication was seen on. In most cases in this screen capture, the communication is over the WAN port, which is the external interface as far as the firewall is concerned. The LAN port is the one on the inside of the network.

No matter which firewall you are using, having the ability to look at the logs will provide you with some different points of correlation to events seen on the network. Additionally, you may see events that you wouldn't see in other places. The firewall may log packets that were dropped. This is not something you would see on the network because they would just not be responded to. You may assume that the packet was dropped by the firewall, but with the firewall log to correlate with other evidence, you may know for sure.

This is not to say that all firewalls will log dropped messages. The iptables firewall, included with Linux, does not automatically log dropped packets, for instance. The person writing the rule would have to either set a policy to log all dropped messages or set a log and a drop message for everything that was being dropped. Without the overall policy, there may be some messages that don't get logged when they are dropped. This is an area where some guidance can be provided to firewall administrators. The firewall can provide an important point of insight to an investigator, especially when it comes to messages that have been identified as ones that shouldn't be seen in the network.

One thing to keep in mind is that firewalls are commonly located on the edges of networks, but your idea of an edge may be different from a network administrator's. An easy place to see an edge is anywhere the enterprise network touches a service provider. This is a boundary point between two networks so it's clearly an edge. However, to a network administrator, the data center may be its own network and any connection to the outside world, even if it's to the internal business network, is an edge. This is because the data center is commonly where the most sensitive information is stored. As a result, there may be

a firewall between the data center networks and every other network in the business. Another place where you may find a firewall is the connection between a WiFi network and the rest of the internal network. The primary point here is that you may find firewalls turning up interesting information in an area you may consider the inside of the network. Different areas of an enterprise network may have different security needs and thus some will have more restrictive rules applied than others.

Diagnostics / States / States						
States Reset States State Filter Interface all Filter expression Simple filter such as 192.168, v6, icmp or ESTABLISHED Filter						
States						
Interface	Protocol	Source (Original Source) -> Destination (Original Destination)		State	Packets	Bytes
WAN	ipv6-icmp	fe80::20c:29ff:fe2c:f00a[60581] -> fe80::46e1:37ff:fe4:f234[60581]		NO_TRAFFIC:NO_TRAFFIC	483 / 89	23 KiB / 7 KiB
WAN	icmp	172.30.42.38:60435 -> 172.30.42.1:60435		0:0	483 / 23	13 KiB / 644 B
WAN	ipv6-icmp	fe80::20c:29ff:fe2c:f00a -> ff02::1:fff4:f234		NO_TRAFFIC:NO_TRAFFIC	178 / 0	13 KiB / 0 B
LAN	tcp	192.168.139.10:60610 -> 192.168.139.1:80		ESTABLISHED:ESTABLISHED	291 / 622	31 KiB / 834 KiB
LAN	tcp	192.168.139.10:60612 -> 192.168.139.1:80		FIN_WAIT_2:FIN_WAIT_2	14 / 14	1 KiB / 4 KiB
WAN	udp	2601:280:5000:43c0:20c:29ff:fe2c:f00a[52768] -> 2001:7fd::1[53]		SINGLE:NO_TRAFFIC	1 / 0	76 B / 0 B
WAN	udp	2601:280:5000:43c0:20c:29ff:fe2c:f00a[53923] -> 2001:7fd::1[53]		SINGLE:NO_TRAFFIC	1 / 0	76 B / 0 B
WAN	udp	2601:280:5000:43c0:20c:29ff:fe2c:f00a[11344] -> 2001:503:c27::2:30[53]		SINGLE:NO_TRAFFIC	1 / 0	76 B / 0 B
WAN	udp	2601:280:5000:43c0:20c:29ff:fe2c:f00a[47831] -> 2001:503:c27::2:30[53]		SINGLE:NO_TRAFFIC	1 / 0	76 B / 0 B
lo0	udp	fe80::20c:29ff:fe2c:f00a[546] -> ff02::1:2[547]		NO_TRAFFIC:SINGLE	1 / 0	100 B / 0 B
WAN	udp	fe80::20c:29ff:fe2c:f00a[546] -> ff02::1:2[547]		SINGLE:NO_TRAFFIC	1 / 0	100 B / 0 B
WAN	udp	172.30.42.38:42450 -> 193.0.14.129:53		SINGLE:NO_TRAFFIC	1 / 0	56 B / 0 B
WAN	udp	172.30.42.38:22741 -> 193.0.14.129:53		SINGLE:NO_TRAFFIC	1 / 0	56 B / 0 B

Figure 7-10: PFSense firewall state table.

Just as with other logs, firewall logs take up disk space. With large networks, firewalls can generate a lot of log data. This may be one reason why not all dropped packets get logged. Since there are adversaries everywhere, there are often port scans and ping sweeps happening on a regular basis. Logging all of this activity when it is dropped can consume valuable resources. As a result, it's important to consider the trade-offs. How much visibility do you want with regard to network information provided by the firewall logs as compared with how much it will cost in disk space and processing by the firewall and log monitor?

Router and Switch Logs

Routers and switches are capable of generating logs, not surprisingly. There are a number of reasons why you may care about the logs these devices can generate. For a start, the logs will indicate who has accessed the administrative interfaces for these devices. You will be able to see who has been doing work on the device and when they were doing it. This may include failed attempts to access the device, which could provide an indication that someone is trying to break into the administrative interface. You may also get accounting information associated with configuration changes. This may be helpful to understand why the network is behaving differently than it had previously. A number of administrative logs can be helpful in uncovering this information.

It is important to keep in mind that routers and switches have multiple interfaces, by design. There are interfaces where the business of the device takes place—whether it's an Ethernet or serial interface for the router to forward packets from one interface to another or whether it's the Ethernet interface each system on the network plugs into on a switch—and there are administrative interfaces. Typically, the administrative functions happen on a separate interface. Switch ports, for example, don't have an IP address so they can't be directly addressed anyway. However, there is a separate interface where an administrator could connect using SSH or HTTP to perform management functions.

Beyond the administrative traffic and logs, however, routers will often have the ability to perform rudimentary access control using access control lists (ACLs). An ACL is a very basic type of firewall, in the sense that it can cause the firewall to drop or allow packets as they come through. Whereas a higher level firewall can make more complex decisions, a standard ACL commonly looks at source and destination addresses and port numbers, though there are extended ACLs that can provide more functionality. Packets an ACL drops may get logged and these logs are worth taking a look at, just as firewall logs are.

On the switch side, since it operates at Layer 2, there are no IP addresses or ports to make decisions on. The best that can happen is to assign a MAC address to a particular port. If a device is plugged into a port with the wrong MAC address, that may get logged and that would be of interest, potentially, because it may demonstrate someone trying to connect a rogue device to the network.

Additionally, switches will commonly keep track of traffic passed across different interfaces. Routers will do the same thing. Looking at statistics provided by these network devices can be helpful in detecting unexpected activity on the network.

Log Servers and Monitors

Earlier, we talked about using a syslog server as a centralized log host, but there are other log servers you can use. An advantage of using a log management package is the other functionality that you can get. In addition to just collecting logs and storing them, packages like Nagios or Splunk will allow you to more completely manage the logs by helping with search and analysis. Just collecting logs and allowing them to sit on a disk isn't particularly helpful to an organization, even if having the logs later on will help you as a forensic investigator. If the logs are just collected and allowed to sit, you may never know there is something to look at. This attitude toward logging may be a cause of organizations having suffered breaches without being aware for years. The evidence could be in the logs and they simply aren't doing any analysis or monitoring.

Nagios began life in the late 1990s as a monitoring solution designed to run on Linux. Currently Nagios has commercial offerings as well as an open source offering that can be installed. Part of the functionality offered by Nagios is being a log management solution. Nagios, like so many other programs these days, uses a web interface for a console. You can see the Nagios console in Figure 7-11. Configuring Nagios for monitoring and log management requires manipulating the configuration files on disk. You can add in multiple services to manage and it supports monitoring multiple types of systems and services.

Another popular solution for log management and monitoring is Splunk. Splunk has commercial offerings but it also offers a light version for free if you want to try it out to see how it operates. Splunk can be configured to look at logs on a local system where the Splunk server is running. The configuration page on the web interface is shown in Figure 7-12. You can also configure it to be a syslog listener for other systems to send log data to. On the left-hand side of the interface shown in Figure 7-12, you can see TCP/UDP. Selecting enables you to configure Splunk to be a listener for syslog data.

Once you have data configured, Splunk indexes the data and enables you to search all of your data sources for patterns. You can also set up alerts, based on information you care about looking at. In fact, Splunk will analyze the logs and locate patterns for you. You can see this in Figure 7-13. On the Ubuntu system Splunk was installed on, Splunk looked through more than 15,000 events to isolate 100 patterns, which are shown. You will see a percentage on the left-hand side of the display. This indicates the percentage of the total events where this particular log entry shows up. Selecting the event will provide a total number of instances, the ability to view the events, and a chance to create an alert based on this particular event.

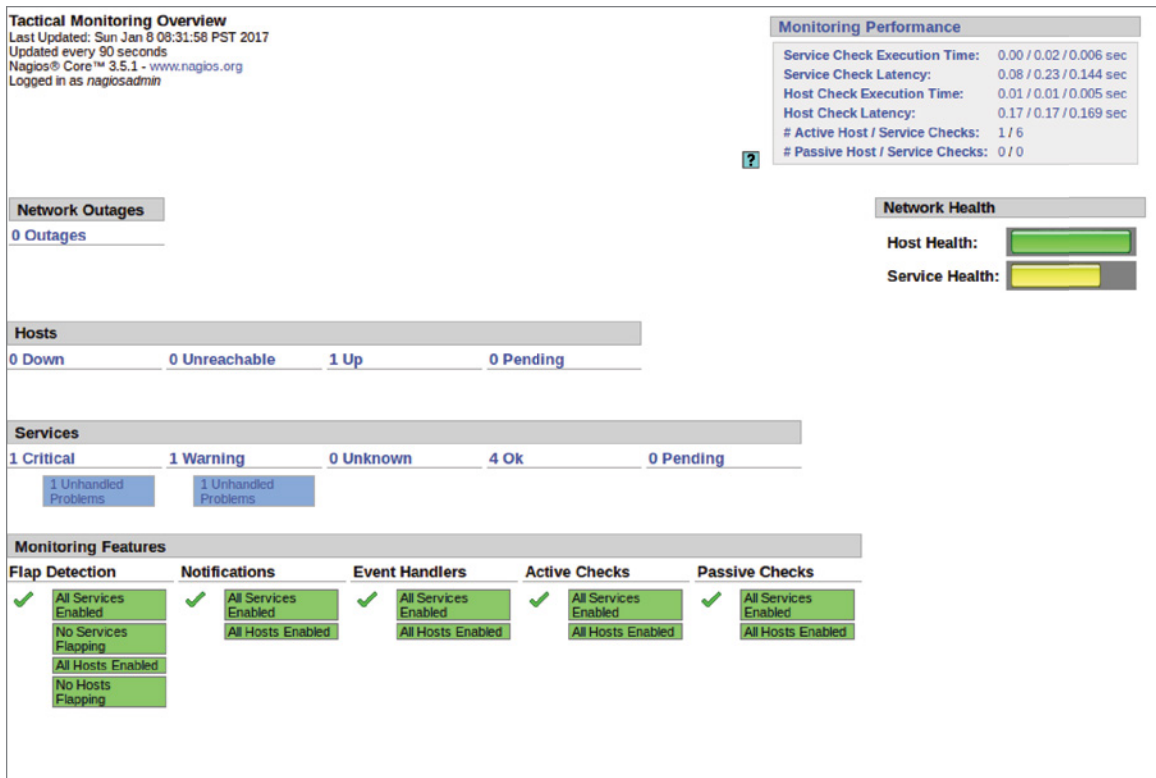


Figure 7-11: Nagios monitoring console interface.

Add Data

Select Source Set Source Type Input Settings Review Done

Files & Directories
 Upload a file, index a local file, or monitor an entire directory.

HTTP Event Collector
 Configure tokens that clients can use to send data over HTTP or HTTPS.

TCP / UDP
 Configure Splunk to listen on a network port.

Scripts
 Get data from any API, service, or database with a script.

Configure this instance to monitor files and directories for data. To monitor all objects in a directory, select the directory. Splunk monitors and assigns a single source type to all objects within the directory. This might cause problems if there are different object types or data sources in the directory. To assign multiple source types to objects in the same directory, configure individual data inputs for those objects. [Learn More](#)

File or Directory?

On Windows: c:\apache\apache.error.log or \\hostname\apache\apache.error.log. On Unix: /var/log or /mnt/www01/var/log.

Whitelist?

Blacklist?

Figure 7-12: Splunk data configuration.

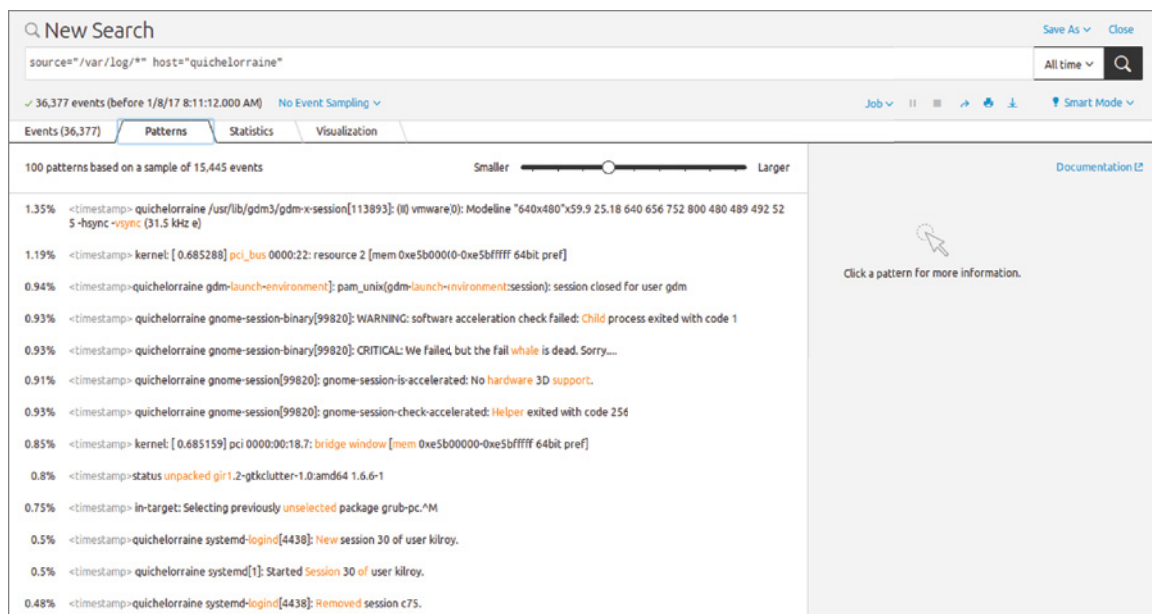


Figure 7-13: Splunk patterns.

Although you can perform analysis on large data sets without acquiring commercial tools, using a tool like Splunk or another log management or analysis program can make life considerably easier. This is even truer when you have more systems. In this case, 15,000 events were generated over the course of less than 24 hours. Not long after taking the screen capture showing 15,000 events, the number of events had risen to 18,000 and that's only on a single host. You can see how having a program to do analysis and management of logs for you could be very helpful. This is not to say that you should rush out and get Splunk; a number of log management systems are available, and based on your needs and budget you may find a different one to be more suitable.

Antivirus

Antivirus programs can be a useful source of information during an analysis. The logs should be able to indicate not only if and when a file has been isolated as potentially problematic, but also when the definitions have been updated. As you may know, antivirus programs rely on definitions of known malware in order to be able to identify a file as a virus. If the program doesn't have a definition, it can't identify the virus. This is one way systems can still be infected in spite of running antivirus, even if the definitions are regularly updated. Until someone has created a definition, the virus is going to go undetected. Knowing when the definitions have been updated, as well as which malware has been added in each definition set, can help to get a timeline that may aid in your investigation.

Some enterprise systems may not allow users the control over what happens when a file has been isolated and removed. However, if the specific implementation you are looking at does allow that control, you may be able to see when users have removed files from quarantine. Again, this can help you with timeline analysis and it may also provide you with filenames to investigate further.

Keep in mind that as you look at enterprise antivirus programs, you may need to look in different places for the log files. They may be stored on the local systems in text files or they may be in the Windows Event system. The location depends entirely on the package used and the way it handles logging. Some packages, if they are multi-platform, may just write logs to disk directly so they don't have to rewrite the logging portion of the software for each platform they operate on. Writing to disk will be the same across all platforms, so it's easy to do once and include on all implementations. You may also find that logs are pushed back to an enterprise console, so they may be stored on a server. Again, you may need to look for them in different places on the server. It is entirely dependent on the implementation of the antivirus software.

Incident Response Preparation

When an incident occurs you will be much happier if you have systems and capabilities in place ahead of time. On the business side, there should be a policy in place that creates an incident response team (IRT). The IRT should make collective decisions regarding how they are going to prepare themselves. This may include making recommendations for different capabilities in the network like enabling NetFlow or having a log management system in place. As with anything else, these will need to be justified from a cost perspective. This requires understanding the potential for an incident to occur as well as the types of incidents that may occur.

This is not a place to be Chicken Little. It's important to be reasonable, rational, and objective when creating potential scenarios. If you go into this saying the world is coming to an end you aren't likely to be taken seriously. You need to be able to really understand the business, what the threats are, and their potential for manifestation. This will help with scale so you aren't asking for the largest possible solution just because it will make your job easier. You need to accept that you won't get everything you would really like to have just because it may one day make your job easier. Resources will always be limited. If you are not already a part of the incident response team, working closely with the incident response team on developing a plan that is both workable and reasonable is important.

This is another place where thinking ahead and some software may be beneficial. When you have a virus outbreak, for instance, there will be an impact on hosts. In the end, many attacks that take place on the network have an impact on the hosts. They will not always be just distributed denial of service (DDoS) attacks that your service provider will have to handle, because nothing you are able to do has any impact other than to further flood your network connection sending out RST or other network messages.

You may need to be able to pull data back to a central place quickly to correlate what you are seeing on the network. Software packages can help with that. In fact, a number of software packages are

available that can help with incident response and forensic investigations. Log management solutions will be covered in another chapter. While much of this is done on the host, it is useful to be able to correlate your network traffic with what is happening on the host. Some of these software packages will not only look at host artifacts like what is on the disk, but they will also look at memory. In the case of a network attack, particularly when it comes to a malware infestation and communication with a command and control server on the Internet, memory analysis can be essential.

Google Rapid Response

Google Rapid Response (GRR) is an open source offering from Google that helps to give you quick access to systems on the network in the case of an incident. GRR uses a client/server model where you install the server somewhere and employ a web interface to use it. Then, you install clients on the endpoints you have control over. The endpoints communicate with the server so you can quickly gather information from them. When something happens, you will want to pull data and do some analysis. Using the Hunt feature, you can search for specific types of data from your clients. You can see a sample of some of the things GRR can hunt for in Figure 7-14.

As noted at the end of the previous section, investigating memory can be important during an incident. In some types of attacks, it is essential. GRR makes use of Rekall, which is a memory analysis tool forked from Volatility. Rekall can extract information from memory dumps, assuming it has an understanding of how the memory is laid out for that particular operating system. Different operating systems, including different versions of operating systems, have different memory layouts when it comes to locating data structures within the kernel that point to where processes are in memory as well as other information that the kernel keeps track of, like network connections.

Of course, GRR can also look at files on the clients, though that would fall under the category of operating system forensics. It is hard to peel apart network forensics and operating system forensics sometimes, though, since what happens on the network doesn't stay on the network. In many cases, you are using the network to corroborate what you are seeing on the host, or you are using the host to corroborate what you are seeing on the network. A tool like GRR can help to corroborate your other, more purely network-based artifacts.

Commercial Offerings

GRR certainly isn't alone when it comes to incident response software. One of the early commercial offerings in this space was Carbon Black. Carbon Black also offered a client/server model where the server is used for clients to communicate with. The server functions as a console for an analyst to communicate with in order to interact with the clients. Using the console, an analyst can extract necessary data when an incident occurs. This is similar to the way GRR works.

As the need for endpoint protection has become more obvious, other companies have gotten into the space and the ones already there, like Carbon Black, are moving away from strictly focusing on incident response prep and more toward detection and response. This response can include prevention

or protection and less strictly from an investigatory perspective. Walking through the vendor booths at a recent BlackHat event, it became clear that many companies that didn't previously have any offerings in the investigatory space are suddenly interested there. Other companies, like CrowdStrike, are newer and are involved in incident response, investigation, and endpoint protection. CrowdStrike uses a Software as a Service (SaaS) cloud-based model for its service offering, which is a little different from others in the space.

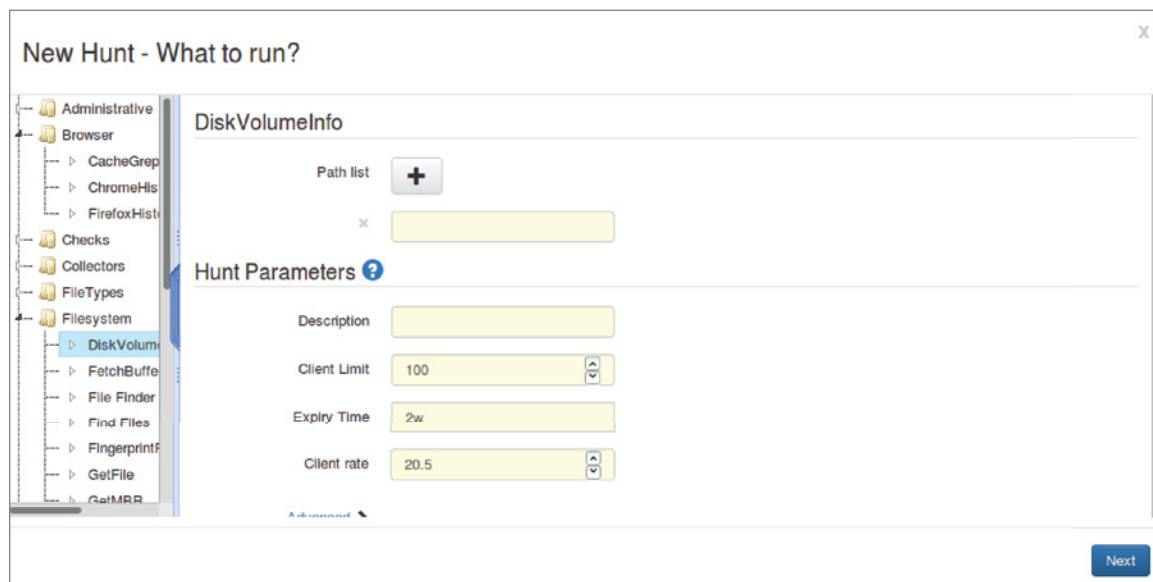


Figure 7-14: GRR hunt.

Even venerable forensic software vendors like Guidance Software, makers of EnCase, are getting in on the incident response action. In addition to EnCase, which is good at a more traditional dead box forensics investigation, Guidance Software, along with other companies that have forensic software, have offerings that are moving more toward the incident response model, recognizing that more and more, live analysis, including detection and perhaps prevention, is necessary given the adversaries companies are facing. Of course, a company like Guidance Software and its multiple offerings can integrate across different type of investigations if you are using all of its solutions.

Security Information and Event Management

One type of software that you may see more and more of in the enterprise space is for security information and event management (SIEM). This is considered an intelligence platform that merges some of what we talked about earlier with respect to log management along with an alerting or event

management platform. In this way, an operations team can get access to all of the information they need as well as manage the event from start to finish. You don't have to create events (tickets) any longer in one system while taking information from another. The idea of a SIEM is to put all of the intelligence and the workflow into a single system. This hopefully prevents the need to stare at multiple screens simultaneously or take your eyes off one screen in order to work in another as you enter notes about an event. Everything comes together in one unified system. This can help to minimize blind spots by not having data scattered across multiple repositories, viewed by multiple groups. Additionally, pulling everything together helps with correlation since there could be multiple events happening on multiple systems that are all related to the same incident. Pulling it all together helps to reduce time spent by multiple groups working the same issue.

An advantage to this type of system is the ability to perform correlation across multiple sources of information. Events are not always isolated, meaning they don't always originate in the network and stay there. They also don't originate on a single host and stay there. There are generally multiple sources of information, as we have discussed. The advantage of a SIEM is that you can provide all of your data sources to the SIEM and it can handle correlation across multiple data points. This does generally mean, though, that you have to understand your network and data points well enough to be able to generate rules to allow the SIEM to create alerts or events that you can follow up on.

The danger of a system like this is simply information overload, though that danger is not at all limited to a SIEM. However, if you have all of your information sources in a single system, there may be either a desire to create a lot of rules to look for potential problems or there may be a lack of understanding of how everything comes together, such that there may not be enough or even the right rules in place to be able to see what is happening. As always, it comes down to defining what you care about so you can only look at those things without needing to look at everything else.

Used correctly, a SIEM provides an extraordinary amount of power by combining disparate data sources from your hosts and your network devices, including intrusion detection systems as well as routers, switches, and firewalls. However, as suggested before, just because you can look at something doesn't mean that you should be spending your time there. It's tempting to alert on everything and follow up on it, but a much smarter and more efficient approach is to identify what you care about so you can pay attention to just that. A SIEM offers the potential to weed out the things you don't care about while still retaining that information in case you need it later on.

As with the log management solutions, a SIEM needs to act as a log server in order to consume all of the different data sources. This means that it can act as a syslog server and you can point all of your Unix-like systems at it. There may be clients that get installed to collect other types of information in addition to being able to act as a log host.

In addition to the commercial offerings in this space, some open source offerings may be worth investigating. OSSIM is one that is currently managed by a company named AlienVault. OSSIM is an entire Linux distribution that you install and get all of the capabilities of the SIEM rather than having to install applications on top of an existing operating system. Considering this is security information, you want your operating system installation to be as hardened as possible. OSSIM takes

the need to install and harden the operating system out of your hands, which should make the vault where your security information is kept harder to get into.

Prelude is another piece of open source software that purports to function as a SIEM in that it takes information from multiple, disparate sources like logs and intrusion detection systems, both network and host-based, and collects them together. Prelude can be installed on any Unix-like operating system and includes a web-based console called Prewikka that functions as the interface. As with other, similar solutions, Prelude uses agents that communicate back to a manager. The web interface communicates with the manager to display all of the information that has been funneled into Prelude.

You may find that you don't need a SIEM for your purposes, but because it is a possibility as a place to aggregate a lot of data sources, including log data from network devices, it's useful to know that these types of software exist. Because you can create alerts from them, it can help to know when there is an incident worth looking at, assuming that the alert has been created correctly.

Summary

Network and host-based investigations are inextricably linked because nothing happens in isolation. As a result, some of what we have talked about in this chapter revolves around collecting information from hosts. This is done in order to correlate with the network information you hopefully already have in place. The network data will tell you how they got there and what was sent, that you may not get from application logging. This information that you can collect as a point of correlation is log data from network-based sources like switches, routers, and firewalls or it can be logs from operating systems where network services are running. Because almost nothing stays in the network, network attacks are often targeted at specific network services. Understanding what is happening with the services and the operating systems can help you get a better grip on what is happening in the network.

A number of software packages and solutions can also help with the enormous volume of information that is accumulating. This may be a log management solution or a security information and event management solution. Incident response solutions are also available that can help with not only collecting information, but making it easier to perform an investigation when an incident happens.

In the end, a lot of it comes down to being prepared. All of these solutions require they be in place ahead of time, whether it's configuring NetFlow on your network devices or acquiring a software package to aid in the investigation. Enterprises should have incident response teams, which can help to set priorities, and define and implement solutions that need to be in place ahead of an investigation. Because an incident response team is often, and probably should be, cross-functional, there may be members of the network or security groups participating. This means they would be able to provide guidance on what is in place and what needs to be done.

Keep in mind that resources are always limited. Prioritizing needs and making a case for expenditures is essential. This is best done through logical, rational, and objective analysis of the potential for incidents and what is absolutely essential when it comes to performing an investigation.

8

Intrusion Detection Systems

In this chapter, you will learn about:

- The value of intrusion detection systems to an investigation
- Different types of intrusion detection systems
- Challenges that can be associated with intrusion detection systems

It may seem obvious to talk about intrusion detection systems (IDS) in a book about network forensics, though you may ask, if an IDS is in place, how much need is there for forensic investigation? The answer is that, in addition to identifying intrusions as they take place, an IDS can also be used to simply generate data that could be used during the course of an investigation. As a result, understanding how an IDS works and how to interpret the output can be beneficial to an investigator. This includes different styles of detection that may be used, the architectures that may be in place and, most specifically, the logs and alerts that the different systems will generate.

A number of intrusion detection systems are available, both commercial and open source, but the rules that are used by Snort, which started as an open source IDS, are used in a number of other places, which makes it useful to understand. As a result, we will be covering the use and implementation of Snort. We will also talk about Suricata, another open source IDS, that uses Snort rules to function. Finally, Bro is referred to as a network security monitor but it will do much the same thing that Snort and other intrusion detection systems do. Because all of these are based on the use of rules, an investigator can make use of their understanding of writing rules to help create data that could be used in the course of an investigation. This is especially true if the case is ongoing.

While the common purpose of an IDS is generally thought to be the creation of alerts so operations staff can follow up on them, there is no reason not to make use of the detection capabilities to just create log messages. An IDS can be a useful tool when it comes to performing network forensic investigations.

The alerts an IDS generates are ideally sent to a management or monitoring console. Not all alerts are sent to these consoles to be acted on, though, depending on their severity. It's also possible for alerts to simply be missed by operations staff. In fact, some attackers will trigger a flood of bogus

alerts in order to slip a real attack in through the noise. When that happens, the attack is logged by the IDS but a system may end up being compromised without anyone seeing that it happened or being able to do anything about it.

There is value for intrusion detection systems from the perspective of a forensic investigation as well as security operations staff. The value may be different, but the functionality is versatile enough that it can work for both populations.

Detection Styles

Intrusions can be difficult to detect, which may seem obvious by reading through the news. Because it can sometimes be a challenge to determine bad traffic from good traffic, there are different ways of making that determination. Challenging problems often have multiple solutions, after all. One of the issues is that illegitimate traffic can look very much like legitimate traffic. Intrusion detection shares some of the same challenges as anti-virus solutions. Often, we don't know what something bad looks like until the bad thing has happened. Using the approach of identifying bad events and determining what they look like is called *signature-based* detection. Another approach that intrusion detection systems can take is to sort out normal activity from what appears abnormal. That is referred to as *heuristic detection*.

Signature-Based

Signature-based intrusion detection systems rely on someone to identify a pattern in the malicious network traffic. This pattern is the *signature*. Because this identification relies on noticing that something bad has happened, there is a very good chance that multiple systems could be breached or attacked prior to someone figuring out what the signature should look like. Think of this as using someone's fingerprints or a mug shot after a crime has been committed as a way to be able to identify the person who has committed the crime. They are great for identifying that person if they commit a crime again, but it first required that a crime be committed in order for the identifying marker to be captured. This is what signature-based identification is—a crime (bad/malicious traffic) has to happen before the identification (signature) can be created.

This signature-based identification is a common approach to intrusion detection, just as it's a common approach to antivirus programs. When it comes to intrusion detection at the network level, the signature may be any number of individual pieces of data or a number of them together. A network signature may be the source IP address, for instance. Any traffic coming from a particular IP address may need to be flagged as being bad in order to generate an alert. It may come down to particular ports that are in use. For instance, you may want to flag whether someone is trying to use the Internet Relay Chat protocol. This means that traffic occurring on port 6667 would be flagged for alert.

Signatures are not limited to just the header information, whether it's IP headers or transport-layer headers like TCP or UDP. An IDS can look at the packet data and signatures can be generated based

on that. This may require decoding of data in the application protocol, and the IDS may have ways of handling that or else the signature may be designed specifically to address even binary protocols.

You may read this description and think about firewalls. In reality, there is some similarity between them. When it comes to firewalls, the objective is to make decisions about whether to block or allow. An intrusion detection system doesn't make such decisions on its own, though it may be integrated with a firewall to pass along information that could be used to establish firewall rules. If an IDS is placed correctly within the network architecture, it may be able to detect and alert on traffic that a firewall may block. This may mean you could get additional information about traffic that isn't getting inside your network but is still targeted at your network. You may also find traffic that should be blocked by the firewall but is showing up

Creating rules requires knowledge and skill. There can be considerable complexity in how to develop an appropriate rule to detect information. We'll get into some basics of some of the rule types in the "Snort" section later in this chapter, but just enough to barely scratch the surface. Complex rules that a network forensic investigation may need could take a lot of practice and a fair amount of knowledge of protocols and certainly of the attack or intrusion that is under investigation.

Rules are at their very core sets of patterns. Each of these patterns needs to be compared to every packet that comes through the IDS. The more patterns there are, the more comparisons need to happen. When it comes to simply matching on the headers, the location for each header field is known, which makes it easier to do the matching. The IDS can jump straight to the location in the header and determine whether or not the pattern is there. When it comes to looking deeper into the packets, however, that's a different story. Each byte needs to be checked against the pattern. That's far more processor-intensive and time-consuming than knowing exactly where a piece of information may be located.

Heuristic

A heuristic detection system is sometimes referred to as anomaly-based. It is based on the idea that we can know what normal network traffic looks like, and anything that is not normal is an anomaly. If traffic that is not normal is seen by the IDS, alerts get generated. Just as a signature is like a mug shot or set of fingerprints, an anomaly-based system is like a door alarm or, perhaps more accurately, a motion detector. Motion detectors expect a particular set of circumstances—a lack of motion. When any motion occurs, it's an anomaly and the system alerts on it.

The problem with approaches like this is the potential for alerts that aren't really intrusions. You can see this in the case of a motion detector perhaps more easily than trying to think about it from a network perspective. If a window is left open and a breeze comes up, it may rustle a piece of paper. That rustle may be enough to trip the motion detector. But even though the alarm goes off, there is no intrusion; no burglar has entered the premises. As another example, you may have left something in a precarious position before setting the alarm and leaving the premises. It may have appeared to be stable and not moving at all, but it could slip and eventually fall. This is another movement that may trigger an alarm.

From the standpoint of a security engineer, an anomaly-based system can be challenging because it may require a lot of tuning to keep the baseline up to date. The baseline determines what is considered normal so that anomalies can be detected. From the standpoint of a forensic investigator, though, an anomaly-based IDS can be useful. It may provide a lot of data about anything that occurs that isn't normal, which may be a lot of information. The problem is that, unlike signature-based systems, there is no control over what gets detected. Another problem is that, as mentioned earlier, attack traffic can look like regular traffic. If someone is attacking a web server, they will do it using HTTP, which is the protocol that the web server understands. This is not to say it will look entirely normal, but it may look normal enough that it will be difficult to differentiate between the attack and regular traffic unless the IDS is good.

Any heuristic or anomaly-based IDS requires a baseline to work from. This means the system has to go through a period of learning. The baseline then needs to be checked as traffic comes through. How the baseline is implemented depends entirely on the vendor who has developed the software. Once the baseline has been created, you will always need to check against the baseline. A signature-based IDS can have multiple rules, and the number of rules can increase over time. This can increase the processing needs of a signature-based IDS. This makes signature-based different from heuristic because signatures are constantly updating, but a baseline remains a baseline until something significant changes, such that the baseline needs to be altered.

Host-Based versus Network-Based

In addition to two different detection styles, there are also two types of IDS. The first is network-based and the second is host-based, and though we are primarily concerned with the former, it's worth talking about both. Although a host-based IDS doesn't have network information, it could be used as corroborating information in an investigation. It's also important to make a distinction between them. While the goal is to detect intrusions at the time they happen, or at least very near the time, the method for detecting the intrusion is very different. Network intrusion detection systems are focused on looking at packets, while host-based intrusion detection systems are focused on what happens on the host. As noted in Chapter 4, the network can't lie. The signals on the wire are what has happened.

There is a challenge with network intrusion detection systems (NIDS), however, that can't be addressed easily. If traffic is encrypted, the NIDS can't do any detection on the payload because it's encrypted. The NIDS would be limited to looking at the headers through the transport layer, which cannot be encrypted. The only way for a NIDS to be able to detect encrypted traffic is for it to have the decryption key, but if the NIDS has the decryption key, it's a violation of end-to-end encryption and confidentiality. We'll get into encryption in more detail later on, but it would be very uncommon for a NIDS to have encryption keys. A host-based IDS, on the other hand, detects events after the decryption process has already happened. A host-based IDS is looking at the events within the operating system after the network traffic has been received and action has been taken by the receiving application.

Although we make a distinction between a network IDS and a host-based IDS, it's also true that you can run a network IDS on a host. The difference there is what is being analyzed. A network IDS looks at traffic that has either been fed directly into the IDS from the network interface or has been captured and saved. In the software we will be looking at here, you can use `tcpdump` or Wireshark to capture the data and store it for analysis offline. This can be helpful if the engagement you are working on has managed to capture some data. Capturing data this way doesn't mean that the opportunity to analyze it through an IDS has been lost. You can feed the packet capture (pcap) file into your IDS to generate alerts. This can save a lot of time in your analysis, because you don't have to manually dig through the packet capture yourself.

NOTE The pcap file format started life in the early 1990s as the BSD Packet Filter (BPF). It has since evolved into the pcap format with an associated application programming interface (API) and library called `libpcap`. The `libpcap` library is part of what makes `tcpdump` work, but it has also been utilized in numerous other programs as well, including the programs we will be looking at in this chapter.

The analysis does require that you have relevant rules set up for what you are looking for. With the systems we are looking at here, there needs to be a rule established for the IDS to alert on. These systems won't just automatically detect bad traffic for you. Even if you were using an anomaly-based system, the bad traffic you were looking for would have to look enough like an anomaly to be detected. An IDS is not a magic bullet to alert on every type of bad network traffic that was available.

The first software package we'll be taking a look at here is Snort. The Snort rules can be used across multiple packages, which makes learning how to read and create the rules useful, even if you aren't using Snort itself. Snort has been around for a long time but another old software package called Bro has seen a lot of use in recent years, especially by incident response professionals. Bro is actually a network monitoring framework, but it can be used to create a network IDS. Because of its power and flexibility, it's worth looking at.

Snort

Though Snort is not the first network IDS, it has been around for a long time and exists in both open source and commercial offerings. The commercial version of Snort was Firepower, offered by the company Sourcefire (which was acquired by Cisco in 2013). Snort was created by Martin Roesch in 1999. Snort is highly configurable and can be run on multiple operating systems and hardware platforms. Rather than relying on `libpcap` as it once did, Snort has introduced a Data Acquisition (DAQ) layer. The DAQ provides an abstraction for the means of handling packets, which decouples it from `libpcap` so other libraries can be used, depending on the platform Snort is running on. While you'll likely be using Snort on either Windows or Linux and they will use some flavor of `libpcap`, the software is capable of running on other systems, including systems like an IDS appliance running a real-time operating system.

Snort has been around long enough that it is not only mature in the sense of capabilities, but it has also had a lot of contributors, which provides for a lot of functionality. Before you even get to the point where you can start thinking about the rules you are going to enable or write, you need to think about the capabilities you want to enable and how you want those capabilities configured. Snort allows for a lot of configurability and also adds external functionality like preprocessors. This means a user can enable that additional functionality from one of the preprocessors that Snort has built in, or someone with knowledge and skill can add their own preprocessor.

Preprocessors

Snort is highly configurable, including not only multiple options for output but also the ability to add modular plug-ins called preprocessors. The preprocessors that have been enabled run before any of the rules are processed, but after the packet has been decoded by Snort. Snort handles packets but each packet may not tell the whole story, and if the detection were based solely on a packet, it would be easy to bypass detection. The only thing an attacker would need to do would be to break apart the communication so the entire sequence of bad traffic wasn't confined to a single packet. A complete conversation can take place over a large number of packets, and since TCP is a stream-based protocol, one of the preprocessors available with Snort is one to reconstruct a TCP stream in order to better perform more complex detection.

Even though TCP is a stream-oriented protocol, conversations also happen over UDP. The protocol itself doesn't support streams, but that doesn't mean streams don't happen using UDP. The Trivial File Transfer Protocol (TFTP) is an example of a protocol that streams data but uses UDP as the transport protocol. Just like firewalls will keep track of state with UDP communications, Snort can also keep track of the state of UDP communications. This is even more important, perhaps, with UDP because UDP communication can be easily spoofed. Someone can easily inject packets that could mimic an existing stream, so having an IDS that can look for bad injections is important. Not all higher-level protocols support anti-spoofing measures, after all, which means injections can be possible.

On top of preprocessors for reconstructing lower-layer protocols like TCP and UDP are a number of application-layer protocol preprocessors. When so many attacks happen at the application layer, getting some additional help there is valuable. SMTP, POP, HTTP, and SSH are all preprocessors that are available within Snort, and anyone with programming skill can add additional preprocessors. The objective of the preprocessor is to make it easier to detect more complex attacks.

Each preprocessor comes with configuration settings that alter the behavior of Snort. Different configurations can enable better detection. As an example, the Session preprocessor is about managing streams of different protocols. You can enable the preprocessor to track streams for TCP, ICMP, and UDP, but you can also disable that ability on a per-protocol basis. This session tracking can be resource-intensive because Snort has to maintain data for each session. To limit the amount of resources Snort is consuming, particularly if the network were to come under attack, the Session preprocessor can be configured to limit the number of streams it is keeping track of.

This is just one example of the type of configurability that is available within the Snort preprocessors. Each of the preprocessors will have their own settings that can be manipulated within the Snort configuration. An example of the default configuration for the Session preprocessor is shown in Listing 8-1.

Listing 8-1: Snort Stream Preprocessor Configuration

```
preprocessor stream5_global: track_tcp yes, \
    track_udp yes, \
    track_icmp no, \
    max_tcp 262144, \
    max_udp 131072, \
    max_active_responses 2, \
    min_response_seconds 5
preprocessor stream5_tcp: policy windows, detect_anomalies,
require_3whs 180, \
    overlap_limit 10, small_segments 3 bytes 150, timeout 180, \
    ports client 21 22 23 25 42 53 79 109 110 111 113 119 135 136 137
        139 143 \ 161 445 513 514 587 593 691 1433 1521 1741 2100
        3306 6070 6665 6666 6667 6668 6669 \ 7000 8181 32770 32771
        32772 32773 32774 32775 32776 32777 32778 32779, \
    ports both 80 81 311 383 443 465 563 591 593 636 901 989 992 993
        994 995 1220 1414 1830 2301 2381 2809 3037 3128 3702 4343
        4848 5250 6988 7907 7000 7001 7144 7145 7510 7802 7777
        7779 \ 7801 7900 7901 7902 7903 7904 7905 7906 7908 7909
        7910 7911 7912 7913 7914 7915 7916 \ 7917 7918 7919 7920
        8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180 8243
        8280 8300 8800 8888 8899 9000 9060 9080 9090 9091 9443 9999
        11371 34443 34444 41080 50002 55555
```

This configuration sample says that we want to track TCP and UDP but we don't want to track ICMP within the `stream5_global` preprocessor, which is the module name for the Session preprocessor we have been discussing. Additionally, TCP and UDP have different settings for the maximum number of sessions that Snort will track using this preprocessor. The `max_active_responses` setting indicates the number of responses Snort will provide. Snort can be configured to respond to messages that it receives. It does this to protect the system or network. In order to drop sessions that are unwanted, Snort can be configured to send a RST message or an ICMP error message. This setting configures the maximum number of those messages that will be sent. Without this setting, Snort could be co-opted into participating in a denial of service attack because it could be replying to a large number of bogus messages, flooding the network connection in reverse. This way, Snort can send an appropriate response to a mistaken connection attempt and ignore anything that is malicious beyond that, other than any alert that has been configured.

The `stream5` preprocessor can be configured to have a different policy based on the operating system at the target. The configuration shown in Listing 8-1 indicates that the policy specified is for Windows systems. This particular policy states that in order to be considered a session within TCP, the three-way handshake has to be completed within 180 seconds. This is not the default setting for this module. By default, the module will track sessions that have not completed a three-way handshake. The policy also says that a maximum of three small segments are allowed before the anomaly gets flagged. According to the policy, a small segment is one that is 150 bytes or less.

Because some rules have a directionality component, it is necessary to track which end is the client and which end is the server. You can see this differentiation in the policy where ports on which the stream reassembly will take place are noted, based on client, server, or both. You can limit the amount of stream reassembly done by limiting the number of ports. You can also disable reassembly for ports that may not be in use on the network on which you are running Snort. If you don't have services there and there are no rules, there is no point in having Snort bother to reassemble the stream for you. Remember, Snort relies on rules to do alerting. Aside from easily flagged anomalies at the protocol level like small segments, Snort won't reconstruct an FTP stream, for instance, and start flagging problems with FTP if you don't have any rules that are targeting FTP communications. Some of the preprocessors have detections built in but for the most part, it all comes down to the rules you have enabled.

Configuration

Most instances of Snort have a very extensive configuration file that is included by default, if you are using either a package created by your Linux maintainer or you are using a Windows installer. While this is helpful because of the amount of documentation that is included in the configuration file, it can also be overwhelming to look at all of the available configuration settings. The file that you will typically look at is very long and has a lot of configurations that you can manipulate. In most cases, the way that Snort is set up by default will work fine for you. However, you may want to take a look at some settings either because you have to set them up or because they may potentially make your life easier.

As one example, Snort knows nothing at all about your network design and layout. However, you can tell Snort what all of your internal networks are so it can differentiate between what is internal and theoretically trusted and what is entirely external. The configuration keyword Snort uses for setting these variables is `ipvar`. Using `ipvar`, you can create variables that can be used later on. For instance, setting `ipvar HOME_NET` tells Snort what IP addresses are considered to be in your home or internal network. Anything outside of that may be considered external, especially if you were to use the configuration setting `ipvar EXTERNAL_NET !$HOME_NET`. This says anything that is not in the variable `HOME_NET` should be considered `EXTERNAL_NET`.

NOTE The use of the `$` symbol in front of a variable name essentially dereferences it. This means that if I were to use `$HOME_NET`, what I really mean is that I want this replaced with the contents of the `HOME_NET` variable.

To make life easier for your rules later on, you can get even more granular by telling Snort where all of the different server types are on your network. The configuration fragment shown in Listing 8-2 will tell Snort where different server types may be located.

Listing 8-2: Fragment of `snort.conf` with Server Settings

```
# List of DNS servers on your network
ipvar DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
ipvar SMTP_SERVERS $HOME_NET

# List of web servers on your network
ipvar HTTP_SERVERS $HOME_NET

# List of sql servers on your network
ipvar SQL_SERVERS $HOME_NET

# List of telnet servers on your network
ipvar TELNET_SERVERS $HOME_NET

# List of ssh servers on your network
ipvar SSH_SERVERS $HOME_NET

# List of ftp servers on your network
ipvar FTP_SERVERS $HOME_NET

# List of sip servers on your network
ipvar SIP_SERVERS $HOME_NET
```

Similar to `ipvar`, you can use `portvar` to set port variables. This may be helpful to configure `SSH_PORTS` and then use the variable in your rules. If you are using multiple SSH ports on your network rather than using the single, default port, having a variable that can be referenced is quite a bit easier than having to re-create or alter rules in order to catch SSH attacks. Referencing `SSH_PORTS` in the rule allows you to change `SSH_PORTS` in the configuration file, and all the rules get the changes automatically once Snort has re-read the configuration file and re-loaded the rules. This shorthand can save a lot of time and maintenance work.

In addition to `ipvar` and `portvar`, you can also use `var` to specify variables. These variables may be useful if you are setting directories, as in the location where Snort may find preprocessors and rules. This allows you to set whatever directory you like for your rules and even move them around. All you have to do is change the variable, and anything that references that variable automatically gets changed.

You can configure a number of settings, which will typically result in either more or fewer alerts, depending on how you have your settings configured. The configuration file fragment shown in Listing 8-3 configures a number of settings that will change the way Snort alerts on behaviors.

Listing 8-3: Snort.conf fragment with Configuration Settings

```
# Stop Alerts on experimental TCP options
config disable_tcpopt_experimental_alerts

# Stop Alerts on obsolete TCP options
config disable_tcpopt_obsolete_alerts

# Stop Alerts on T/TCP alerts
config disable_tcpopt_ttcp_alerts

# Stop Alerts on all other TCPOption type events:
config disable_tcpopt_alerts

# Stop Alerts on invalid ip options
config disable_ipopt_alerts

# Alert if value in length field (IP, TCP, UDP) is greater the length
  of the packet
# config enable_decode_oversized_alerts
```

The comments, indicated by a # at the beginning of the line, provide documentation about what each configuration setting is doing. Most of the ones you can see here are based on disabling certain alert types. This is likely because the alerts would generally be false positives. If you are interested in seeing more of each of these types of alerts, you can easily change `disable` to `enable` in the configuration file and then restart Snort. The new changes will be picked up and you will start to see more of these alerts. In some situations, more of these types of alerts are good. The default for these is `disable` because most people are not going to be particularly concerned with them. Modern operating systems are generally going to be okay with handling the types of attacks that these sorts of alerts may flag. Since there is little danger, any alert would just create more data than the security operations team probably needs to have.

NOTE Remember that when it comes to alerts, more is not better. On the operations side, every alert should be looked at. Creating a lot more alerts has the potential to overwhelm the operations or monitoring team. This could end up in legitimate attacks being missed because they were buried in a lot of alerts that were meaningless. Enabling all available rules is almost never the way to go.

The preprocessor section is extensive, if you are choosing to enable all or even most of the preprocessors. If your network is typical, it will likely have a mixture of server types, and the preprocessors that help to decode and translate the different protocols are helpful in getting useful alerts. Because of that, each preprocessor needs to be enabled and then configured. The `stream5_tcp` configuration settings shown earlier are only one section of the entire configuration for the Sessions preprocessor. UDP would also have a set of configurations, as would ICMP were it enabled. This is also just a single

preprocessor. Snort has a number of available preprocessors, and each would have a number of lines of configuration settings.

The next important section is where you indicate the output types. This isn't only about where you output to but also about what gets output—Snort can output alerts but it can also output packets, which provide the traces necessary to see what actually happened. You could also configure Snort to output both. Snort has no problem at all with providing output in multiple forms simultaneously. You can see the default configuration settings for the `snort.conf` file in Listing 8-4.

Listing 8-4: Snort.conf Output Section

```
# unified2
# Recommended for most installs

output alert_fast: alerts

# output unified2: filename merged.log, limit 128, nostamp,
#   mpls_event_types, vlan_event_types
output unified2: filename snort.log, limit 128, nostamp,
#   mpls_event_types, vlan_event_types

# Additional configuration for specific types of installs
# output alert_unified2: filename snort.alert, limit 128, nostamp
# output log_unified2: filename snort.log, limit 128, nostamp

# syslog
# output alert_syslog: LOG_AUTH LOG_ALERT

# pcap
# output log_tcpdump: tcpdump.log
```

The one change from the default that was installed on an Ubuntu Linux system is the addition of `output alert_fast: alerts`. This tells Snort to just dump alerts out to a file without any of the other data. In the `unified2` output type, Snort is sending both alerts and packets to a single alert file. Even this has a number of settings that determine how and what is being logged. As an example, the limit of the file size in megabytes is 128. If you want larger output file sizes, you can configure that. This also indicates that MPLS and VLAN event types should be included, though those settings also require that Snort has been compiled with that ability built in. If Snort was not configured that way, these output types would not be created because Snort would have no way of generating them. When Snort is built from source code, features have to be enabled that turn on the ability to output in particular ways, such as to a database.

The output provided by Snort can be difficult to parse easily, which is why additional software may be required. This additional software can also be necessary because some of the output types have

been removed from Snort. The program `barnyard2` can read the Snort `unified2` output and forward the data along to other output types, including databases.

Rules

As hinted at earlier, while preprocessors and other capabilities are important to Snort, the rules are the heart of what happens. Although you can create your own rules, a substantial number of rules can be provided for you. By default, the Snort source doesn't come with rules. Some Linux distributions will include some rules with the Snort package but that's not always the case. Because rules are regularly being contributed, these additional signatures need to be downloaded. This means that new rules need to be pulled, and Snort doesn't come with the ability to do that without help. You can use a program like `pulledpork` to regularly download the new sets of rules that are available. `Pulledpork` and other automatic rules managers require an `oinkcode` that can be obtained for free from `Snort.org`.

New rules are typically based on new types of attacks. Because rules can be so specific, new rules are needed to really target the specifics of newer attacks. This is especially important because the message that is configured in the rule indicates the type of attack. A generic rule may match a number of different types of attacks, but that would require additional work on the part of the operations staff looking at the alerts. This means they would have to do an investigation before they determine what happened and whether something needs to be done. More detail in the message can help to easily determine the response that would be needed. This additional detail can result in a lot of new rules with a lot of detail in them.

To understand how Snort rules are constructed, let's look at a couple of very simple rules, shown in Listing 8-5. The lines have been numbered to make things clear, but the numbers would not be included in a rules definition. It just makes it easier to see where one rule stops and the next starts because the lines can appear to run on if you aren't familiar with what to look for.

Listing 8-5: Snort Rules

```
1. alert icmp any any -> any any (msg:"Large ICMP packet"; dsize:>1200;  
   reference:arachnids,246; classtype:bad-unknown; sid:200049; rev:4;)  
2. alert tcp any any -> any any (msg: "SYN scan"; flow: not_established,  
   to_server; threshold: type threshold, track by_src, count 15,  
   seconds 60; flags: S; sid: 100090; rev: 1;)
```

The first rule is based on the ICMP protocol. The rule says that messages can come from anywhere and be destined anywhere. The `any any` indicates source address and port, and though there are no ports in ICMP, the structure of the rule remains the same. Once we have identified the source, destination, and the protocol, we can start looking at what the rule actually says. This all takes place inside the parentheses.

The order of the pieces of the rule doesn't make any difference. Each keyword that is essential to the rule is followed by a colon. The first keyword is `msg` and it indicates what should be included

in the alert should this rule trigger. This particular rule is paying attention to the data size (`dsize`). What we are looking for is an ICMP packet with a payload larger than 1200 bytes. The `reference` keyword indicates where to go for additional information. `arachnids` indicates that the rule may have come from the database at WhiteHats.com. This particular rule doesn't care what type of ICMP message is sent, just that the packet size is large.

After the `reference`, we indicate the `classtype`. This is a way of categorizing the rule, which can allow for additional statistics related to what is being detected. This type of information can provide insight into the areas where resources may be deployed based on the attacks that are common. The `classtype` allows you to pull a number of specific rules into a single bucket. Finally, the Snort ID (`SID/sid`) is just a numerical identifier to identify the particular rule, and the revision number. If the rule changes, the revision number can also change to indicate there have been alterations to the rule.

The second rule uses TCP rather than ICMP but we are still looking at any source address and port as well as destination address and port. In order to change that, we can use classless interdomain routing notation. For example, you could say `alert any any -> 10.0.0.0/8 80` to indicate that the destination is anything on the 10. (sometimes pronounced ten dot) network with a destination port of 80. This means the rule applies to the web port on anything on the private address range of 10.0.0.0–10.255.255.255.

These, as indicated, are very simple rule examples with only a subset of capabilities of Snort. The rules in Listing 8-6 are much more complex with more checks.

Listing 8-6: More Complex Snort Rules

```
1. alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"COMMUNITY ICMP
Linux DoS sctp Exploit"; icode:2; itype:3; content:"|28 00 00 50 00
00 00 00 F9 57 1F 30 00 00 00 00 00 00 00 00 00 00|";
reference:nessus,19777; classtype:attempted-user; sid:100000164;
rev:2;)
2. alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB
msqueue andx bind attempt"; flow:established,to_server; content:"|00|";
depth:1; content:"|FF|SMB"; within:4; distance:3; pcre:"/^(\\x75|\\x2d|
\\x2f|\\x73|\\xa2|\\x2e|\\x24|\\x74)/sR"; byte_test:1,!&,128,6,relative;
content:"%"; depth:1; offset:39; byte_jump:2,0,little,relative; content:
"&|00|"; within:2; distance:29; content:"|5C|PIPE|5C 00|"; distance:4;
nocase; byte_jump:2,-17,relative,from_beginning,little; pcre:"/^
{4}/R"; content:"|05|"; within:1; byte_test:1,!&,16,3,relative; content:
"|0B|"; within:1; distance:1; content:"|B0 01|R|97 CA|Y|D0 11 A8 D5 00
A0 C9 0D 80|Q"; within:16; distance:29; flowbits:
set,smb.tree.bind.msqueue; flowbits:noalert; reference:cve,2003-0995;
reference:url,www.eeye.com/html/Research/Advisories/AD20030910.html;
reference:url,www.microsoft.com/technet/security/bulletin/MS03-026.mspx;
classtype:protocol-command-decode; sid:3164; rev:2;)
```

The first rule is an ICMP rule, much like the first rule in Listing 8-5. Unlike that rule, which didn't care what ICMP type was being sent, this rule checks that. This rule is looking for an ICMP

destination unreachable message (type 3) and more specifically, protocol unreachable (code 2). This is a message that would be originated by a system if a sender were trying to communicate using a protocol not supported by the recipient. You will also note that there is a content field in the rule. This content is specified as hexadecimal bytes, indicating that the data is likely binary in nature and not representable in an ASCII format, which would present as a readable string, just as this text that you are reading is. If the data translates to unprintable characters, it needs to be represented as bytes and hexadecimal is a quick way of representing that information. A single 8-bit byte can be represented as two hexadecimal characters rather than a string of eight 0s and 1s. The rule comes from a set of community-contributed rules, as indicated by the message that gets sent if the alert is triggered.

The second rule is quite a bit longer because it is looking for more detailed information within the packet. This rule also indicates flow designation, which was discussed earlier. First, this is an established connection, meaning that the three-way handshake has been completed and the two systems are in conversation. The rule specifies that it should only apply on messages directed to the server. The server would be the recipient of the initial SYN message and the client is the sender.

NOTE Regardless of the type of system you may be looking at, clients originate requests to servers. A web server can originate a request to another system, even a desktop, but the web server originating the request would be the client in that communication stream.

Beyond the flow, the rest of the rule primarily deals with the content of the message. Using keywords like `within` and `distance`, you can specify where in the message to look for specific content. When writing rules, you can indicate exactly where in the message to start looking by using the `offset` keyword. This is the byte location as offset by the beginning of the data portion of the packet. This doesn't include the headers from the transport and network protocols. Only the data portion of the packet is considered when it comes to offsets. It may be hard to see but there is an offset indicated in this rule. The offset indicates 39 bytes from the start of the data payload in the packet.

To determine where to stop, you have to use a keyword like `depth`. As an example, in this rule, you can see that it specifies `|00|` as the very first byte in the packet. This is indicated by the depth of 1. If we wanted to allow that `|00|` byte to be within the first three bytes, we could specify a depth of 3.

You can match on multiple contents within the packet. Once you have found the first match, you can tell Snort where to start looking for the next match. You do this with the `distance` keyword, which indicates how far away from the first match Snort should start looking for the next one. A distance of three bytes is essentially the offset from the last content match point. Whereas `distance` tells Snort to jump ahead, `within` says that a match must occur within the specified number of bytes. The `within: 16` in Listing 8-6 says that the match must occur within 16 bytes of the last match and no further than that. Between `distance` and `within`, you get the upper and lower bounds of where Snort should be looking for a match.

There are different ways to look for content within a packet. The first is by byte values using a hexadecimal notation. The second is an ASCII-printable string. You could indicate that you are looking for the content “n00bz,” which would be a human-readable string and not bytes that are not represented in a way humans can easily read. Another way to look for content is to use pattern matching, such as Perl compatible regular expressions (PCRE). Regular expressions are ways to programmatically identify patterns where you may know what content should look like without necessarily knowing what the content actually is. As an example, Listing 8-7 shows a way of identifying phone numbers using a regular expression. We don’t know what the phone number is necessarily, just that we are looking for something that looks like a phone number in its pattern.

Listing 8-7: PCRE for Telephone Number

```
'/^(((([0-9]{1})*[- .]*([0-9]{3})[- .]*[0-9]{3}[- .]*[0-9]{4}))+)*$/'
```

Using this collection of rule settings, you can get very specific about what you are looking for. This does, though, require that you can read the data. There are two problems here. The first is that you need to have collected enough data to know exactly what the rule should look like. The second is that if you get very specific, but patterns within the packets change from one packet to another, your rule won’t match. As an example, if the pattern in half of the packets associated with an intrusion is |00 0C| and the other half is |00 0B| but you only saw half of them and your rule is based on |00 0C|, then you’ll miss half of the messages. Your detection is only as good as your rule and your rule is only as good as the information you have that led to the creation of the rule.

Suricata and Sagan

Suricata is an open source intrusion detection system that can use Snort rules to operate. In addition, it can output to the same `unified2` output format as Snort. When Suricata was developed, it had the advantage of being multithreaded where Snort was single-threaded. The multithreaded design allowed for better performance with Suricata because multiple execution threads could be operating on messages at the same time. Snort has since moved to a multithreaded program design in version 3.0. Suricata was also designed to be more oriented toward application layer protocols. When it was developed, it had the capacity to do advanced work with HTTP. As mentioned earlier, Snort supports multiple preprocessors to handle application-layer protocols like HTTP.

One significant difference between Snort and Suricata is the configuration file. Whereas Snort uses a complex configuration file with different ways of setting information, Suricata is very straightforward, using YAML (Yet Another Markup Language or YAML Ain’t Markup Language), which is a way of serializing data in a human-readable format. YAML uses a simple and straightforward way of storing complex data. Unlike XML, YAML doesn’t have as much overhead but is similarly self-documenting

if used correctly. The default Suricata configuration file provided on an Ubuntu system is detailed and large, just as the Snort configuration file is. Because it's in YAML, though, it looks quite a bit different, as shown in Listing 8-8.

Listing 8-8: Suricata Configuration Fragment

```
# Configure the type of alert (and other) logging you would like.
outputs:

# a line based alerts log similar to Snort's fast.log
- fast:
    enabled: yes
    filename: fast.log
    append: yes
    #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'

# Extensible Event Format (nicknamed EVE) event log in JSON format
- eve-log:
    enabled: yes
    filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
    filename: eve.json
    #prefix: "@cee: " # prefix to prepend to each log entry
    # the following are valid when type: syslog above
    #identity: "suricata"
    #facility: local5
    #level: Info ## possible levels: Emergency, Alert, Critical,
    ## Error, Warning, Notice, Info, Debug
```

At the top level, we have a configuration block. The block is for outputs. The `:` indicates that what comes after is all of the parameters that go with the outputs section. YAML uses whitespace to indicate where in the block it is. Everything at the same indentation level belongs to the same level of configuration. `fast` and `eve-log` are two different sections of outputs, and everything indented under those two belong to each section header. Under `fast`, for instance, we have the parameter `enabled`. This fragment of configuration indicates that fast output has been enabled. We can also see that there is a filename. All fast alerts are to be written to the log file `fast.log`. It also indicates to append to the log file. The filetype is commented out, as indicated by the line starting with a `#`. The subsequent comment, which would be left in place if the line was uncommented, indicates that the options for this parameter are `regular`, `unix_stream`, or `unix_dgram`. The latter two would be used in the case of network output. Stream indicates using TCP while `dgram`, short for datagram, would be for UDP.

Sagan isn't so much a network intrusion detection system as a correlation engine. A correlation engine is used to pull together disparate sources and make sense of them. It is included here because it uses a similar structure to Snort's configuration file and also uses Snort's rule format. This allows Sagan to be used with an existing Snort installation. The same programs that can parse output and

pull rules for Snort can also be used for Sagan. Using Sagan, scripts can be triggered based on an alert. This allows Sagan to respond to alerts rather than simply alerting on them.

While there is no particular reason to use either Suricata or Sagan to perform analysis of the network, you should be aware that if you happen to run into one of these installations, you can use what you know about Snort to create rules. One disadvantage to both Suricata and Sagan is that by comparison with Snort, the documentation level isn't as high. This is not to say that they should be discounted, however. Both continue to be actively developed. Both have had releases within a matter of days or weeks of this writing. While intrusion detection systems are generally only as good as their rules, there is enough additional functionality in both of these software packages to make them worth looking at.

Bro

Bro has been around about as long as Snort has. It was first described as an intrusion detection system in 1999 by Vern Paxson at the Lawrence Berkeley National Laboratory. Paxson was also affiliated with the AT&T Center for Internet Research. The intention of Bro was to provide a network intrusion detection system that could support high-speed connections like Fiber Distributed Data Interface (FDDI). At the time Bro was developed, FDDI was about as fast as you could go. The challenge with network intrusion detection systems at the time was that the hardware available for these systems to run on was insufficient to keep up with a fully utilized network. It was not uncommon for network intrusion detection systems to run on slower network links and sometimes at only half-duplex, which meant that the link could either transmit or receive but not both at the same time.

Unlike Snort and the other IDSs that we have looked at, where rules that are essentially pattern matching are written for detection of events, Bro has an entirely different way of looking at events. Rules are not simply ways of identifying packets and alerting on them. Instead, Bro is based around the idea of policies and events. Bro is event-driven, which means that when events happen, functions get called. Bro provides a lot of flexibility, as compared to an IDS like Snort, that provides a small number of actions that can be triggered. Bro, by contrast, allows you to write scripts to perform the detection and also handle how you want to respond. This not only allows for custom alerting but also for reacting to events. An example of an event response is shown in Listing 8-9.

Listing 8-9: Bro Event Script

```
event http_reply(c: connection, version: string, code: count, reason:
string)
{
    if ( /^[hH][tT][tT][pP]:/ in c$http$uri && c$http$status_code
== 200 )
        print fmt("A local server is acting as an open proxy: %s",
c$id$resp_h);
}
```

This script triggers when an HTTP reply is seen by Bro. The script looks for http without caring about case and also for the status code to be equal to 200. The status code 200 means success. To be clear, the script isn't generating the event. It is responding to the event. Bro generates the event based on either watching network traffic or by looking at data in the pcap file provided to Bro.

The architecture and design of Bro provides a lot of power. The scripting language used by Bro provides more flexibility when it comes to identifying packages. Even though the Snort rules syntax is very rich and powerful, the scripting language enables you to dig deep into the data that has been provided. Like most scripting languages, the one used by Bro supports constants, local variables, data types, and sets. Additionally, Bro supports tables, which can allow for quick lookup of information.

The `bro_init` event gets triggered when the system is started up and should be handled by scripts that you are writing. This allows the script to initialize anything that needs to happen within the context of the script, including creating logs. The function `Log::create_stream` will generate a log file that can be written to within the script. Anything can be sent to the log file. If you can generate a piece of data programmatically, you can write that data out to the log you have created. This means you can provide much more context than you might otherwise be able to using just a Snort rule.

It does mean, however, that writing event handlers for Bro can be quite a bit more complex than writing Snort rules. It requires learning a little about programming and also understanding the nature of event-driven systems. Using an event-driven system, you are waiting on events to happen rather than driving the behavior of the script yourself. Once an event happens, you have control over whatever data has been generated by the event. However, you are still bound within the context of the event. This is not nearly the same as starting from scratch with a script that you have written yourself. In fact, in many regards, while you are bound within the context that Bro provides to you, you are also provided with a number of frameworks that can make your life quite a bit easier.

Where languages like Python, C++, or C# have libraries that may be provided by the vendor to make life easier for the programmer, Bro has frameworks. These frameworks provide the ability to do things like simple pattern matching, using the signature framework. The signature framework allows someone to write more traditional rules that are based on pattern matching. A simple signature is shown in Listing 8-10.

Listing 8-10: Bro Signature

```
signature basic_http {
    ip-proto == tcp
    dst-port == 80
    payload == /GET/
    event "Found a GET request"
}
```

This signature simply looks for a GET request in a web communication. The destination port is 80, which is the well-known port for web-based communications. The signature framework allows

for signatures to be written based on network and transport protocol headers. This includes source and destination address and port. In addition to basic content matches, the signature framework also directly supports HTTP by allowing matches in the `http-request`, `http-request-header`, `http-request-body`, `http-reply-header`, and `http-reply-body`. In addition, Bro supports FTP and finger within the signature framework. All of this is just the start of what the signature framework for Bro is capable of doing.

Other frameworks offered by Bro include file analysis and logging. Additionally, Bro offers a NetControl framework that allows Bro to interface with external network devices like firewalls. Bro can also interface with routers and switches. This turns Bro into an intrusion prevention system because it can send messages to devices that are capable of blocking traffic, telling them what traffic should be blocked. Functions like `drop_address`, `drop_connection`, `redirect_flow`, `quarantine_host`, `whitelist_address`, and `whitelist_subnet` all provide a lot of control over how Bro can have other devices respond to threats.

Using a tool like Bro, a network administrator can not only generate a lot of data that could be used by a forensic investigator or incident responder, but also can protect the enterprise at the same time.

Tripwire

Tripwire is very different from the IDS tools we have talked about so far. It is, however, useful to provide correlation with the other IDS results. Tripwire is a host-based IDS and was long focused on identifying file changes. Commercial versions of Tripwire have moved on to broader functionality but when most people hear Tripwire, they think of monitoring files. Other programs like Samhain can provide similar functionality. Programs like Samhain and Tripwire generate a table of cryptographic hashes of the contents of critical system files. Periodically, the program runs through the filesystem, generates hashes of all of the critical system files that have been configured, and compares them against the database. If there are changes, Tripwire generates an alert indicating that the file has changed.

This sounds pretty simple based on that description. In actuality, it takes a lot more work to ensure that the system itself is protected. If an attacker broke into a system and found Tripwire running, it should be easy to just update the database once any critical system file was altered. This means that it shouldn't be easy to update the database. It should require additional protections like passphrases to unlock the cryptographic keys that have been used to encrypt the database.

As mentioned, Tripwire is not the only program to offer this functionality, though it was the first. It was originally developed in the early '90s and was contributed as open source in 2000. There is currently an open source version of Tripwire in addition to the commercial offering by the company of the same name. Although it doesn't have anything to do with network traffic directly, it can be used to correlate attacks. Attackers who gain access to systems will likely make changes. This may be especially true with regard to anything related to system services since the attacker will probably want some sort of remote access. This could include a backdoor or just a process that periodically checks another system like a command and control server. Because configuration changes need to happen in order for an attacker to install that kind of software, Tripwire and other, similar, software can help generate the alerts necessary to provide correlation.

OSSEC

OSSEC is another host-based intrusion detection system, which means it's not directly related to network forensics. In addition to file integrity monitoring, which is what Tripwire does, it also offers monitoring for rootkits, log monitoring, and process monitoring. While file integrity is important, monitoring of file integrity is limited to files that have been configured to be checked. Files can be added to the system without any notice being taken. If we are only looking for changes to existing files, and not new ones being added, we could have a problem if they are malicious files. Also, as noted earlier, file integrity checks are not perfect. This is why the additional checks offered by OSSEC are important. A rootkit detection capability determines when backdoors have been added to the system. This may also include alteration of system programs that could otherwise be used to detect the existence of the rootkit.

Log monitoring is important, even in cases where there is file integrity checking. This is especially true when log monitoring is done in real time and alerts are sent off-system. Anything that is on the system is suspect because it could be altered by the attacker. This includes log entries that may disguise their events, like any audit logs that may exist. Process monitoring is another important aspect because attackers may start up additional processes in order to accomplish what they need to do, whether it's to steal data or join the system to a botnet. The process monitoring can detect that behavior.

In cases where the network traffic is encrypted so the behaviors can't be seen within the packet captures, this level of insight can be incredibly important. The network traffic can provide evidence that the communication is happening between the local system and a remote system, and a tool like OSSEC can provide some additional evidence of what is happening on the local system.

Fortunately, OSSEC is multiplatform, like the other tools we have been talking about. It is also open source and free to use. This makes it another valuable tool to have in your arsenal as you are investigating incidents and attacks.

Architecture

One of the challenges associated with network intrusion detection is gathering the network traffic. This can lead to the use of sensors rather than full-blown IDS devices. The sensors may have some limited intelligence, sending traffic back to a more centralized system that has more intelligence in it. Even when using sensors, there is still the challenge of getting the traffic to the IDS. Modern networks use switches. Switches send network traffic only to the target machine. This keeps systems isolated and network performance high. For an intrusion detection system to work, however, all network traffic needs to get to the IDS.

Fortunately, there are ways to accomplish this. Most switches have the ability to use something called a port span, span port, or port mirror. This means that you can specify a switch port to send

some portion of traffic or all traffic to, so that traffic that is being sent to one system also gets sent to the switch port where the IDS is installed.

While this is good, it can also be problematic. Even at a small scale, if you have ten 1G interfaces that are being mirrored to a single 1G port, you have a problem. Oversubscription works pretty well since most network connections are barely used, and you can get away with oversubscription in most cases. There comes a time, however, when you simply have too much data for the port you are trying to cram that data down. This is why determining the architecture ahead of time is important. Understanding what your network architecture looks like and what your utilization looks like will help determine the number of IDS devices you need in order to gather all data.

You don't have to span end ports, of course. You can span trunk ports, which are the ports that link switches to one another. In doing only the trunk ports, though, you have the issue of missing all of the host-to-host communication within the switch itself. Any communication that happens within the switch remains within the switch and never gets to the trunk port. The same is true with virtual environments where communications may remain within a virtual switch and never make it somewhere to be analyzed.

These are all problems that can be solved, however. They require planning and understanding of the infrastructure that is in place. They also require knowing exactly what it is you are looking for. It may not be necessary to monitor every single network connection within your network—more is not necessarily better. In fact, in this case, more can be decidedly worse since you'll just have made a lot more data that you need to look through. Finding the needle in the haystack isn't made easier by adding in more hay that may potentially include a needle. It could just simply be a lot more hay that you have to wade through.

In cases where you need to isolate a single device and you don't have access to the network infrastructure, you may be able to use a tap. These exist for both copper and fiber interfaces. A tap allows you to insert a device in the middle of the connection between the end system and the network device. Bro can be configured as a software tap if the host is configured to bridge two interfaces. This will then have connections that can be sent to the IDS device so network traffic gets to it without impacting any of the network infrastructure.

When it comes to monitoring IDS alerts, additional infrastructure may be necessary. We'll come to some of that in the next section, but as a start, a monitoring console may be useful. This allows an operator to get the alerts that the IDS is generating.

Alerting

The rules we have looked at provide an indication of what the alerts will look like. If the rules are well-constructed, the alerts will provide a good indication as to what has happened. There will be additional information in the alert aside from just the message that has been indicated in the rule.

Listing 8-11 shows a log sample. This particular log sample from Snort highlights the importance of good rules, and also of the challenge associated with false positives.

Listing 8-11: Snort Alert File

```
02/19-22:29:48.037186  [**] [1:100090:1] SYN scan [**] [Priority: 0]
{TCP} 192.168.86.65:57531 -> 192.168.86.83:80
02/19-22:29:48.091657  [**] [1:100090:1] SYN scan [**] [Priority: 0]
{TCP} 192.168.86.65:57546 -> 192.168.86.83:80
02/19-22:29:48.167276  [**] [1:100090:1] SYN scan [**] [Priority: 0]
{TCP} 192.168.86.65:57561 -> 192.168.86.83:80
02/19-22:30:11.142697  [**] [1:100090:1] SYN scan [**] [Priority: 0]
{TCP} 192.168.86.65:57576 -> 192.168.86.27:2869
02/19-22:33:09.277492  [**] [1:100090:1] SYN scan [**] [Priority: 0]
{TCP} 192.168.86.65:57593 -> 157.240.2.7:443
02/19-22:38:08.796005  [**] [1:100090:1] SYN scan [**] [Priority: 0]
{TCP} 192.168.86.65:57620 -> 157.240.2.7:443
```

This set of alerts came about through the use of a web scanner on a vulnerable web application. You can see that there are no alerts based on the active scanning, which should have included sending SQL injection messages, command injection messages, and other web-based attacks. These alerts suggest that there was a SYN scan taking place. The rule that triggered the SYN scan alert was just looking for a number of SYN messages in a short period of time. Because the web scanner was using a number of threads to run the scan, it appeared to Snort as though there was a SYN scan going on.

The alert messages you can see indicate that there was a SYN scan, which is the message that was set in the rule. Additionally, you get the Snort ID (SID), 1000090. This is definable by the person who wrote the rule. The protocol that was used to generate this alert was TCP, as indicated by the alert entry. Finally, you see the source IP address and port as well as the destination IP address and port. This is the output that is generated from the fast alert module in Snort. Entries from the `unified2` log module would look considerably different and also typically require additional scripts to generate useful output. This was just a text-based log file that was easily readable.

Summary

Intrusion detection systems can provide a lot of assistance when it comes to analysis of network communications. This includes both live captures where the IDS is running on the system where the packets are being captured, and also offline analysis where another system has performed the packet capture and the IDS is run against the packet capture file. This requires that packet capture software like `tcpdump` or `Wireshark` be run on the system where the analysis is needed. Adding intrusion detection software can be too much of a burden from a processing perspective, but packet

capture software may be lightweight enough to be able to capture packets for analysis on another system where the IDS software is running.

In addition to commercial IDS software, open source solutions are available as well. Snort has long been considered a de facto standard when it comes to network intrusion detection. In fact, the Snort rules are so widely known that other software packages like Suricata and Sagan have adopted usage of these rules. Each of these has their own benefits, of course, but the rules they use are the same as those used by Snort. Another open source IDS is Bro, which has the capability of using signatures like Snort does but is also an event-driven monitoring solution that requires a script to handle events and determine what to do about them. Bro doesn't use signatures in a traditional sense since what can be done in a Bro script could be more complicated and address more situations, including generating statistics. Snort also has the capability to have very complex signatures to detect very specific events, just as Bro is capable of. The difference is that the capability of doing something about the event and subsequent detection comes with Bro where Snort would require external capabilities to do anything further.

One consideration that should be addressed in the implementation of IDS, even on a temporary basis, is the location of the IDS within the network. It may be that you need to monitor several network connections at once and while that's possible, there is a potential for oversubscribing the network connection to the IDS if port spanning or port mirroring is used. This means that it is necessary to address the needs of the monitoring you are performing in order to determine whether you will have adequate network infrastructure to support your planned implementation.

9

Using Firewall and Application Logs

In this chapter, you will learn about:

- Using syslog and how to read syslog output
- Windows Event logs
- Different types of firewall logs

Sitting at his computer, he realizes that everything he sends across the wire has the potential to be logged somewhere. This may be at the firewall that he may be passing through transparently or it may be a result of the program he is connecting to logging his actions. The best thing to do is to not leave any traces at all, but that's not a realistic expectation. As a result, when he gets into a system, one thing he makes sure to do is to track down the logs. He has tools that are capable of cleaning out log files. The question he ponders is whether it's worth it, based on whether logs are being pushed off to another system for storage and analysis. If logs are being sent somewhere else, discrepancies might raise more questions. This means doing a little digging.

He looks at the processes that are running to see whether there are agents set up for some of the popular log management systems. He also checks the network connections to see if there is something that has escaped him in the process tables. Finally, he goes looking at the logs themselves just to see what sort of trail there may be, based on how much is being logged. Some businesses barely bother logging at all. Those businesses are much easier to deal with. Once he is satisfied there is little being logged, he is free to move on. Depending on the permissions he has managed to obtain, he may even turn down some of the logging before he really starts working.

One of the challenges with network forensic analysis is that anything that happens on the network is gone the moment it happens. The wires don't store anything and there are no remnants once packets have traversed the wires. This is completely unlike magnetic hard drives that may store information even after the sectors have been written over. Even solid state hard drives may hold onto information. Where older networks were sometimes store and forward, meaning they would hold onto information until the connection to the next hop came online, modern networks are real time. Nothing is left behind once the packet has passed through.

This makes performing an investigation on network-based attacks much harder, unless you have the infrastructure and storage capacity to store all of the traffic that has passed across all of the nodes in your network. This is where it's useful to have some additional help. While the complete packets will have passed through, other locations can provide correlating information that can help to determine what happened over the network.

Network communication doesn't happen in isolation. It's not like it starts nowhere, passes into the ether, and disappears. Applications are the beginning and the end of network communication—clients on one end and servers on the other. Where clients don't often generate log files, servers typically do. This includes the operating system where the server resides. Between the operating system logging network-related events and the applications logging connections and other important events, you can build up additional information to support the efforts of network investigations.

Both Windows and Linux provide subsystems that can be used by applications and the operating system. Windows systems use an event subsystem that is primarily seen by users in the Event Viewer. On the Linux side, there is a logging system called syslog. Although you will find different implementations of syslog, the underlying concepts are the same. With both of these system logging infrastructures, services and other applications can generate logs. These logs are managed by a single entity, meaning that they have a consistent structure and format, so the same tools can be used to investigate and analyze those logs.

Not every application and server makes use of the existing system logging infrastructure. Sometimes, a service like a web server may use its own log format and logging system. Other services that generate logs that can be beneficial in a network forensic analysis are firewalls. Firewalls come in many forms, including what some may think of as traditional firewalls that block packets. Other firewalls may be focused on applications, like web transactions. This may be a proxy server, a form of application layer gateway. A web application firewall (WAF) is another form of firewall.

All of these log systems include different formats for the log files. While there may be different needs when it comes to analyzing the logs, one advantage to log files is they are often just text-based. This means that if you know what you are looking for, you can read the logs with your eyes rather than needing to look for a tool that can read the logs for you. This is not always the case, however.

Syslog

Syslog started life as part of an electronic mail system called Sendmail. The developer of Sendmail created a logging system to go along with it. The logging system was useful enough that it began to be adopted by others and is now the de facto logging standard on Unix-like operating systems, including Linux. This does not mean that the implementation has remained the same over the decades since syslog was created. Syslog has been implemented over multiple types of Unix by a number of different vendors. In the time that Linux has been around, there have been different syslog implementations, including rsyslog and syslog-ng.

Although it was originally developed as part of Sendmail, syslog became standardized. In 2001, it was documented in RFC 3164, which has since been obsoleted by RFC 5424. Syslog is a text-based

logging system, and each log entry has two pieces. The first is what the syslog server itself generates. The second part is what is provided by the application. Additionally, syslog provides the ability to generate multiple log files. It is able to do this because syslog isn't just a service that takes messages from an application and writes out timestamped logs. The syslog standard defines both facilities and severities. The facilities are used to determine which log to write the message out to, or whether to do anything with it at all. The severity can be used to determine whether or not the log message should be written.

Although the facility is determined by the application generating the log message, the facilities are designed to indicate the area of functionality the log message belongs in. This helps to isolate related messages into one place. With related log messages in one place, you don't have to dig through a lot of extraneous entries to find the information that is most relevant to you. Table 9-1 shows the facilities that have been defined for syslog in RFC 5424, though they are essentially the same as those defined in RFC 3164.

Table 9-1: Syslog Facilities

Code	Facility
0	Kernel
1	User
2	Mail
3	Daemons
4	Security/Authorization
5	Internal to the syslog daemon
6	Line printer (lpr)
7	Network news (nntp)
8	Unix-to-Unix Copy (UUCP)
9	Clock
10	Security/Authorization
11	FTP
12	NTP
13	Log audit
14	Log alert
15	Clock daemon (formerly cron in RFC 3164)
16–23	Local use 0–7 (local0–local7)

In addition to facilities, syslog uses severities to categorize messages. Even within a facility, you can determine how much you want to see in your log files. Applications may generate a number of levels of messages, depending on what is happening within the application. Starting up, the application may send an Informational message just to provide a marker as to when the application started up. Table 9-2 shows the severities that are defined for syslog.

Table 9-2: Syslog Severities

Code	Severity
0	Emergency
1	Alert
2	Critical
3	Error
4	Warning
5	Notice
6	Informational
7	Debug

According to RFC 5424, syslog defines a priority as being the facility code multiplied by 8 then adding the severity code. The lowest number has the highest priority. If the kernel fails and generates an emergency message, you'd have a 0 if you multiply the 0 for the kernel message by the 0 for the emergency. In fact, any emergency or any kernel message would automatically give you a priority of 0 because anything multiplied by 0 is 0. As we work further into messages, you can end up with a local7 debug message and that would have a priority of 191. The priority is not included in the log files themselves. The priority is calculated as above. In the case just noted, I know that the facility is local7, which has a value of 23 so to get the priority, I multiply 23 by 8 and then add 7, which is the value for a debug message. The values used came from the syslog RFC.

Because no standards are defined for what files are used for any message, you would rely on the configuration file for the syslog implementation on your system. As with other configuration files on a Unix-like operating system, you would find the syslog configuration in the `/etc` directory. The exact file will depend on the specific implementation of syslog used. Listing 9-1 shows part of the configuration from rsyslog on an Arch Linux system. The different implementations of syslog do establish default locations because of the configuration file that is packaged with the software. However, that doesn't mean that files or even locations can't be altered easily by way of the configuration file. This means that even two systems that are using rsyslog may have different files in use.

Listing 9-1: rsyslog.conf

```

auth,authpriv.*          /var/log/auth.log
*.*;auth,authpriv.none   -/var/log/syslog
#cron.*                   /var/log/cron.log
daemon.*                  -/var/log/daemon.log
kern.*                    -/var/log/kern.log
lpr.*                     -/var/log/lpr.log
mail.*                    -/var/log/mail.log
user.*                    -/var/log/user.log

mail.info                 -/var/log/mail.info
mail.warn                 -/var/log/mail.warn
mail.err                  /var/log/mail.err

news.crit                 /var/log/news/news.crit
news.err                  /var/log/news/news.err
news.notice               -/var/log/news/news.notice

```

Note that all of the log files are in the `/var/log` directory. Though it is not required, it is a convention that makes log files easier to locate on an unfamiliar system. The Filesystem Hierarchy Standard (FHS) is used on the majority of Linux and other Unix-like systems. If you are working on a Unix-like system, you can generally be sure that the log files, regardless of what is being logged, will be in `/var/log`.

The notation used here is one of facility.severity. In cases where you see an `*` being used, it is a wildcard that matches anything. The line that says `lpr.*` indicates that any severity from the `lpr` facility will be logged in `/var/log/lpr.log`. This indicates that everything being generated on that facility will be logged. You can be even more granular than that, however. As an example, `mail.info` messages are being logged into `/var/log/mail.info`. All informational messages on the mail facility are being logged into that file. Error messages in the mail system are logged to a different file. For ease of quick understanding, the human-readable versions of the numeric codes are used. If all you saw was numbers in the file, it would take more time to determine where the information you wanted was being kept. Of course, if you look at the `/var/log` directory on most systems, you will get a pretty good idea of what you are looking for.

You will see that the filenames are generally descriptive. Mail messages are stored in files named with some variation of “mail.” User-related messages on the user facility are stored in the file `user.log`. You could have guessed that just based on the name of the file. What you don’t get is what is being logged. By default, many of these are set to log everything that comes in. If you have a very active system, this can consume disk space and you may be able to limit what is being logged. In the case of a forensic investigation, though, you may want to see more in the way of messages than less.

Centralized Logging

One challenge when it comes to investigating systems that are either desktops or end-user reachable servers is that logs may have been altered. If an attacker or malicious user gains access to the system, meaning they can run programs on the hardware, they may be able to tamper with log files. One potential solution for this is to introduce a centralized log host. The central log server listens for messages from other systems in the network, storing them. This will help reduce the potential for tampering because you don't have to rely on the logs on the endpoint being the only logs available. If logs are also stored on the endpoint, they can be checked for validity against those stored on the centralized log system.

There are a number of ways to establish a centralized log system. One of the easiest is to just use syslog. In addition to specifying how a log system may work with organization and messages, syslog also includes the potential for sending messages over the wire. This includes specifications on the sending and receiving side. Initially, syslog was sent over UDP for the transport layer. UDP was faster and had less overhead, and since most syslog messages may typically have been sent over the local network, or at least within the enterprise, it was seen as less likely that messages would be lost. Modern implementations may be sent over longer distances and log messages are considered, perhaps, more important. As a result, syslog supports transport over both TCP and UDP. Regardless of the transport protocol used, the port number is 514.

The syslog server we have been using, rsyslog, can just as easily be configured to both receive and send log messages. The rsyslog package uses modules to enable functionality, and in order to enable receiving messages, we need to enable the modules that provide that functionality. Both TCP and UDP have their own modules. The configuration file fragment shown in Listing 9-2 enables that functionality.

Listing 9-2: rsyslog Configuration for Receiving Log Messages

```
# provides UDP syslog reception
module(load="imudp")
input(type="imudp" port="514")

# provides TCP syslog reception
#module(load="imtcp")
#input(type="imtcp" port="514")
```

The configuration shows that the module for UDP has been enabled. The lines for TCP are there but are commented out, meaning they don't get read by rsyslog. They are only there as notes for someone working on the configuration file who may later want to enable the functionality. The first line related to UDP tells rsyslog to load the module named imudp. The second tells rsyslog to

accept input over UDP on port 514. Though this is the default port, you can change this parameter to run over a different port, if you prefer. Similarly, with the TCP connections, you can specify a port you want to listen on. This is something to keep in mind, though. If you use something other than the default port, the client end of the connection will need to be similarly configured, otherwise the server won't receive the messages from the client.

Once you have a syslog server set up, you can start receiving messages from different systems. Unless you do something different, however, all of the messages will be dumped into the files as indicated in Listing 9-2. It may be useful to configure a template that uses some variables to make sure messages from each separate host end up in their own files. You can use the lines in Listing 9-3 to make sure that logs from different locations end up in different places.

Listing 9-3: rsyslog Configuration for Remote Systems

```
$template Msgs, "/var/log/%fromhost%/%programname%.log"
*. * ?Msgs
```

The first line defines the template using the variables `%fromhost%` and `%programname%`. In place of `fromhost`, you could use the similar variable `fromhost-ip`, which always uses the sending host's IP address rather than whatever hostname may be configured. Similarly, you could use `hostname` or `source`. The difference between these two and the two mentioned before is that if you have multiple syslog systems that are forwarding, the `fromhost` and `fromhost-ip` variables are the ones that are just before the host you have this configuration on. In other words, they will provide the information from the host that the log message was received from, rather than the host the message was generated on. This may be an important distinction if you have multiple forwarding syslog servers in your messaging chain.

NOTE While I have referred to them as *variables*, because they are used like a variable name might be used in a script or program, rsyslog refers to them as *properties*. rsyslog keeps track of a number of other properties that could be used in templates.

Once you have your syslog server set up, you need to enable the client to forward messages to the server. This is easy to do. You can also enable syslog servers to not only forward all log messages, but also have them log locally. This is useful if your log server becomes unreachable for some reason. If your log server goes down, you may lose those messages during the period of time the server is unreachable unless you have local logging enabled. So, in addition to the entries in Listing 9-3 indicating where to log locally, you may have entries like those in Listing 9-4 to forward messages to a log server.

Listing 9-4: rsyslog Configuration for Remote Logging on Clients

```
*.* @192.168.86.20:514
*.* @@192.168.86.10:514
```

The first line with the single @ sign indicates that the transport protocol should be UDP, which is the default transport. If you want to use TCP instead, you need to use a second @ sign to indicate that. You will also note that there is a :514 at the end of each of these lines. This indicates the port. You would need to use whatever port you had specified when you were setting up your server. While the entries here use IP addresses, you can also use hostnames. There are trade-offs for each of these choices. The IP address means you can log even if DNS, for whatever reason, is down or unreachable. The hostname means you don't have to reconfigure all of your syslog files if you happen to change the IP address where your syslog server is at some point.

Syslog is not the only possibility for a central syslog server. Another possibility is NXLog, which is a log management package offered by IBM. There is a community edition in addition to the commercial offering. An example of an NXLog configuration file is shown in Listing 9-5.

Listing 9-5: NXLog Configuration File

```
#####
# Global directives                                     #
#####
User nxlog
Group nxlog

LogFile /var/log/nxlog/nxlog.log
LogLevel INFO

#####
# Modules                                             #
#####
<Extension _syslog>
    Module      xm_syslog
</Extension>

<Input in1>
    Module      im_udp
    Port        514
    Exec        parse_syslog_bsd();
</Input>

<Input in2>
    Module      im_tcp
    Port        514
</Input>
```

```

<Output fileout1>
  Module    om_file
  File      "/var/log/nxlog/logmsg.txt"
  Exec      if $Message =~ /error/ $SeverityValue =
  syslog_severity_value("error");
  Exec      to_syslog_bsd();
</Output>

<Output fileout2>
  Module    om_file
  File      "/var/log/nxlog/logmsg2.txt"
</Output>

#####
# Routes                                     #
#####
<Route 1>
  Path      in1 => fileout1
</Route>

<Route tcproute>
  Path      in2 => fileout2
</Route>

```

In addition to being able to handle syslog as a central log host, NXLog has a number of other capabilities. You can see where multiple inputs are specified. The inputs specified are for both TCP and UDP listeners for syslog messages. One thing NXLog offers is the ability to add functions to each definition. This allows for special handling to be defined on inputs and outputs. Since it's a log management solution, NXLog also provides the ability to consume non-syslog messages in order to store everything into a single place. The outputs are specified in the configuration file as well. You can specify log files to write to or you could also send messages to other log hosts. You can have NXLog installed on your remote systems, collect all of the logs, whether they are syslog or otherwise, and send them off to a central, managed log system. This central, managed log system could be running NXLog or it could be using some other log service.

Another feature NXLog offers is routing. This would allow you to have a normal configuration for regular operations and a separate route for logs that are related to forensic investigations. The ability to quickly create custom routing for log messages with some or all of the configured inputs is one of the benefits of using a log management system over just a basic syslog installation.

Of course, NXLog is not the only log management system that's available. There are others as well, including Splunk, which has a lite version for small installations. This is akin to the community version of NXLog. Also, you can use the Elastic Stack, which is sometimes called ELK. (The name has changed from ELK, which was an acronym for Elastic, Logstash, and Kibana, a collection of programs that were used for log management.) The different components are used for search, acquisition and storage, and a user interface that can be used for management or data visualization. Splunk similarly offers a user interface component that can be used to search and visualize the data you are collecting.

Reading Log Messages

Getting all of your log messages safely to a central log host is one thing, but it's meaningless unless you know what you are looking at. Fortunately, syslog messages are in plain text. While they are not designed to be read like a book, they are at least not stored in a binary format. Nor are they stored in a more complicated format like Extensible Markup Language (XML) or JavaScript Object Notation (JSON). Although both of those are also plain text, they contain a lot of overhead describing the data, which can make it harder to quickly parse visually. That's not to say that you can't parse XML or JSON, but there is just more text to look at. Syslog messages are single lines and if you know what you are looking at, they are easy to read. You can see an example of log entries in Listing 9-6.

Listing 9-6: Syslog Messages

```
Mar 12 17:09:01 zulu CRON[5296]: (root) CMD ( [ -x
/usr/lib/php/sessionclean ] && /usr/lib/php/sessionclean)
Mar 12 17:17:01 zulu CRON[5352]: (root) CMD ( cd / && run-parts
--report /etc/cron.hourly)
Mar 12 17:36:25 zulu systemd[1]: Time has been changed
Mar 12 17:36:25 zulu systemd[1]: snapd.refresh.timer:
Adding 3h 27.866580s random time.
Mar 12 17:36:25 zulu systemd[1]: apt-daily.timer:
Adding 5h 42min 4.381482s random time.
Mar 12 17:36:25 zulu systemd[5610]: Time has been changed
Mar 12 17:39:01 zulu CRON[5360]: (root) CMD
( [ -x /usr/lib/php/sessionclean ] && /usr/lib/php/sessionclean)
Mar 12 18:01:08 zulu systemd-timesyncd[34299]: Synchronized to time
server 91.189.94.4:123 (ntp.ubuntu.com).
```

Let's take the first two lines since they are similar. The first part of each line is the timestamp, including the date. This is followed by the name of the system the log was created on, `zulu`. After that is the name and process ID of the program that created the message. In both cases, the message was created by `cron`, which is a program used to schedule tasks. The process ID is different from one to the other, meaning the `cron` process ran twice, starting up a different set of tasks.

The next part of the message is what the application has created for the message. What both of these messages say is that the root user ran a set of tasks. The first one was `sessionclean`, related to the PHP programming language and web programming in particular. The second line says that `cron` was asked to run a report on the tasks it is configured to run hourly.

Looking at the remaining lines, you can see that `systemd` produced some messages, `cron` produced another message, and finally `systemd-timesyncd` generated a message relating to synchronizing the clock with a time server. These messages came out of the syslog file that was used on the system the logs were collected on for most log messages. Not all log messages are logged to the syslog file, though. Specifically, authorization messages are logged to a different file on this system. You can see an example of the sorts of messages that are in an authorization log in Listing 9-7.

Listing 9-7: Auth.log Messages

```

Mar 12 17:00:12 zulu sudo: pam_unix(sudo:session): session closed for
user root
Mar 12 17:00:50 zulu sshd[5179]: pam_unix(sshd:auth): authentication
failure; logname= uid=0 euid=0 tty=ssh ruser=
rhost=192.168.86.65 user=kilroy
Mar 12 17:00:52 zulu sshd[5179]: Failed password for kilroy from
192.168.86.65 port 52820 ssh2
Mar 12 17:00:57 zulu sshd[5179]: Accepted password for kilroy from
192.168.86.65 port 52820 ssh2
Mar 12 17:00:57 zulu sshd[5179]: pam_unix(sshd:session): session opened
for user kilroy by (uid=0)
Mar 12 17:00:57 zulu systemd-logind[4338]: New session 123 of user
kilroy.
Mar 12 17:01:14 zulu sudo: kilroy : TTY=pts/0 ; PWD=/home/kilroy ;
USER=root ; COMMAND=/usr/bin/apt install modsecurity
Mar 12 17:01:14 zulu sudo: pam_unix(sudo:session): session opened for
user root by kilroy(uid=0)
Mar 12 17:01:14 zulu sudo: pam_unix(sudo:session): session closed for
user root
Mar 12 17:01:19 zulu sudo: kilroy : TTY=pts/0 ; PWD=/home/kilroy ;
USER=root ; COMMAND=/usr/bin/apt install mod-security
Mar 12 17:01:19 zulu sudo: pam_unix(sudo:session): session opened for
user root by kilroy(uid=0)
Mar 12 17:01:20 zulu sudo: pam_unix(sudo:session): session closed for
user root
Mar 12 17:01:59 zulu sudo: kilroy : TTY=pts/0 ; PWD=/home/kilroy ;
USER=root ; COMMAND=/bin/cat /var/log/apache2/modsec_audit.log

```

The format is the same here, though the content is different. The entries you can see may be authorization-related, but they all come from different applications and subsystems. The first entry, for example, comes from `sudo`, which is the program that is used to get administrative access to the system. Sudo makes use of `pam_unix`. PAM is the pluggable authentication modules library on Unix-like operating systems that performs a number of functions related to authorization and authentication.

Other entries are from the secure shell daemon (`sshd`). The second and third entries (time stamps 17:00:50 and 17:00:52 respectively) appear to be the result of a mistyped password since the fourth entry indicates that a password was accepted for that user. Knowing what PAM and `sudo` are helps to make the rest of the log entries much more easily understandable. The `sudo` entries relate to running specific programs that need elevated privileges. You can see the programs and the arguments needed by the program in the `COMMAND=` section of the log entry. The same log entry also indicates what user called `sudo`. Additionally, since `sudo` not only allows users to run with effective root permissions, you can see the user whose context the program is running in. In these cases, it's always root, but `sudo` may allow for any program to be run as a separate user.

As indicated while looking at the configuration in the “Syslog” section earlier in this chapter, different log files will contain entries from different applications for different purposes. The basic

structure of the syslog files is the same. Once you get used to how the log entries look, you can read any of them. This doesn't always mean you will be able to understand what's going on, because each application generates its own messages, and sometimes the log message is only useful to the developer of the application. That's not the ideal circumstance, but the log entries are created by the software developer. They are not generally designed to be read by end users.

LogWatch

Log management systems let you query the log data you have available to you. If you just have a small setup, you may not need something as elaborate or complex as a log management system. However, the need to keep an eye on events is still important, particularly if what you are investigating is ongoing. Even in the case of smaller setups, you may need help keeping an eye on logs and extracting important details. One program you can use is called LogWatch, which is a log analyzer and reporter.

LogWatch can be configured with a number of options to look at log files and services. It will present a report when you run it with output you have specified. An example of what you can see when you run LogWatch is shown in Listing 9-8.

Listing 9-8: LogWatch Output

```
##### Logwatch 7.4.0 (03/01/11) #####
Processing Initiated: Fri Mar 17 17:34:28 2017
Date Range Processed: yesterday
                      ( 2017-Mar-16 )
                      Period is day.
Detail Level of Output: 10
Type of Output/Format: stdout / text
Logfiles for Host: boingers.washere.com
#####

----- System Configuration Begin -----

CPU:      2 AMD FX(tm)-8350 Eight-Core Processor at 4012MHz
Machine: x86_64
Release: Linux 3.10.0-514.10.2.el7.x86_64
Total Memory: 3791 MB
Free Memory: 3037 MB

----- System Configuration End -----

----- clam-update Begin -----

No updates detected in the log for the freshclam daemon (the
ClamAV update process). If the freshclam daemon is not running,
you may need to restart it. Other options:
```

- A. If you no longer wish to run freshclam, deleting the log file (default is freshclam.log) will suppress this error message.
- B. If you use a different log file, update the appropriate configuration file. For example:

```
echo "LogFile = log_file" >>
/etc/logwatch/conf/logfiles/clam-update.conf
```

 where log_file is the filename of the freshclam log file.
- C. If you are logging using syslog, you need to indicate that your log file uses the syslog format. For example:

```
echo "*OnlyService = freshclam" >>
/etc/logwatch/conf/logfiles/clam-update.conf
echo "*RemoveHeaders" >>
/etc/logwatch/conf/logfiles/clam-update.conf
```

----- clam-update End -----

----- Disk Space Begin -----

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/mapper/cl-root	23G	1.4G	22G	6%	/
devtmpfs	1.9G	0	1.9G	0%	/dev
/dev/sda1	1014M	184M	831M	19%	/boot

LogWatch will provide summaries of the events of the last day and typically runs as a scheduled job with the output being mailed to a user. This may be the root user or anyone else who needs to see the output from the program. Having the summaries can be very beneficial because it's a lot easier to look at a summary of the Linux firewall logs or the mail activity than it is to have to try to read and search without any idea where to begin. Having the summary statistics and some of the details in aggregate will provide a good starting point. LogWatch can be a very valuable tool that can point out problems you may not have otherwise seen.

The e-mail logs can also be problematic to look through. If you were being attacked with spear phishing, the rejected messages from guessed-at addresses may get lost in the noise of all the other traffic that is going through your mail system. Using a tool like LogWatch, you will see that a large number of messages were rejected by the mail system. This may be a warning that something is going on that you should pay attention to. You may also be able to track attacks on a web server since LogWatch will present all of the errors that were triggered. A large number of errors showing up may indicate that an attack is going on and it may require additional investigation. The LogWatch output will provide enough detail that you can find the specific log entries in order to find a timestamp. The timestamp will provide you a reference point to see what else was going on at that point. Having your times synchronized across your systems helps the process of reconstructing events. You'll see this point echoed several times throughout this book because of its importance.

Event Viewer

Where syslog runs on Unix-like systems as the de facto logging system, on Windows systems, you can count on the Event Logs and being able to view them using the Event Viewer. The logs that are available on Windows systems fall into two categories: Windows logs and applications and services logs. The Windows logs are grouped into three primary categories: Application, Security, and System. Two other log categories were added in recent versions of Windows: setup and forwarded logs. The three primary categories have been around for as long as there have been Windows logs. The Application log contains information related to applications. The Security log contains auditing information, logon information, and events related to resource utilization. The System log contains events related to the operating system itself. This might be messages from device drivers, for instance.

Since Windows Vista, the logs have been stored in XML. You can view the XML directly through the Windows Event Viewer, as shown in Figure 9-1. Looking at the details of any event in the Event

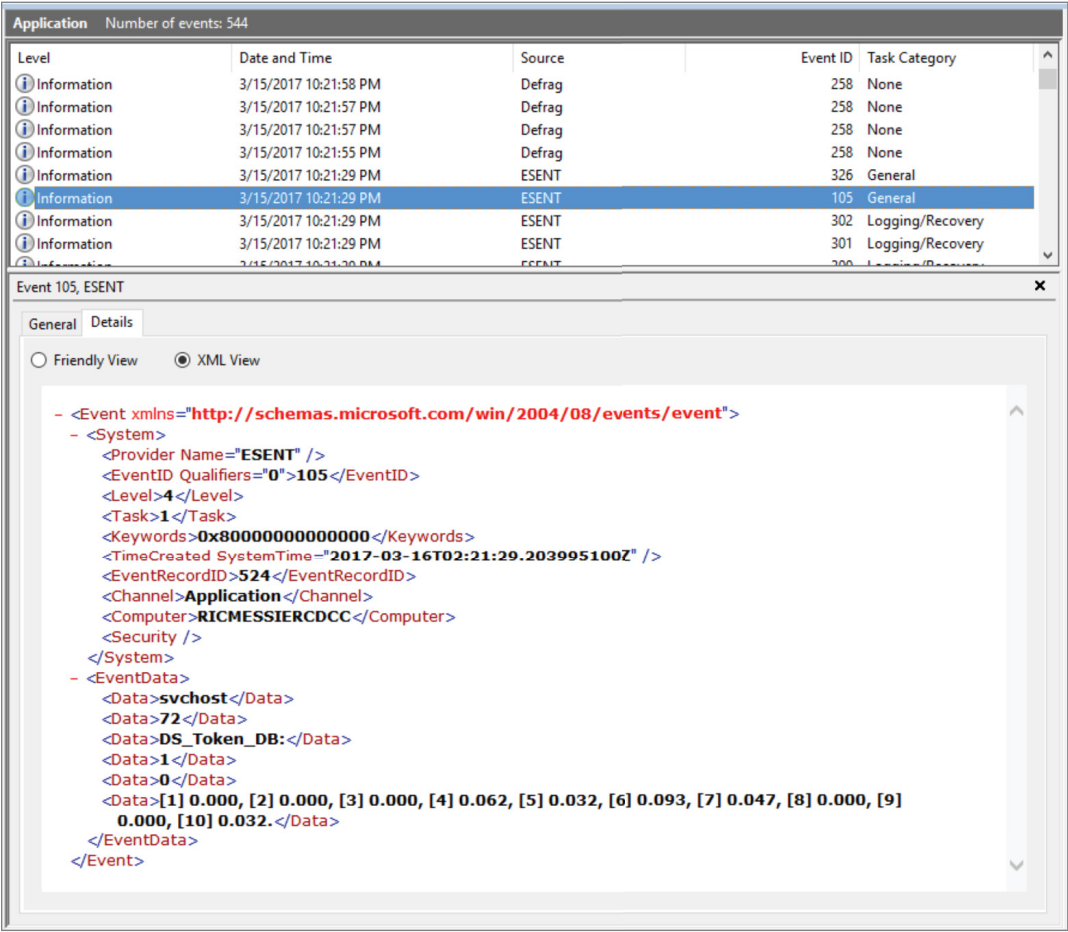


Figure 9-1: Windows Event Viewer displaying XML.

Viewer, you can select the XML view and see the event in its XML form. The XML is read and the information in it is presented in the UI without all of the metadata describing each field.

If we were to look at the same event in the UI without looking at the details, you would see what is in Figure 9-2. This presents the information in a way that you can parse quickly. There are headers for all of the fields, bracketed by <>, which can make the information self-documenting. Because the information is spread out, the information may be easier to take in. XML can be a dense way to present a lot of information and if you are less familiar with it, it can be harder to determine what information relates to what else.

The five categories of Windows logs are events that have system-wide impact. Windows enables other applications to generate log data that doesn't have any impact on the operating system and may not be critical to the system overall. The Applications and Services Logs are where those logs go. Different categories of information can be stored in Applications and Services Logs. In Figure 9-3, you can see Hardware Events, Internet Explorer, Key Management Service, ThinPrint Diagnostics, and Windows

The screenshot shows the Windows Event Viewer interface. At the top, a header bar indicates 'Application' and 'Number of events: 544'. Below this is a table listing several events. The selected event is Event 105, ESENT, which is highlighted in blue. The details pane for this event is open, showing the 'General' tab. The event description reads: 'svchost (72) DS_Token_DB: The database engine started a new instance (1). (Time=0 seconds)'. Below the description is an 'Internal Timing Sequence' with 10 data points. At the bottom of the details pane, there is a summary section with fields for Log Name, Source, Event ID, Level, User, OpCode, and More Information.

Level	Date and Time	Source	Event ID	Task Category
Information	3/15/2017 10:21:58 PM	Defrag	258	None
Information	3/15/2017 10:21:57 PM	Defrag	258	None
Information	3/15/2017 10:21:57 PM	Defrag	258	None
Information	3/15/2017 10:21:55 PM	Defrag	258	None
Information	3/15/2017 10:21:29 PM	ESENT	326	General
Information	3/15/2017 10:21:29 PM	ESENT	105	General
Information	3/15/2017 10:21:29 PM	ESENT	302	Logging/Recovery
Information	3/15/2017 10:21:29 PM	ESENT	301	Logging/Recovery
Information	3/15/2017 10:21:29 PM	ESENT	300	Logging/Recovery

Event 105, ESENT

General Details

svchost (72) DS_Token_DB: The database engine started a new instance (1). (Time=0 seconds)

Internal Timing Sequence: [1] 0.000, [2] 0.000, [3] 0.000, [4] 0.062, [5] 0.032, [6] 0.093, [7] 0.047, [8] 0.000, [9] 0.000, [10] 0.032.

Log Name: Application
Source: ESENT
Event ID: 105
Level: Information
User: N/A
OpCode:
More Information: [Event Log Online Help](#)

Logged: 3/15/2017 10:21:29 PM
Task Category: General
Keywords: Classic
Computer: RICMESSIERCDDC

Figure 9-2: Windows Event Viewer displaying details.

PowerShell. Additionally, there is another folder you can see labeled Microsoft. Inside this folder are all the various Windows applications and components. Each of them will have their own event log.

Four subtypes of logs are stored within the Applications and Services logs. These are Admin, Operational, Analytic, and Debug. You can see one Windows component that has logs for two of these subtypes in Figure 9-4. On the left-hand side of the screen capture, under DHCPv6-Client, you can see two logs. One of them is the operational log and the other is the admin log. In the right-hand side of the screen, under Log Name, it says Microsoft-Windows-DHCPv6 Client Events/Admin. This

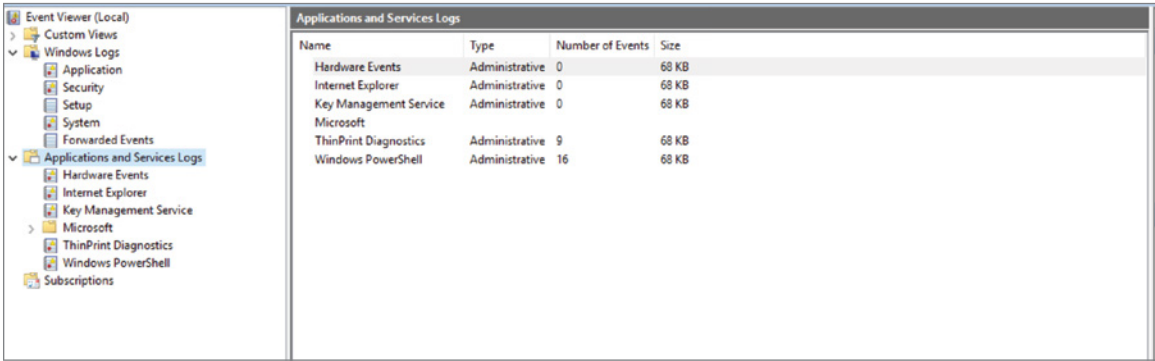


Figure 9-3: Windows Event Viewer categories.

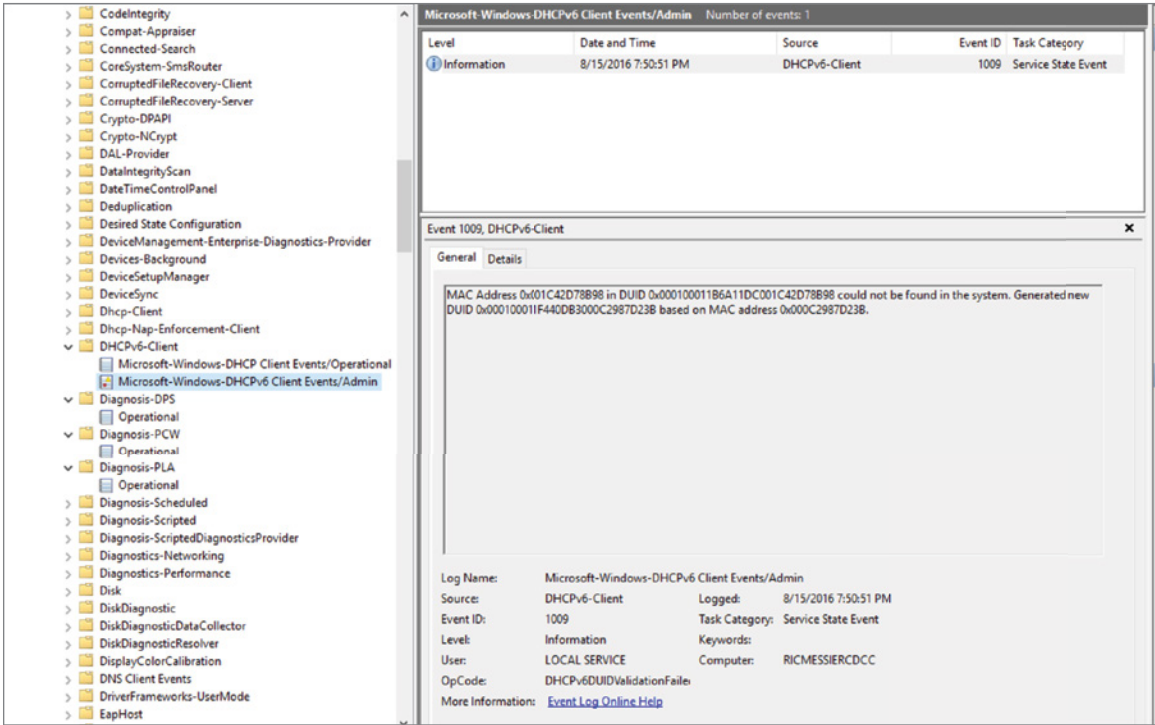


Figure 9-4: Windows Event Viewer applications and services.

indicates that underneath the DHCPv6 client, there are two subtypes of logs. The additional log detail is presented in the UI just as the other logs. You can also see the XML view with the Applications and Services logs, just as you can with the Windows Logs. Underneath the presentation of the data, the storage is taken care of by the event subsystem.

The log files themselves are stored in `\Windows\System32\WinEvt\Logs` but you won't be able to just pop them open in a text editor. While they are presented in XML format, they are stored on disk in a binary format in which the XML resides. Essentially, they are stored in a database. This makes it much easier to filter and query the information in the log files.

Querying Event Logs

Unlike the raw text files that syslog uses, Microsoft's log files are stored in such a way that they can be filtered and queried. You can certainly create scripts to dig through syslog files and some are available. You can also use tools like LogWatch to help aggregate information to present it in a more easily digested format. When data is aggregated, problems can jump out more easily. Windows Event Logs are slightly harder because they are not stored in a plaintext-based format so you can't just use common text editing and search tools. There are tools, however, that you can use to search through Event Logs. The first is built into the Event Viewer and can be used to search through Event Logs on the system they were generated on. This is the default behavior. When you open Event Viewer, it will display all of the log files on the system you are running it on. However, you can also use Event Viewer to connect to another system and open log files there, if you can provide appropriate credentials. Another option is to open saved files, which would allow you to open files that had been created on another system.

Once you have your log files open, you can use the Create Custom View feature to essentially query the database and show only what you believe may be relevant. Figure 9-5 shows the Create Custom View dialog box. Once you have selected which logs you want to query from the pulldown menu, you can select keywords from a second pulldown. This is not a freeform set of keywords. Instead, you are presented with keywords from the logs themselves. For instance, you can select Audit Success or Audit Failure, which would be log entries if you had auditing enabled on the system. Create Custom Query view also allows you to limit your search to just the levels you are interested in.

You can narrow your search quite a bit further than that, though. You can choose to select by log from the categories we discussed earlier, but you can also choose to select by source. If you choose to view by source, you can select components or applications that would have generated the log events. As an example, if you were only interested in messages from the SMBClient, you could select just that source. Further, you could say you were only interested in viewing messages from the last 12 hours, or the last week. You can limit the span of time you are looking through to better limit the amount of data you would be digging through.

Windows Events all have IDs, and each ID is related to a specific event. For example, when the Software Protection Service stops, it generates Event ID903. This will be consistent so if you see Event ID 903, you won't even need to look at the text because it will say that the Software Protection

Service has stopped. This may also help you limit the view that you are creating. If you have already seen something and you know what the Event ID is, you can create a view based on that Event ID and see all instances of that Event ID to determine when it had happened before.

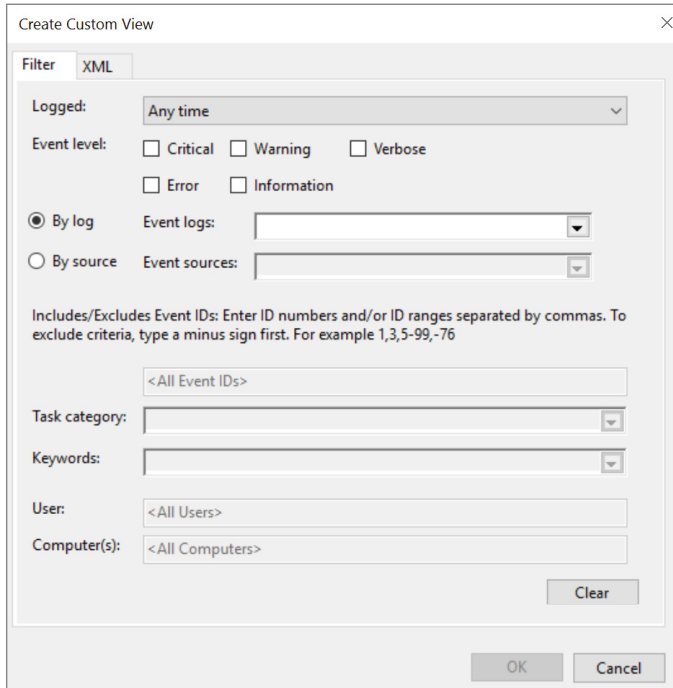


Figure 9-5: Event Viewer Create Custom View dialog.

In addition to being able to create a custom view that would allow you to save the view and load it any time you wanted, you can also perform a filter of the existing logs. This is done on the fly, though the process is the same. If you create a filter, you are presented with a dialog box that looks the same as that shown in Figure 9-5. You have all the same options as you would if you were to create a custom view. The difference is that you are doing a filter on data that you may only look at once. You may never need to do the same filter again. A custom view would be a specific way of looking at the logs that you would expect to re-use. You would save a view to be able to re-apply it when you wanted or needed. A filter is something you do in the moment if you don't think you would need to use it again.

You don't have to resort to using the Event Viewer to investigate log files, though. For a start, you could write a PowerShell script, making use of `Get-Eventlog`. You can also use a command-line utility, `wevtutil`. This program, installed on current versions of Windows, can be used to do a lot of work with the Windows Event Logs. As one example, you can get information about the log using the `get log information (g|i)` command, as shown in Listing 9-9. This command will tell you when the log file was created, last accessed, last written to, and how large it is.

Listing 9-9: wevtutil Get Log Information

```
C:\Windows\System32\winevt\Logs>wevtutil gli Application
creationTime: 2016-08-15T23:47:28.194Z
lastAccessTime: 2016-08-15T23:47:28.194Z
lastWriteTime: 2017-03-16T02:01:10.086Z
fileSize: 1118208
attributes: 32
numberOfLogRecords: 626
oldestRecordNumber: 1
```

This is not all you can do with wevtutil, however. You can directly query the Event Logs, though it takes some getting used to the syntax for the queries, because they are not straightforward. They are based on XPath. XPath is a way of querying XML documents. Since the Windows Event Logs are stored in XML, you are querying an XML document, so you need to learn a little XPath in order to generate queries that will bear fruit. A very simple one that will return all of the instances of EventID 1003 is shown in Listing 9-10 as well as the first result.

Listing 9-10: wevtutil Query for EventID 1003

```
C:\Windows\System32\winevt\Logs>wevtutil qe Application
/q:"*[System[(EventID=1003)]]" /f:text
Event[0]:
  Log Name: Application
  Source: Microsoft-Windows-Security-SPP
  Date: 2016-08-15T19:47:40.390
  Event ID: 1003
  Task: N/A
  Level: Information
  Opcode: N/A
  Keyword: Classic
  User: N/A
  User Name: N/A
  Computer: RICMESSIERCDCC
  Description:
The Software Protection service has completed licensing status check.
Application Id=55c92734-d682-4d71-983e-d6ec3f16059f
Licensing Status=
1: 040fa323-92b1-4baf-97a2-5b67feafeddb, 1, 0 [(0 [0xC004F014, 0, 0],
[(?)(?)(?)(?)(?)(?)(?)(?))(1 )(2 )(3 )]
```

You can see that we use the `qe` (query event) command to wevtutil. Then we provide the query string. In this case, we are just saying we want all instances of EventID 1003 from the Application log. We additionally have to specify that we want the output in a text format. Without that, you get

XML-formatted output, which is slightly more difficult to parse. Listing 9-11 shows an example of the same query shown in Listing 9-10 without the instruction to output in text.

Listing 9-11: wevtutil Query Without Text Format

```
C:\Windows\System32\winevt\Logs>wevtutil qe Application
/q:"*[System[(EventID=1003)]]"
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'>
<System><Provider Name='Microsoft-Windows-Security-SPP'
Guid='{E23B33B0-C8C9-472C-A5F9-F2BDFEA0F156}' EventSourceName='Software
Protection Platform Service'/><EventID Qualifiers='16384'>
1003</EventID><Version>0</Version><Level>4</Level>
<Task>0</Task><Opcode>0</Opcode><Keywords>0x80000000000000</Keywords>
<TimeCreated SystemTime='2016-08-15T23:47:40.390190200Z'/>
<EventRecordID>7</EventRecordID><Correlation/><Execution ProcessID='0'
ThreadID='0'/><Channel>Application</Channel><Computer>RICMESSIERCDC
</Computer><Security/></System><EventData>
<Data>55c92734-d682-4d71-983e-d6ec3f16059f</Data><Data>
1: 040fa323-92b1-4baf-97a2-5b67feafddb, 1, 0 [(0 [0xC004F014, 0, 0],
[(?)(?)(?)(?)(?)(?)(?)(?))(1 )(2 )(3 ))]
2: 0724cb7d-3437-4cb7-93cb-830375d0079d, 1, 0 [(0 [0xC004F014, 0, 0],
[(?)(?)(?)(?)(?)(?)(?)(?))(1 )(2 )(3 ))]
3: 0cdc4d08-6df6-4eb4-b5b4-a373c3e351e7, 1, 0 [(0 [0xC004F014, 0, 0],
[(?)(?)(?)(?)(?)(?)(?)(?))(1 )(2 )(3 ))]
```

This is the same query but it lacks the `/f:text` at the end. The output contains the same data as that in Listing 9-10 but it's in XML format. While all of the data is there, it isn't as easy for the eye to just gravitate toward the information we may be looking for. You can read it but it's not as easy to quickly parse visually for most people as the formatted view in Listing 9-10.

Microsoft also provides APIs for programmatically interacting with the Event Log subsystem. There are libraries you can get for Python to write scripts to interact with the Windows Event Log subsystem. If you wanted, you could write scripts to pull data programmatically. Microsoft, at its core a developer's company since it started by just offering the BASIC language as its first product, also has its own ways of getting data. If you were to use the .NET framework, writing in one of the languages that supports .NET, you could very quickly and easily write a program that could extract data from the Windows Event Logs. A very simple query is a couple of lines of code in a C# program, as an example. Microsoft provides all of the functionality to interact with the Event Log subsystem in a library module. What you do with the data you have retrieved is then up to you in the program you are writing.

Clearing Event Logs

It is possible to clear event logs; they can easily be deleted using the Event Viewer. In Figure 9-6, you can see the list of actions you can perform with the Event Viewer. In addition to filtering and creating a custom view, as discussed in the preceding section, you can also clear the log. This would wipe all of the entries from the log file, though you would get the option to save the log entries before clearing them.

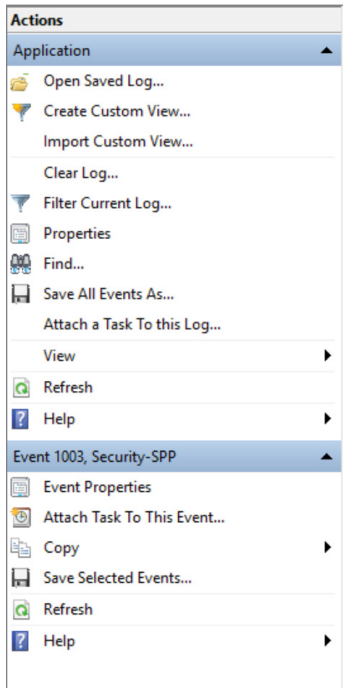


Figure 9-6: Event Viewer actions.

In fact, after you have cleared the Application log, there is nothing at all left. Not even an entry indicating that the log file has been cleared. Other logs don't allow you to get away scot-free, however. The Security log will leave behind an audit event indicating that the log had been cleared. The same is true of the System log. In fact, the System log will also generate an event saying that the Application log has been cleared, if the Application log were to be cleared. If you clear the System log, you will be left with a single event, similar to the Security log, saying that it had been cleared. You can see the single event and its details in Figure 9-7.

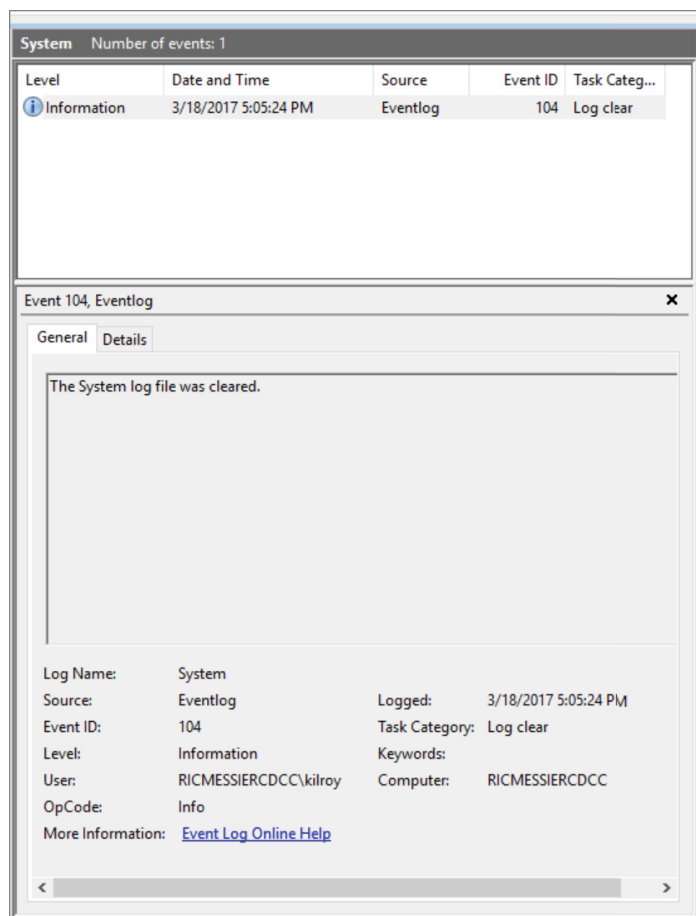


Figure 9-7: Cleared system log.

The Event Viewer is not the only way to clear the log, though. You can also use `wevtutil`. If you use `wevtutil cl Application`, you will clear the Application log. Just as if you had used the Event Viewer, you won't leave a trace in the Application log, but you will leave a trace in the other logs. In the System log, you will have a single entry. Listing 9-12 shows the use of `wevtutil` and then the query of the System log showing the single entry that says the log file was cleared.

Listing 9-12: System Log Clearance

```
C:\Windows\System32\winevt\Logs>wevtutil cl System

C:\Windows\System32\winevt\Logs>wevtutil qe System /f:text
Event[0]:
  Log Name: System
  Source: Microsoft-Windows-Eventlog
```

```

Date: 2017-03-18T17:12:05.142
Event ID: 104
Task: Log clear
Level: Information
Opcode: Info
Keyword: N/A
User: S-1-5-21-3194329012-3227895253-4200476558-1001
User Name: RICMESSIERCDCC\kilroy
Computer: RICMESSIERCDCC
Description:
The System log file was cleared.

```

All of this is to say that if an attacker uses tools to clear the event log, there will typically be an entry indicating the event log had been cleared. Even deleting the files is made challenging because Microsoft locks the files on a running system. Attempting to delete the files generates an error that they are in use. This is not to say that clearing log files or manipulating them in some way is impossible, but it does require the skills of someone more experienced and technical than an average user.

Firewall Logs

Firewalls come in a variety of flavors. There are network-based firewalls and host-based firewalls. This is very different from the case where intrusion detection systems are host-based or network-based, which serve different purposes. No matter where you place a firewall, whether on a host or in a network, the objective is to block traffic. A host-based firewall will block traffic or activity on the host. The firewall lives low in the network stack and evaluates network traffic as it comes into the system from the network interface. You may find a multi-featured endpoint security system that includes a firewall, but the entire software package is not the firewall. This is especially true for our purposes.

A network firewall would be placed in a network location such that it would become a choke point, making decisions about what traffic to allow through and what to block. At its core, no matter where it is positioned, the job of a firewall is to make decisions about what to allow and what not to allow at the packet or flow level.

Beyond network-based and host-based, there are other ways of thinking about firewalls. One is a *stateless firewall*. The stateless firewall is essentially an access control list. It has very simple rules to determine whether to allow traffic or block it. As an example, you can define a stateless firewall rule by saying any traffic coming into port 22 should be dropped. That's just an access control rule. You could make rules based on whether the source matched an acceptable network location or not. These are not very complex rules and making decisions based on them is trivial.

A stateful firewall adds another layer of complexity. In addition to the simple port, protocol, and address rules, you can also determine whether to allow traffic based on its state. This means that if communication has originated from inside the network, the return traffic should always be allowed because the communication has already been established. The firewall can pay attention to the state

of the communication based on flags in the TCP headers, but with UDP, the firewall would have to keep track of the state of communication using a state table. As soon as it sees a message going out, it flags information from that message in the state table in order to match the response traffic when it is seen.

A deep packet inspection firewall goes even deeper. Where stateless and stateful firewalls look at headers, a lot of malicious activity happens at the application layer. Looking at the data and not just the headers is time consuming. There is a chance that looking at the data inside the packet, which wouldn't be as structured and easy to investigate as the packet headers, could slow down delivery of the message. This is less of an issue than it used to be with much faster processors in firewalls today.

No matter what type of firewall, though, looking at firewall logs will be useful. The logs will help to correlate what you have been seeing and can fill in missing pieces if there are messages that have been dropped. This is traffic that you wouldn't have seen hit the application and get logged there, but you may have seen it come in over the network if you had capture sensors in place. Not all firewalls will automatically log dropped messages but they can often be directed to log drops. These logs will provide you with useful information. You can even have rules established to just log all traffic passing through the device.

NOTE Logging network traffic, even if you are just logging metadata, will increase processor overhead on the firewall and will definitely consume disk space. Depending on the location of the firewall and the network the firewall sits on, this may not be a very big deal. Other networks generate enough log data that it could require significant disk space to store the data. It would be quite a bit worse if you were trying to store all of the packet data in addition to the metadata from the headers.

To provide a sample of firewall logs, Listing 9-13 shows some logs generated by the firewall on a Linux system. These were generated by a CentOS 7 system with firewalld, but the older iptables firewall will generate logs that look a lot like this. Of course, other firewalls will generate log files that may look different from what you are looking at but it's easy to see the important elements from the logs here, and those elements will be included in the log files from most firewalls. Because the log messages were generated by the kernel on a Linux system, they were sent to syslog so you will recognize the format here.

Listing 9-13: Linux Firewall Logs

```
Mar 18 15:13:23 boingers kernel: IN=ens160 OUT=
MAC=00:0c:29:3f:0f:1e:f4:5c:89:b7:2c:89:08:00
SRC=192.168.86.65 DST=192.168.86.110 LEN=40 TOS=0x00
PREC=0x00 TTL=64 ID=129 PROTO=TCP SPT=65403 DPT=22
WINDOW=4096 RES=0x00 ACK URGP=0
Mar 18 18:27:41 boingers kernel: IN=ens160 OUT=
MAC=00:0c:29:3f:0f:1e:f4:5c:89:b7:2c:89:08:00
```

```

SRC=192.168.86.65 DST=192.168.86.110 LEN=40 TOS=0x00
PREC=0x00 TTL=64 ID=31938 PROTO=TCP SPT=49898 DPT=22
WINDOW=4096 RES=0x00 ACK URGP=0
Mar 18 20:26:25 boingers kernel: IN=ens160 OUT=
MAC=00:0c:29:3f:0f:1e:f4:5c:89:b7:2c:89:08:00
SRC=192.168.86.65 DST=192.168.86.110 LEN=64 TOS=0x00
PREC=0x00 TTL=64 ID=63522 DF PROTO=TCP SPT=57743 DPT=22
WINDOW=65535 RES=0x00 SYN URGP=0
Mar 18 20:27:52 boingers kernel: IN=ens160 OUT=
MAC=00:0c:29:3f:0f:1e:f4:5c:89:b7:2c:89:08:00
SRC=192.168.86.65 DST=192.168.86.110 LEN=64 TOS=0x00
PREC=0x00 TTL=64 ID=37753 DF PROTO=TCP SPT=57753 DPT=22
WINDOW=65535 RES=0x00 SYN URGP=0
Mar 18 20:37:25 boingers kernel: IN=ens160 OUT=
MAC=00:0c:29:3f:0f:1e:f4:5c:89:b7:2c:89:08:00
SRC=192.168.86.65 DST=192.168.86.110 LEN=64 TOS=0x00
PREC=0x00 TTL=64 ID=48005 DF PROTO=TCP SPT=57965 DPT=80
WINDOW=65535 RES=0x00 SYN URGP=0
Mar 18 20:37:25 boingers kernel: IN=ens160 OUT=
MAC=00:0c:29:3f:0f:1e:f4:5c:89:b7:2c:89:08:00
SRC=192.168.86.65 DST=192.168.86.110 LEN=64 TOS=0x00
PREC=0x00 TTL=64 ID=16288 DF PROTO=TCP SPT=57966 DPT=80
WINDOW=65535 RES=0x00 SYN URGP=0
Mar 18 20:37:25 boingers kernel: IN=ens160 OUT=
MAC=00:0c:29:3f:0f:1e:f4:5c:89:b7:2c:89:08:00
SRC=192.168.86.65 DST=192.168.86.110 LEN=64 TOS=0x00
PREC=0x00 TTL=64 ID=4327 DF PROTO=TCP SPT=57967 DPT=80
WINDOW=65535 RES=0x00 SYN URGP=0
Mar 18 20:37:27 boingers kernel: IN=ens160 OUT=
MAC=00:0c:29:3f:0f:1e:f4:5c:89:b7:2c:89:08:00
SRC=192.168.86.65 DST=192.168.86.110 LEN=64 TOS=0x00
PREC=0x00 TTL=64 ID=39602 DF PROTO=TCP SPT=57968 DPT=80
WINDOW=65535 RES=0x00 SYN URGP=0
Mar 18 20:37:27 boingers kernel: IN=ens160 OUT=
MAC=00:0c:29:3f:0f:1e:f4:5c:89:b7:2c:89:08:00
SRC=192.168.86.65 DST=192.168.86.110 LEN=64 TOS=0x00
PREC=0x00 TTL=64 ID=8476 DF PROTO=TCP SPT=57969 DPT=80
WINDOW=65535 RES=0x00 SYN URGP=0

```

First, as with any syslog entry, you will see the date and time followed by the short name of the system that generated the log. In this case, the system name is `boingers`. This is followed by the application name. As mentioned previously, the kernel generates these logs because the firewall lives inside the kernel, where the network stack is. Some of the remaining fields are going to be recognizable, while others may require some explanation.

- **IN/OUT**—These two values indicate the interfaces that are used. The log entries here only indicate an `IN` interface because there isn't a second interface. If the firewall were passing traffic through it, there would be an `OUT` interface indicating where the packet was going next.

- **MAC**—This is the Media Access Control address, which is the address used at layer 2 in network communications.
- **SRC/DST**—These fields are the source and destination IP addresses.
- **LEN**—This is the length of the packet in bytes.
- **TOS/PREC**—TOS is the Type of Service, while PREC is the precedence given to the packet. These two fields would be used if there was quality of service in use in the network. They would allow a network device to determine their priority.
- **TTL**—The TTL is the time to live. This is the number of hops the packet can travel before it is considered to have lost its way, at which point it will be dropped.
- **ID**—The ID field is the IP identification value. This value is used to reconstruct fragmented packets. The ID for the fragments should be the same across all the fragments.
- **DF**—This indicates whether the Don't Fragment bit has been set.
- **PROTO**—This is the protocol set in the IP header, indicating the transport layer protocol in use.
- **SPT/DPT**—This is the source and destination ports from the TCP headers.
- **WINDOW**—This is the number of bytes that can be transmitted without an acknowledgment being transmitted.
- **RES**—These are the reserved bits, which shouldn't be used.
- **SYN**—This indicates that the message has the SYN bit set.
- **URGP**—This is the urgent pointer. It would be used in case the URG flag were set. If the URG flag is set, this field would be used to indicate where the urgent data is located.

In essence, these log values just read out the values from the packets that were logged. There is no indication here of the disposition of the packets themselves, so these particular logs may be of less value. However, it is possible, within the Linux firewalls, to tag messages providing more of an indication of the disposition. Again, this is just the basic firewall that comes built into Linux. Other firewalls, especially commercial ones, would have more useful reporting features in some cases. These logs could also be sent to a system like a log management system or a security incident and event management system (SIEM). These types of systems may be able to correlate the data from these logs and provide a bigger picture that may be more useful.

Proxy Logs

A proxy is another type of firewall, since it is capable of making determinations as to whether traffic should be allowed to pass through. In most cases, the type of proxy you will run into is a web proxy. The web proxy takes in requests from the user, then re-originates those requests as though it were doing the requesting for itself. When the response comes back, the proxy has a mapping for who requested what so it knows who to pass the response message back to. The proxy is also capable of determining whether web traffic should be allowed through. If it's not, it would generate an HTTP error and send that back to the user.

Proxy servers are often used as a way of conserving bandwidth, since the proxy server will cache copies of web documents locally. This means the request won't need to go back out to the Internet every time a user inside requests the same page or set of pages. As an example, let's say everyone in your company decided to go visit Wiley's website today. The first one would get the copy and the proxy server would cache it. The next person's request for the Wiley web page would be much faster because it was being served out of the cache rather than being requested over the Internet. Additionally, proxy servers can be used to protect the enterprise, using rules to determine what websites are okay to visit and what websites are not. If a website has been determined to be unacceptable, for whatever reason, the request would get rejected.

In Listing 9-14 you can see a fragment from a proxy server log. This is from the open source proxy server, Squid. You will see that the log files here look much like web application logs. What is happening is the proxy server is determining whether there is a file cached or not. If it's not, there is a miss. This means the proxy server has to reach out to the Internet and request the document from the server where it is stored.

Listing 9-14: Squid Proxy Logs

```
1489887470.182    623 172.30.42.12 TCP_MISS/302 1054 GET
http://s1133198723.t.eloqua.com/visitor/v200/svrGP? -
HIER_DIRECT/142.0.160.13 text/html
1489887470.526    701 172.30.42.12 TCP_MISS/200 455 GET
http://s1342121626.t.eloqua.com/visitor/v200/svrGP.aspx? -
HIER_DIRECT/142.0.160.13 image/gif
1489887470.851    667 172.30.42.12 TCP_MISS/200 455 GET
http://s1133198723.t.eloqua.com/visitor/v200/svrGP.aspx? -
HIER_DIRECT/142.0.160.13 image/gif
1489887470.875    18 172.30.42.12 TCP_MISS/200 1769 GET
http://media.wiley.com/spa_assets/R16B094RC8/site/wiley2/pvo/
images/favicon.ico - HIER_DIRECT/23.216.88.157 image/x-icon
1489887470.920    63 172.30.42.12 TCP_MISS/200 691 GET
http://nsg.symantec.com/Web/Seal/Dynamic.aspx? -
HIER_DIRECT/184.85.193.14 text/javascript
1489887471.225    369 172.30.42.12 TCP_MISS/200 621 GET
http://zn3lwsfwmnkphulzr-wiley.siteintercept.qualtrics.com/
WRSiteInterceptEngine/? - HIER_DIRECT/23.206.197.230
application/javascript
1489887473.362    4199 172.30.42.12 TCP_TUNNEL/200 73998 CONNECT
script.hotjar.com:443 - HIER_DIRECT/23.111.9.32 -
1489887473.388    4218 172.30.42.12 TCP_TUNNEL/200 6278 CONNECT
vars.hotjar.com:443 - HIER_DIRECT/94.31.29.64 -
1489887477.631    8339 172.30.42.12 TCP_TUNNEL/200 3485 CONNECT
gtrk.s3.amazonaws.com:443 - HIER_DIRECT/52.216.18.128 -
1489887477.632    8339 172.30.42.12 TCP_TUNNEL/200 3485 CONNECT
```



```

gtrk.s3.amazonaws.com:443 - HIER_DIRECT/52.216.18.128 -
1489887481.198 11948 172.30.42.12 TCP_TUNNEL/200 470 CONNECT
logx.optimizely.com:443 - HIER_DIRECT/34.196.177.18 -
1489887481.198 11896 172.30.42.12 TCP_TUNNEL/200 655 CONNECT
www.google.com:443 - HIER_DIRECT/216.58.217.4 -

```

This sample indicates that there are requests to Wiley's website, as well as Google's. Looking at Web activity logs through the perspective of a proxy server can be interesting since you will see all of the requests that go into generating a page. Some of these requests are undoubtedly the result of the pages that were visited having relationships with analytics engines in order to better track user activity and engagement. Hotjar.com, for instance, allows the site owner using it to get a better understanding of the users who are visiting the site. It's important to note here that not all requests you will see in these logs are originating from the hand of the user. Just because a user requested one page doesn't mean that dozens of other requests came because the user manually went to those pages. Analytics and ad engines, directed by a page from a legitimate site, may generate traffic that can look strange or unusual.

Proxy servers like Squid's can be run through analyzer programs to generate statistics, just like Web logs can be. A number of programs exist that will take in Squid logs. The program Webalizer, for instance, will analyze Squid proxy logs as well as Web server logs, and a number of other programs can be used. Having an analyzer tool like Webalizer or one of the others to take your logs in and aggregate some of the data can be very beneficial and much easier than poring through page after page of logs like you see in Listing 9-14.

Web Application Firewall Logs

Where the proxy server is used to protect the user, the Web application firewall (WAF) is used to protect the server. This is another service that can be used to block traffic. Instead of blocking outbound traffic like a proxy server would, the Web application firewall would block inbound traffic. These types of servers are used inline with web application servers because the network firewalls in front of them are generally inadequate to protect against application attacks, which web applications can be highly susceptible to and there can be a lot of traffic focused at web servers trying to exploit web applications. The Web application firewall will inspect all of the traffic coming in at the application layer. It makes no network decisions. Instead, it looks at behaviors in the HTTP that is being sent into the server and determines whether an attack is underway.

Much like network firewalls, the WAF runs based on rules. When someone knows what an attack looks like, they would write a rule to do something based on that. They create a signature that the attack can be mapped to. The rule also indicates what the disposition of the inbound request should be if the rule is matched. This particular WAF is modsecurity, which is a module that runs with the Apache web server. It will also run with IIS and Nginx. While you can design your application architecture in such a way that modsecurity can be on a standalone box on the edge of the architecture, you can also just run it inside the Apache web server that is perhaps serving up your content

to the user. In either case, when a rule is triggered, modsecurity generates a complex log that can be seen in Listing 9-15.

Listing 9-15: Web Application Firewall Logs

```
--d480d20a-A--
[19/Feb/2017:22:30:24 --0500] WKpivH8AAQEAAUwA904AAAAH
192.168.86.65 57575 192.168.86.83 80
--d480d20a-B--
OPTIONS /security.php HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:42.0)
Gecko/20100101 Firefox/42.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://192.168.86.83/security.php
Cookie: PHPSESSID=e945t3jli2psaes0klfd7upnl2; security=impossible
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 77
Host: 192.168.86.83

--d480d20a-F--
HTTP/1.1 500 Internal Server Error
Content-Length: 532
Connection: close
Content-Type: text/html; charset=iso-8859-1

--d480d20a-E--
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error or
misconfiguration and was unable to complete
your request.</p>
<p>Please contact the server administrator at
webmaster@localhost to inform them of the time this error occurred,
and the actions you performed just before this error.</p>
<p>More information about this error may be available
in the server error log.</p>
</body></html>

--d480d20a-H--
Message: Error reading request body: Partial results are valid but
processing is incomplete
Apache-Handler: application/x-httpd-php
Stopwatch: 1487561404207982 20005648 (- - -)
```

```
Stopwatch2: 1487561404207982 20005648; combined=770, p1=762, p2=0, p3=1,  
p4=0, p5=6, sr=203, sw=1, l=0, gc=0  
Response-Body-Transformed: Dechunked  
Producer: ModSecurity for Apache/2.9.0 (http://www.modsecurity.org/);  
OWASP_CRS/2.2.9.  
Server: Apache/2.4.18  
Engine-Mode: "DETECTION_ONLY"  
  
--d480d20a-Z--
```

Modsecurity uses a multi-part log format. Each part has a different purpose, though not all parts will always exist within the log output. The sections we can see in the preceding code are A, B, F, E, H and Z. The line `--d480d20a-A--` indicates at the end that this is section A. You can find the other sections in the logs because they will have the same header, just with a different letter at the end, indicating which log output section follows. The A section is the audit log header including information about the audit log entry, such as a time stamp. Section B is the request headers. This is the HTTP message that was sent, without any data that may have been transmitted outside of the headers. F includes the response headers. In this case, it appears that the response was a 500 internal server error that was sent back. Section E is related to F and includes the intended server response. This includes more than section F. For example, in addition to the headers, you can see the actual error message that was sent back.

Section H is the audit log trailer that includes more information related to what happened. As an example, you can see two stopwatch lines. The two stopwatch lines can be used to help determine the performance of the web server as well as that of Modsecurity. Section H also tells us about the server and the version of Modsecurity in use. It also says that we are using the core rules set (CRS) from OWASP. Finally, we can see that the Engine-Mode is set to detection only. This means that even if Modsecurity determines that something bad is going on, all it will do is generate a log entry. It will not block the message. The final section is Z and that just indicates that the entire log message is done.

Common Log Format

One of the early web servers came out of the National Center for Supercomputing Applications (NCSA) at the University of Illinois, Champaign-Urbana. The NCSA HTTP daemon (httpd) was released in 1993 and a version of it became the source code for the Apache web server, which has been the most used web server since the mid-1990s. The NCSA server, being one of the first, defined the log format that would continue to be used on web servers. It is a very basic log format where information is stored on a single line, and because it has become a standardized way of logging information from servers, it is called the common log format (CLF). Each entry in a log where CLF is used contains the following information:

- Host—This is the remote host that originated the request.
- Identification—RFC1413 defines user identity, which can be provided by the identity daemon on the user's system.
- Authenticated user—After a user has logged in, they become an authenticated user and this field contains the username.
- Date—This is the date of the request.
- Request—This is the request that has been made, including the method and universal resource location (URL).
- Status—The status that the server provided back from the request.
- Bytes—The number of bytes that have been transmitted back to the client.

In cases where there is no data, a dash (-) is used. This may be common in cases where the identity is not provided by the client. Many systems don't include the identity daemon, as they would have commonly at the time the NCSA server was developed and primarily used. Also, not all requests need to be authenticated.

While the common log format is still in use, web servers may be more likely to use the combined log format. It contains the same information that is contained in the common log format but it carries additional information as well. The combined log format adds two fields. The two fields can be configured and can be any request header. As an example, you could include the referrer header to log what site the request originated from, if there is one. You could also log the user agent, which is a character string indicating the browser and operating system that was used to generate the request. You can see samples of the combined log format from a fragment of a log from an Apache web server in Listing 9-16.

Listing 9-16: Apache Log Entries

```
192.168.86.65 - - [19/Feb/2017:22:30:26 -0500] "GET
/vulnerabilities/sqli/?Submit=Submit%2526zap%253Dzapoxy HTTP/1.1"
302 333 "http://192.168.86.83/security.php" "Mozilla/5.0
(Macintosh; Intel Mac OS X 10.12; rv:42.0)
Gecko/20100101 Firefox/42.0"
192.168.86.65 - - [19/Feb/2017:22:30:26 -0500] "GET /login.php HTTP/1.1"
200 1841 "http://192.168.86.83/vulnerabilities/sqli/" "Mozilla/5.0
(Macintosh; Intel Mac OS X 10.12; rv:42.0) Gecko/20100101 Firefox/42.0"
192.168.86.65 - - [19/Feb/2017:22:30:26 -0500] "GET /login.php HTTP/1.1"
200 1841 "http://192.168.86.83/security.php" "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10.12; rv:42.0) Gecko/20100101 Firefox/42.0"
192.168.86.65 - - [19/Feb/2017:22:30:26 -0500] "GET
/vulnerabilities/sqli_blind/?Submit=Submit%2526zap%253Dzapoxy HTTP/1.1"
302 333 "http://192.168.86.83/vulnerabilities/sqli/?id=%27+or+%27a%27+
%3D+%27a&Submit=Submit" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12;
rv:42.0) Gecko/20100101 Firefox/42.0"
```

```
192.168.86.65 - - [19/Feb/2017:22:30:26 -0500] "GET /login.php HTTP/1.1"
200 1841 "http://192.168.86.83/vulnerabilities/sqli/?id=%27or+%27a%27+
%3D+%27a&Submit=Submit" "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10.12; rv:42.0) Gecko/20100101 Firefox/42.0"
```

The start of each line in the log file looks exactly like what the common log format would look like. You can see the host that originated the request. The host is an IP address on my local network. The hosts would commonly be stored as IP addresses, to save on doing reverse lookups for every request that comes in. This is followed by two dashes because `identd` is not running on the client that generated the request. Additionally, the request did not require a login so there is no authenticated username. This is followed by the request date and time and then the request. The request is in quotes and includes both the method (GET) as well as the URL that is being requested. In the last log entry, the URL is `/login.php`. This is followed by the HTTP version, `HTTP/1.1`. The response code is 200, which means success and the response from the server was 1841 bytes.

After the number of bytes is the referrer. This is the URL that led to the request on this log line. The last entry on this line is the user agent. Based on the user agent string, the request came from a Mac OS system running Firefox. You can also tell from the user agent string what version of the operating system is running on the client.

Web servers also have error logs and while they aren't as standardized, they can be just as useful. The sample in Listing 9-17 is from the same Apache web server that the access log in Listing 9-16 came from.

Listing 9-17: Apache Error Log

```
[Sun Feb 19 22:29:57.170543 2017] [:error] [pid 84966] [client
192.168.86.65:57572] script '/var/www/html/vulnerabilities/
view_source_allbackup.php' not found or unable to stat
[Sun Feb 19 22:29:57.172893 2017] [:error] [pid 84966] [client
192.168.86.65:57572] script '/var/www/html/vulnerabilities/
Copy of view_source_all.php' not found or unable to stat
[Sun Feb 19 22:29:57.175141 2017] [:error] [pid 84966] [client
192.168.86.65:57572] script '/var/www/html/vulnerabilities/
Copy (2) of view_source_all.php' not found or unable to stat
[Sun Feb 19 22:29:57.177458 2017] [:error] [pid 84966] [client
192.168.86.65:57572] script '/var/www/html/vulnerabilities/
Copy (3) of view_source_all.php' not found or unable to stat
[Sun Feb 19 22:30:02.345315 2017] [:error] [pid 84969] [client
192.168.86.65:57574] PHP Fatal error: Uncaught Error: Call to
undefined function dvwaMessagePush() in /var/www/html/dvwa/
includes/DBMS/MySQL.php:10\nStack trace:\n#0 {main}\n thrown in
/var/www/html/dvwa/includes/DBMS/MySQL.php on line 10, referer:
http://192.168.86.83/dvwa/includes/DBMS/
[Sun Feb 19 22:30:02.756811 2017] [:error] [pid 85016] [client
192.168.86.65:57571] PHP Warning: define() expects at least
```

```

2 parameters, 1 given in /var/www/html/dvwa/includes/
dvwaPhpIds.inc.php on line 4
[Sun Feb 19 22:30:23.801370 2017] [:error] [pid 85016] [client
192.168.86.65] ModSecurity: Error reading request body: Partial
results are valid but processing is incomplete [hostname
"192.168.86.83"] [uri "/login.php"]
[unique_id "WKpiu38AAQEAAUwYU6IAAAAF"]

```

There are entries that are not shown here that relate to the service itself. You will get those entries in addition to the messages shown. The messages can indicate that someone is looking on the server for pages that don't exist. This may be part of a site crawl using a dictionary attack or it could just be a mistake. Having log entries like this can help to correlate other evidence from a network investigation.

Summary

Log files can be an important place to obtain supporting information for a network forensic investigation. Unix-like systems use syslog as the common logging platform. Syslog is a text-only logging format. On a Windows system, the system and applications use the Windows Event Log. Windows Event Logs are stored as XML in a binary file. They can be queried using the Windows Event Viewer. They can also be queried using Xpath with the wevtutil command-line utility.

While the network is the primary source of information and has the most accurate data, there are secondary sources that can be used. One advantage of using secondary sources is that you can't always guarantee you will have network captures. Firewall logs will provide information about what is happening at the network level without necessarily relying on the network itself. Both host and network firewalls can provide data that can correlate what happens over the network. While the firewalls may not necessarily provide you the data automatically, they can be configured to provide more logging that can support what can be acquired from the network.

Firewalls come in many styles. A traditional firewall could be stateless or stateful. It may also be deep packet inspection. You can get network data out of firewall logs. On top of the network-level firewalls, you can get application layer gateways and the logs that are associated with them. A proxy, which is an application layer gateway for web traffic, can provide a lot of data about what users are doing on an enterprise network. Since proxies are capable of blocking traffic from enterprise users, you can acquire information about bad sites that have been used as well as sites that have been blocked.

Web application firewalls provide application layer data to go along with any network-based information. Where a traditional firewall can get TCP/UDP/IP header information, a Web application firewall can provide details in the HTTP headers and data related to web attacks.

10

Correlating Attacks

In this chapter, you will learn about:

- How to synchronize time across multiple systems
- How to aggregate logs
- The importance of timelines and how to create them
- Security information and event management systems

As he looks at the systems he has gathered from this one company, it appears that they are scattered around the United States with one or two in Ireland. He recognizes how important the events that he is performing are in creating a trail demonstrating his jumping from one system to another. Unfortunately, when logs are sent off the system they originated on, it's hard for him to protect himself. He can't just wipe the logs because they are sent to a waiting system in another part of the network the moment they are created. Wiping the local log has no benefit there.

He checks time zones and clock settings on each of the systems he has entered. He realizes that not only do they span different time zones, but in many cases, their times don't match up. Sometimes even within the same building. Clocks are sometimes off by several minutes or more. Sometimes, clock settings are entirely wrong. This is good news for him, since anything these systems are generating can be misleading. This is in addition to the obfuscation he is creating. The more he can do to create a quicksand to mire down those who may eventually be tracking him, the better off he will be. More days under his control means more money in his pocket. This is a good thing.

When you are working within an enterprise, you will be working on multiple systems. You will have multiple artifacts, including logs, that need to be presented in a coherent way. This can be especially true when it comes to an enterprise that is diverse geographically. Fortunately, there are ways to synchronize systems across multiple time zones and there are ways to do this so no matter where your systems are around the globe, they can be synchronized to standard, reliable time sources. This will be important when it comes to pulling a timeline together.

The timeline is the end goal of everything we are doing here. A timeline represents the entire history of an incident in a concise manner that is easy to follow chronologically. Putting events

into a timeline helps to determine correlation, as well as what may be related and what may not be related to the entire incident. It also helps to isolate outliers within the entire timeline of events; the outliers may prove to be useful or they could be ruled out. Looking at a long list of timestamps can make it hard to chase down what happened. This is why pulling events out of all of the sources of information you have and placing them in a timeline can be useful. You can see, visually, how it all unfolded and what it looked like.

A number of tools are available to assist with the construction of timelines. Because a timeline is essentially a visual tool, the programs that are used to create timelines are typically graphical programs because it can be much easier to just drag events around on the timeline if you need to rearrange or just make room for new events. There is a significant program that is not graphical that is used to help generate timelines, though. The program `log2timeline` and other, associated programs, is important because it can be used to help extract important time information from logs and other data sources. This can help to extract what is necessary from sources that may be difficult to get time information from in a consumable manner.

Packet captures don't include their own per-frame times. Everything is based on offsets from the beginning of the packet capture. As a result, packet captures can also be used as a source of timeline information, and this is important because of the value of using packet captures in a network investigation. Packet captures often need to be correlated with logs, and logs can come from a large number of systems. It can be problematic to extract logs from every system that is related. Fortunately, there are ways to deal with that by making sure that the logs are pulled together at the moment they are created.

Ultimately, the most important thing, though, is to have a solid foundation to build your evidence on. That solid foundation relies on synchronizing time across your systems, no matter what time zone they are in.

Time Synchronization

Time synchronization means that every system in the network, or at least every system you care about, is using the same source for its timing information. Local clocks on systems can have a tendency to drift some, which means they should be regularly checking with a more reliable time source to ensure all systems are accurate. Even a couple of minutes of drift in the clock can make a big difference in correlating events. If you have a network intercept that says one thing but a log that says something else, and it's not clear there is a clock problem, you can end up with events completely out of sequence. That may end up having an impact on your attempt to determine a root cause or attaching actions to a particular actor.

Time Zones

Time zones are geographic distinctions made around the globe because of the way the earth rotates during the day. Since the way we measure time is based around celestial objects like the sun, we rely

on our own orientation to that celestial body. As an example, when the sun is at its apex in the sky, we consider that noon. The problem is that when it is at its apex for me, it is a good way on the wane on the east coast of the United States. In Europe, the sun may have gone down altogether. In order to correct for this, we use time zones. Everyone in the same sliver of the globe, from top to bottom, is in the same time zone. While it's not perfect, noon for me in Colorado is going to have the sun at roughly the same position in the sky for someone in Salt Lake City, also in the Mountain Time zone.

The point of origin for all of the time zones is sometimes called Greenwich Mean Time (GMT). The same time measurement is also called Coordinated Universal Time (UTC). Every other time zone around the world is an offset from that measurement. As an example, I am seven hours behind GMT and UTC. Eastern Europe is ahead. Riga in Latvia, for instance, is three hours ahead, while Bern in Switzerland is two hours ahead.

NOTE While we call it Coordinated Universal Time or Universal Time Coordinated, the abbreviation is UTC. The reason is to have a single way of referring to the time standard. English speakers proposed CUT as the abbreviation, while the French wanted TUC (*temps universel coordone*). UTC was the compromise. UTC is not a time zone. GMT is a time zone. UTC is a time standard and the point of origin for all time zones. In practice, UTC and GMT are the same thing when it comes to the time.

Another thing to consider is daylight saving time (DST). Many time zones around the world observe daylight saving, which can vary a local time by an hour from the base time. Neither GMT nor UTC observe or make adjustments for daylight saving. In the United Kingdom, the time zone designation becomes British Summer Time (BST). In the United States, it is a letter change. Where MST is Mountain Standard Time, the time measurement based on the sun being more or less at its apex at noon, in the summer when the time changes, it becomes Mountain Daylight Time (MDT). Since computers are capable of making the adjustment automatically when the time changes, the time zone may be referred to as MST7MDT, indicating that in standard time, the clock is -7 from UTC. Part of the year, though, is different from that. At the moment, as I write this, I am only six hours different from UTC.

When systems need to all be on a particular time standard in order to make sure that what happens can be resolved easily later on, as in a timeline, they may simply set their local clocks to UTC. The local time can always be calculated easily, but if the system clock is set to UTC, every system, no matter where it is, has the same time and no resolution needs to happen to ensure every event captured has the same base time.

Network Time Protocol

The network time protocol (NTP) was created to share time across the network. When systems use NTP, they are typically clients to a time server. Periodically, the NTP client will check with the NTP server to synchronize the time. The idea is that the NTP server has a more reliable clock than the

client. Regular synchronizations will correct for any drifting of the local clock. The clock on the client will get reset based on what it receives from the NTP server.

Because speed in these cases is important, NTP transmits and receives over UDP. The idea is that you want to receive the messages from the NTP server with the shortest delay absolutely possible. Additionally, in the case of a time synchronization message, it is less important that it get there guaranteed. Instead, if it doesn't get to the server or client, the application will retransmit as necessary. This means that there is no verifying that you are talking to the correct server. Speed is the most important factor.

NTP uses a hierarchy to determine the reliability of the time. A stratum 0 clock would be either an atomic clock or a global positioning system (GPS) clock where the time is accurate to milliseconds. Beneath the stratum 0 clocks would be stratum 1 clocks. A stratum 1 clock is a reference clock. These synchronize to the stratum 0 clocks and their time would be accurate to microseconds rather than milliseconds. The US Naval Observatory, for instance, operates several stratum 1 servers that synchronize with their own atomic clock, though they may also synchronize with GPS. Following down the chain, a stratum 2 server would synchronize its clock with a stratum 1 server, while a stratum 3 server would synchronize its clock with a stratum 2 server.

NOTE While NTP is a hierarchy, lower stratum clocks will also sometimes peer with each other to have additional validity, creating a more robust time on each of the clocks.

Each system within a network would be configured to synchronize with a local time source. This would ensure that all systems within a network had the same time source, maintaining local time to within fractions of seconds. Operating systems like Windows and Mac OS X are configured by default to synchronize with a time server maintained by Microsoft and Apple, respectively. In the case of Linux, systems would commonly use one of the pool servers maintained by ntp.org. The hostname pool.ntp.org would resolve to a system at the time it was checked since pool.ntp.org doesn't resolve to a single IP address. Instead, the resolution would rotate through multiple servers in the pool. Each time a request was made, the client making the request would get a different address than the one provided to the last client.

Desktops and servers are not the only systems that need to have their times synchronized. All network devices and other appliances would need to have their times synchronized as well. Routers, switches, and firewalls, along with other devices, are going to generate information that will be important in a network investigation. As a result, all devices you expect to be working with should be synchronized to the same source since that will make life much easier when it comes to trying to pull everything together.

Packet Capture Times

There is nothing in Ethernet frames or IP packets that provides any sort of timestamp that could be used to reliably mark packets or frames. You can look through the headers if you like, but there is nothing there. So, if we are acquiring a packet capture, how do we know anything about the time that any frame came in? The capturing program knows when it began running, and it knows, based on that, when captured frames arrived. A program like tcpdump would know exactly what time a frame arrived. In the output of tcpdump, as shown in Listing 10-1, you can see the time each frame arrived down to milliseconds.

Listing 10-1: tcpdump Output

```
17:40:01.212754 IP milobloom.lan.afpovertcp > oliver.lan.56504:
Flags [.), ack 8028, win 4094, options [nop,nop,TS val 324412441
ecr 195692146], length 0
17:40:01.212757 IP milobloom.lan.afpovertcp > oliver.lan.56504:
Flags [P.), seq 69244:69359, ack 8028, win 4096, options
[nop,nop,TS val 324412441 ecr 195692146], length 115
17:40:01.212794 IP oliver.lan.56504 > milobloom.lan.afpovertcp:
Flags [.), ack 69359, win 4092, options [nop,nop,
TS val 195692152 ecr 324412441], length 0
17:40:01.230318 IP testwifi.here.domain > oliver.lan.54130:
44910 1/0/0 PTR jc-in-f125.1e100.net. (80)
17:40:01.233093 IP jc-in-f125.1e100.net.5223 > oliver.lan.56527:
Flags [.), ack 31, win 860, options [nop,nop,
TS val 562035498 ecr 195692137], length 0
17:40:01.404911 IP oliver.lan.56507 > edge-star-shv-01-ord1.
facebook.com.https: Flags [.), ack 4217269812, win 4096, length 0
17:40:01.405822 IP oliver.lan.63399 > testwifi.here.domain: 12655+ PTR?
1.74.13.31.in-addr.arpa. (41)
17:40:01.423314 IP testwifi.here.domain > oliver.lan.63399:
12655 1/0/0 PTR edge-star-shv-01-ord1.facebook.com. (89)
17:40:01.441983 IP edge-star-shv-01-ord1.facebook.com.https >
oliver.lan.56507: Flags [.), ack 1, win 2043, options [nop,nop,TS val
919358756 ecr 195647271], length 0
17:40:01.774558 IP oliver.lan.56749 > google-home.lan.8009:
Flags [P.), seq 115:230, ack 116, win 4096, options [nop,nop,
TS val 195692712 ecr 5440334], length 115
17:40:01.782858 IP google-home.lan.8009 > oliver.lan.56749:
Flags [P.), seq 116:231, ack 230, win 344, options [nop,nop,
TS val 5440835 ecr 195692712], length 115
```

This output shows the actual time each frame came in because tcpdump knew what time it was. Wireshark, by contrast, represents time as an offset from the time the packet capture was started. This is, though, configurable. By default, you would see the first frame show 0.000000. The next frame would show the number of seconds since the first frame, down to milliseconds. Every successive frame would also show as being an offset from the start of the file in seconds. This is a view setting in Wireshark, though. You can change the view setting to show you the number of seconds since the start of Epoch time (1970-01-01 00:00:00). You can see Wireshark showing the number of seconds offset from Epoch time in Figure 10-1.

No.	Time	Source	Destination
1	1491867808.277480	fe80::1cab:a350:506:7c35	ff02::fb
2	1491867808.277880	milobloom.lan	224.0.0.251
3	1491867808.485626	edge-star-shv-01-ord1.fac...	oliver.lan
4	1491867808.485736	oliver.lan	edge-star-shv-01-ord1.facebook...
5	1491867808.667795	oliver.lan	testwifi.here
6	1491867808.667956	oliver.lan	testwifi.here
7	1491867808.668028	oliver.lan	testwifi.here
8	1491867808.668071	oliver.lan	testwifi.here
9	1491867808.668225	oliver.lan	testwifi.here
10	1491867808.668238	oliver.lan	testwifi.here
11	1491867808.671399	testwifi.here	oliver.lan
12	1491867808.671816	testwifi.here	oliver.lan
13	1491867808.672551	testwifi.here	oliver.lan
14	1491867808.673116	testwifi.here	oliver.lan
15	1491867808.673684	testwifi.here	oliver.lan
16	1491867808.686973	billthecat.lan	192.168.86.255
17	1491867808.687086	billthecat.lan	192.168.86.255
18	1491867808.687482	billthecat.lan	239.255.255.250

Figure 10-1: Wireshark showing Epoch time in seconds.

In reality, the file format, pcap or pcapng, stores timestamps for all frames in Epoch time. In addition to the timestamp on the file, stored in the global header, each frame will have its own header to store metadata. The frame header stores the timestamp on the frame in Epoch time, which is the number of seconds since January 1, 1970. Additionally, all times stored in a pcap file are UTC. There is no time zone information stored in the frame metadata, though the global header for the file will include the offset from UTC that was configured on the system at the time the packets were captured. You can see in Figure 10-2 that the time zone from the pcap header is set to 0. The line that says “int32 thiszone” has a value of 0, which is UTC. You’ll also see under Frame[0] that the time in seconds (Epoch time) shows the date and time that the frame came in on. The line below that shows the time in microseconds. In order to get the exact time, you put the two lines together, which is shown in the line that says Frame[0].

Figure 10-2 is from a hex editor using a pcap template to be able to extract the specific data segments from the file. Without this, it would be difficult to parse through the raw pcap file. You can, of course, just let Wireshark show you the time the way you want it by changing your view preferences under View ⇨ Time Display Format.

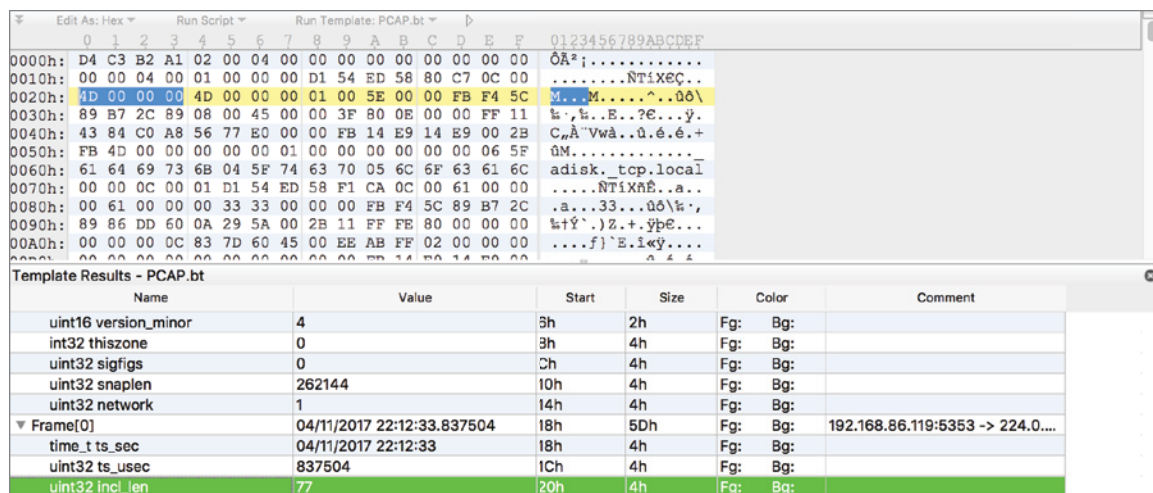


Figure 10-2: Hex editor showing times from PCAP.

Log Aggregation and Management

It is hard to overstate the importance of logging. From a forensic investigation standpoint, of course, the paper trail that logs provide is very helpful. Additionally, though, system administrators, network engineers, security engineers, and various other people who have to touch systems can get a lot of benefit from logging. As a result, turning on system logging is a good thing. The problem is where to put the logs once you have them. As mentioned previously, the problem with storing logs on the endpoints (servers, desktops, and so on) is that if they are compromised, the attacker can get access to those logs to wipe or alter.

The best approach is to offload logs to a central log host. This is not perfect. Just because you create a central log host doesn't guarantee that attackers won't be able to get to it. After all, if it is reachable enough from the endpoint to be able to get logs to, then the attacker can also reach it, even if it's only through the port listening for the log messages. A number of readily available solutions will allow you to create a centralized log host from open source to commercial offerings. Your needs should help you to determine which one may be the right way to go for you.

Although logging and different logging mechanisms were covered in Chapter 9, we will talk about the same kinds of logging from the perspective of collecting and correlating in the next sections.

Windows Event Forwarding

Windows uses the Event Log to store log information. If you want to manage the logs, view them, or search for information in them, you would use the Event Viewer. The Event Viewer also allows you to

set up log forwarding by creating subscriptions. If you look in the left-hand pane of the Event Viewer, as shown in Figure 10-3, you can see both Forwarded Events and Subscriptions. In the Forwarded Events log, you will see, probably not surprisingly, events that have been received (forwarded) from another system. This will allow you to separate the system logs from the system where the log collector is running from all of the other logs that may be sent to it from other systems.

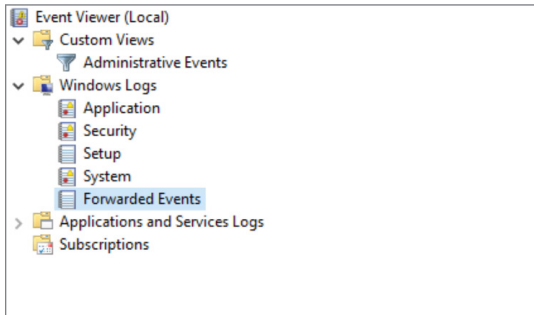


Figure 10-3: Windows Event Viewer.

In order to set up forwarding, you need to set up a system as a collector. The collector can be configured to either reach out to other systems or it can wait for other systems to contact it to send logs along. One way of getting logs is to use the collector to create a subscription. The subscription allows the collector system to subscribe to logs from another system. When you want to start a subscription, you go to the Subscriptions folder that you can see in Figure 10-3. From there, you click Create Subscription in the right-hand side of the window. This will bring up the dialog box that you see in Figure 10-4.

The dialog allows you to specify where you want the received logs to go. By default, they would go into the Forwarded Events log, though you can change the log you would like them to go into. You can select Collector Initiated or Source Initiated. If you wanted to have the collector poll for logs, you would select Collector Initiated. If you would rather the source computer send them along, you would select Source Initiated. Once you have determined how you want the subscription to work, you can create a filter to determine exactly what logs you want forwarded. If you enable a lot of logging and auditing, Windows can generate a lot of log entries, and perhaps you don't want all of them going to the collector. If this were the case, you would create a filter to specify exactly what you wanted forwarded on. It's important to note that the filter will let you indicate what you want forwarded, which means there are messages you are leaving on the endpoint and not forwarding them off somewhere.

Syslog

Another free offering that just comes with the operating system is syslog. This is available, typically, for Linux and Unix-like systems. Configuring a syslog server to listen for syslog messages is easy. Any implementation of syslog should have the ability to listen for syslog messages from other systems.

As an example, the following configuration fragment shows how to configure a syslog-ng server to listen for syslog messages over TCP. By default, syslog uses UDP, but TCP guarantees that the messages get to the server or will return an error indicating why so it's preferable to use TCP rather than UDP. This is especially true on critical systems. You'll also notice that in the configuration shown in Listing 10-2, I have specified which IP address to pay attention to, which effectively determines the interface I'm listening for messages on. While interfaces can have multiple IP addresses, any IP address can only be configured on a single interface on a system, so by indicating the IP address, we know which interface the message will be arriving on.

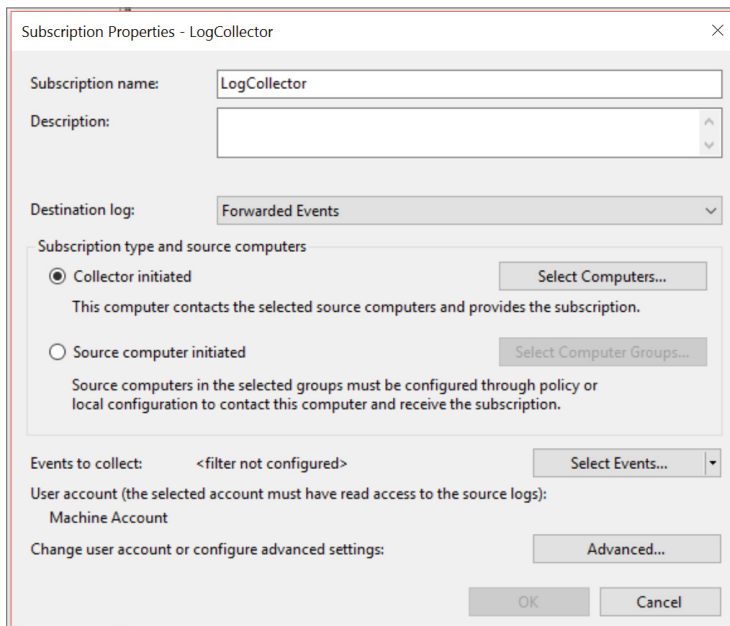


Figure 10-4: Creating a subscription in Windows Event Viewer.

Listing 10-2: syslog-ng Server Configuration

```
source network { syslog(ip("172.16.144.176") transport("tcp")); };
destination d_local { file("/var/log/messages_${HOST}"); };
log { source(src); source(network); destination(d_local); };
```

On the client side, you would essentially reverse the two lines at the top. Your destination becomes a network destination, configured just as you did in the source. You might label your destination `d_network` to make it clear it's a network destination. Your source would then be your local logging. You can see what that may look like in Listing 10-3.

Listing 10-3: syslog-ng Client Configuration

```
source src {
    system();
    internal();
};
destination d_network { syslog(ip("172.16.144.176") transport("tcp")); }
log { source(src); destination(d_network); };
```

Once you have these two configurations in place, you will be forwarding your syslog messages from one system to a central log host, also running syslog. You are not restricted to just Unix-like hosts when you are taking in syslog messages, though. A number of software packages are available that will install a syslog agent onto a Windows system. This will allow you to take log messages from a Windows system and send them off to a syslog server. As an example, rsyslog is a common implementation of syslog and rsyslog provides an agent that will monitor the Windows Event Log to forward events off to a syslog server.

Log Management Offerings

Windows Event Viewer and syslog are both essentially free offerings that you can take advantage of with your existing operating systems. You may, though, want something that gives you some additional power. There are a number of products around, both commercial and community-supported. With a log management solution, you are expanding on the capability of just taking logs in, though certainly Windows Event Viewer Subscriptions and even some implementations of syslog provide you with some of the advanced features of the log management solutions we're going to be talking about here.

One solution you can make use of is nxlog. Currently, IBM offers nxlog as a commercial offering, though there is a community edition as well. One advantage to nxlog is that it can be configured to accept logs in a variety of formats, generating a single, coherent log file that you can use. An example of an nxlog configuration is shown in Listing 10-4. What you can see in the listing is that nxlog has been configured to be a listener for syslog messages.

Listing 10-4: nxlog.conf Sample

```
<Extension _syslog>
    Module      xm_syslog
</Extension>

<Input in1>
    Module      im_udp
    Port        514
    Exec        parse_syslog_bsd();
</Input>
```



```

<Input in2>
  Module   im_tcp
  Port     514
</Input>

<Output fileout1>
  Module   om_file
  File     "/var/log/nxlog/logmsg.txt"
  Exec     if $Message =~ /error/ $SeverityValue =
syslog_severity_value("error");
  Exec     to_syslog_bsd();
</Output>

<Output fileout2>
  Module   om_file
  File     "/var/log/nxlog/logmsg2.txt"
</Output>

#####
# Routes                                     #
#####
<Route 1>
  Path     in1 => fileout1
</Route>

<Route tcproute>
  Path     in2 => fileout2
</Route>

```

This example has nxlog listening on both UDP and TCP, as indicated by the Input sections. The two Output sections write messages coming in to files. One advantage to nxlog is its ability to execute functions that can allow you to manipulate and massage data. You'll see in the first Output section that it not only sets the severity value based on whether it sees error in the message, but it also outputs to a syslog format rather than the nxlog format. A number of variables are defined by nxlog that allow you to make determinations based on what you see coming in. Finally, we have to join the input and the output. That is done in the Route sections. In each route, you have a path that indicates inputs and outputs. You can have multiple inputs going to a single output. If you need multiple outputs, you need multiple route sections. The two routes here say that UDP messages should be written out in syslog format, using Output1 since the assumption is that they are coming in using syslog format. Output2, on the other hand, just dumps the messages coming in to /var/log/nxlog/logmsg2.txt.

There are a number of other offerings when it comes to log management. Searching has become a very popular and important option when it comes to log management. One option, formerly called ELK, includes the search feature as the first part of its name. ELK is an acronym for Elasticsearch, Logstash, and Kibana. The three components together offer log management, search capabilities, and a web interface for management and queries. The trio of software is currently named Elastic Stack.

Another product that is popular is Splunk. Splunk has a commercial offering but there is also a lite version that you can use for small installations that you can download and install for free. If you have larger data needs, there are other offerings of Splunk. Like Elastic Stack, Splunk offers a web interface for management. Figure 10-5 shows the web interface where you can add a TCP or UDP listener to take in syslog messages or messages from other sources. Similar to nxlog, Splunk has the capability to parse different sources of data and will auto-detect some data sources. Splunk will also take in local sources. If you point Splunk at a directory where your logs are, `/var/log` for instance, Splunk will add the data source and start consuming and indexing the logs.

The screenshot shows the 'Add Data' web interface in Splunk. At the top, there's a progress bar with four steps: 'Select Source' (active), 'Input Settings', 'Review', and 'Done'. Below the progress bar, on the left, is a list of data sources: 'Files & Directories', 'HTTP Event Collector', 'TCP / UDP' (selected), and 'Scripts'. The main area on the right is titled 'Configure this instance to listen on any TCP or UDP port to capture data sent over the network (such as syslog)'. It features two tabs, 'TCP' and 'UDP', with 'TCP' selected. Below the tabs, there's a 'Port' field with the value '514' and an example 'Example: 514'. There's also a 'Source name override' field with the value 'optional' and a 'host:port' label. At the bottom, there's an 'Only accept connection from' field with the value 'optional' and an example 'example: 10.1.2.3, fbedhost.splunk.com, *.splunk.com'.

Figure 10-5: Creating listener in Splunk.

Once you have the logs in, you can use the Splunk interface to start reading all of the logs. Splunk also offers a number of ways to query the logs and drill into the data. On the left-hand side of Figure 10-6, there is a list of fields that you can search by. Clicking any of those will display data related to the link you clicked. Additionally, even within the log message, there are clickable links to look at the data in a different way. For instance, each entry has a host field that is clickable. If I were to click that field, Splunk would show me logs from that host. The same is true with the source and sourcetype fields. I can present information only from `/var/log/auth.log` in my output by clicking that source. Likewise with syslog—if I had multiple sources, I might want to just show my syslog messages.

Using a log management solution has a number of benefits. You have one-stop shopping when it comes to looking at your information and if the log management solution is good, you will have a number of ways to look at your logs, using queries and filters. If your log management server is well-protected, you can also keep all of your logs intact and away from attackers. It also helps to maintain your logs in a useful order because sources are providing the data as it happens to the log management solution.

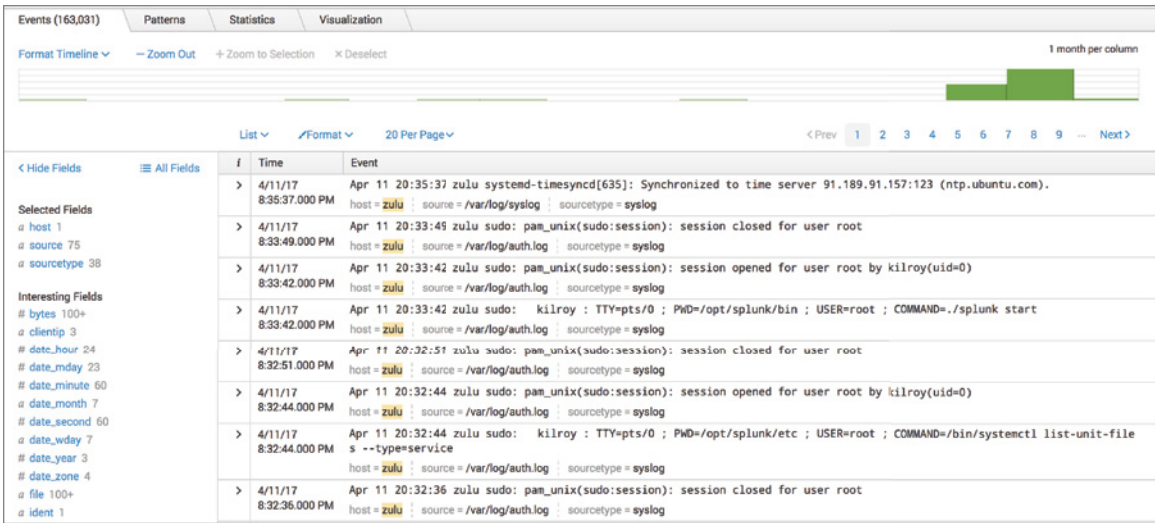


Figure 10-6: Displaying logs in Splunk.

Timelines

If you aren't using timelines in your work today, you probably remember them from school. One text book or another would have presented a timeline, whether it was a geological timeline showing the different periods the earth has been through or whether it was a historical timeline showing important events in the history of the country or the world. The idea of a timeline is to put events into a coherent, chronological order. When events are out of order, it's hard to follow what happened. As an example, Figure 10-7 shows a timeline but the order of events doesn't make any sense at all.



Figure 10-7: Illogical timeline.

According to the timeline in this figure, I drove to work, showered, and then got out of bed. I could drive to work then shower but I wouldn't be able to do either of the first two without doing the third. Without some additional contextual information, this timeline is illogical. When you get logs from disparate sources, you can end up with what appears to be an illogical sequence of events unless you can get everything into order by the actual date and time that they happened. The finer the grain you can get on the time the more precisely you can order the timeline. Computers are very fast and multiple events can happen within the same second. Having that fine granularity will give you the ability to put everything you believe has happened into a precisely accurate order.

A number of software products are available to help with timelines. There are also products you can use to generate your own timeline from what you have observed without necessarily having to extract events from your artifacts. As an example, the timeline shown in Figure 10-7 was created using Aeon Timeline. Using a simple tool like that, you can create a visual timeline and attach notes to each event. However, there are tools that you can use to automatically extract information from packet captures, logs, and other artifacts you may have acquired.

Plaso

Plaso is a collection of software that provides the ability to ingest a number of data types. A sample of the types of data that can be handled is shown in Figure 10-8. This list was produced by running `log2timeline.py --info`. One of the important components of Plaso is `log2timeline`, which does the work of taking the input, performing any parsing, and filtering then outputting a storage file that other tools like `psort` can use to generate output that you can use in your reports or in any other timeline you may want to create. Plaso is a collection of Python scripts and all of them are run from the command line. While you can output to different file types, including Excel spreadsheets, comma-separated values, and different database types, none of the outputs will cleanly create the sort of visual timeline shown earlier.

The output shown in Figure 10-8 was created on a Kali Linux system using the Plaso package. If you were to install it on your own or on a different system, you may get a different collection of input and output formats, depending on the dependencies that may be installed. Plaso relies on a number of libraries and other software packages to handle some of the work, so if those libraries are not installed some of the functionality Plaso is capable of won't be available.

***** Parsers *****	
Name	Description
android_app_usage	: Parser for Android usage-history.xml files.
asl_log	: Parser for ASL log files.
bencode	: Parser for bencoded files.
binary_cookies	: Parser for Safari Binary Cookie files.
bsm_log	: Parser for BSM log files.
chrome_cache	: Parser for Chrome Cache files.
chrome_preferences	: Parser for Chrome Preferences files.
cups_ipp	: Parser for CUPS IPP files.
custom_destinations	: Parser for *.customDestinations-ms files.
dockerjson	: Parser for JSON Docker files.
dpkg	: Parser for Debian dpkg.log files.
esedb	: Parser for Extensible Storage Engine (ESE) database files.
filestat	: Parser for file system stat information.
firefox_cache	: Parser for Firefox Cache version 1 files (Firefox 31 or earlier).
firefox_cache2	: Parser for Firefox Cache version 2 files (Firefox 32 or later).
hachoir	: Parser that wraps Hachoir.
java_idx	: Parser for Java WebStart Cache IDX files.
lnk	: Parser for Windows Shortcut (LNK) files.
mac_appfirewall_log	: Parser for appfirewall.log files.
mac_keychain	: Parser for Mac OS X Keychain files.
mac_securityd	: Parser for Mac OS X securityd log files.
mactime	: Parser for SleuthKit's mactime bodyfiles.
macwifi	: Parser for Mac OS X wifi.log files.
mcafee_protection	: Parser for McAfee AV Access Protection log files.
mft	: Parser for NTFS \$MFT metadata files.
msiecf	: Parser for MSIE Cache Files (MSIECF) also known as index.dat.
olecf	: Parser for OLE Compound Files (OLECF).
openxml	: Parser for OpenXML (OXML) files.
opera_global	: Parser for Opera global history.dat files.
opera_typed_history	: Parser for Opera typed_history.xml files.
pe	: Parser for Portable Executable (PE) files.
plist	: Parser for binary and text plist files.
pls_recall	: Parser for PL/SQL Recall files.

Figure 10-8: Plaso filetypes.

PacketTotal

Just like the website VirusTotal is capable of doing analysis on potential malware samples, PacketTotal can perform analysis on packet captures, presenting the information in a variety of forms. Figure 10-9 shows PacketTotal showing the list of all of the frames that have been captured. This is in chronological order, though it's probably not significantly more helpful than Wireshark is at just showing all of the communication that took place over the time period of the packet capture. However, in addition to just the list of all packets, PacketTotal has also extracted some other information, which can be

seen in the tabs along the top. In addition to just the connections, which you are seeing here, you can also view SSL Certificates, Transferred Files, and Strange Activity. There was strange activity in this particular capture but it was two messages that were not responded to, which isn't all that unusual, considering the capture may have been stopped before the response came back.

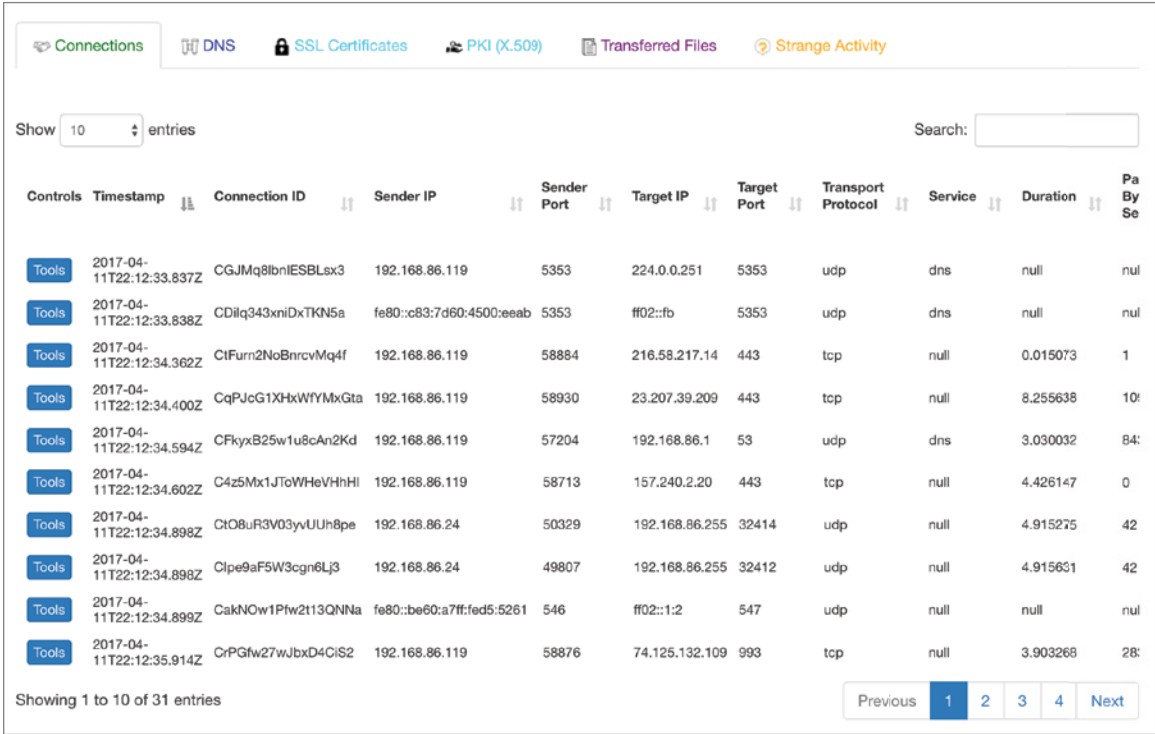


Figure 10-9: PacketTotal console view.

You haven't seen it all, though. PacketTotal is actually capable of generating the timeline for you if you didn't want to deal with the tedium. The timeline in Figure 10-10 shows all of the communication sessions that were found in the packet capture file that was provided. You are able to zoom in and out using your trackpad or scroll wheel on your mouse. If you zoom in, you can then scroll back and forward in time using the buttons on the page.

PacketTotal will also help perform some analysis. The Analytics page provides a number of charts and graphs that represent the different communication streams in different ways. For example, PacketTotal breaks out the communications by service and protocol as well as source and destination address and port. This is perhaps a better way of visualizing the packet capture since you can see clearly where your most active participants were during the time period of the packet capture.

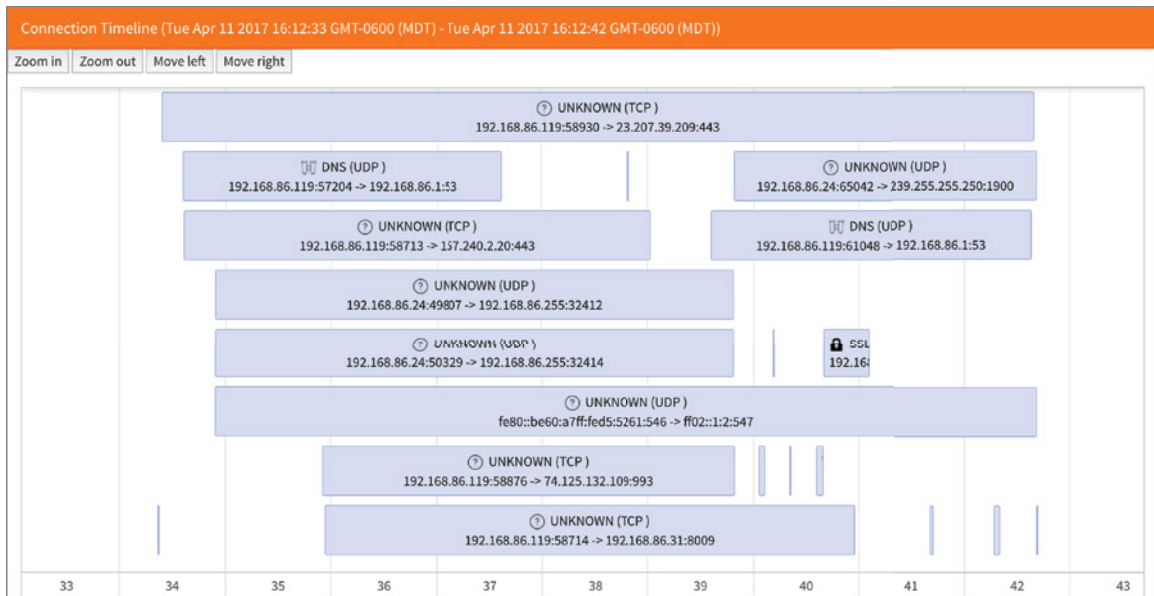


Figure 10-10: PacketTotal timeline view.

Wireshark

Wireshark has its own capabilities when it comes to generating time-based data that can help you present information within a timeline. Wireshark can't create a timeline for you other than the main Wireshark window that displays every frame captured in chronological order. However, it can represent information in different ways. One way is just using standard filters within Wireshark. A filter provides you a way to isolate just the specific packets you want to look at, which can limit extraneous noise. If you isolate down to a single conversation stream, you can see the order that the messages happened in. Wireshark will also decode the messages for you so you don't have to flip from one packet to another trying to read the payloads. If you Follow Stream, whether UDP or TCP, you will see the messages without all of the headers. That can help provide a better understanding of what happened within the course of the conversation. You would still need to refer back to the individual frames to get timestamps, however.

In addition to filters, the Statistics menu provides a number of functions that can be useful. The Conversations statistics, shown in Figure 10-11, provides a list of all of the conversations that took place. From the Conversations window, you can generate graphs, though the graphs are more oriented toward network analysts who are looking at specific protocol information. It is less oriented toward providing a timeline of the conversation. However, the graphs that are generated do show the timeline on the X-axis. The Y-axis is less likely to be interesting to you as a forensic investigator, however.

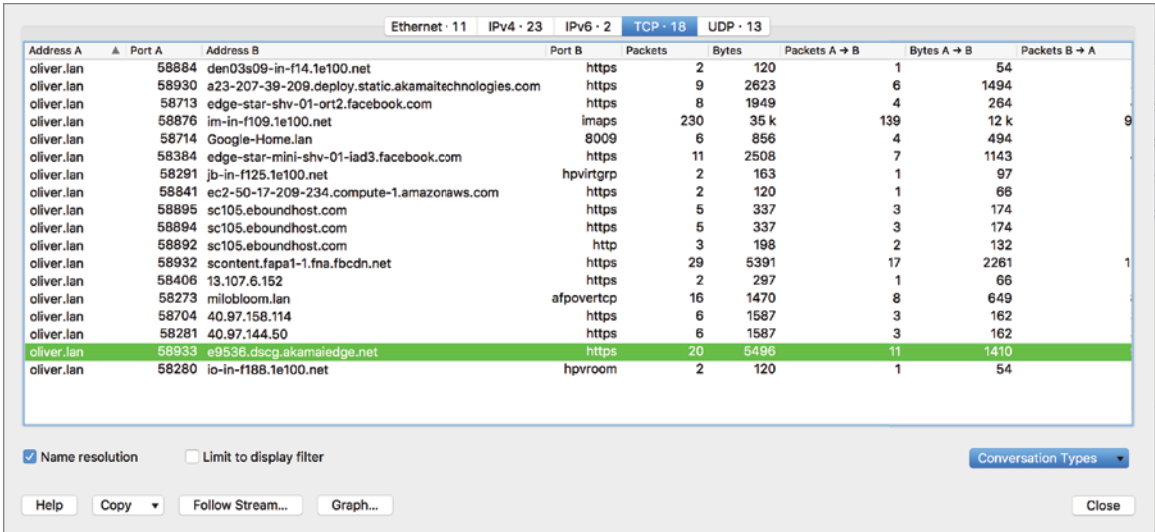


Figure 10-11: Wireshark conversation.

Wireshark’s Analyze menu also provides some sources of information, including the Expert Information, which provides warnings of packets that were potentially problematic on the network. In cases of low-level packet attacks, this information may be useful. For example, excessively fragmented packets will show up here as will packets where the headers appear to be incorrect, which may be evidence that they were tampered with.

Security Information and Event Management

We have talked about log management solutions. Similar to log management solutions is the Security Information and Event Management (SIEM) system. There is a lot of intersection between the log management solutions we talked about earlier and a SIEM solution. This is especially true when you are talking about one of the more full-featured solutions like Splunk. While there is overlap, to some degree, there are solutions that are specifically SIEMs. As an example, IBM has a SIEM named QRadar. McAfee has one called the Enterprise Security Manager.

Ultimately, what’s the difference between a SIEM and a log management solution? A log management solution is more generic. It takes a lot of data in and provides you with a way to search for information with your log storage. There can be a lot of value there but when it comes to security information, especially when it comes to incidents, security analysts will want to do more than just search for information. They want the system to be able to perform a lot of correlation of events to filter out a lot of the noise. Additionally, they want a workflow tool that can help track the incident from origination to resolution. This is primarily where a SIEM would differ from a log management solution.

This is not to say that the heavyweights in the log management space don't have capabilities around incident work flows, like generating alerts and handling them, but the object of a SIEM is to focus specifically on security-related information. This may include logs, it may include alerts from other systems like antivirus, and it may include integration with other security software products in order to assist in the troubleshooting and resolution of the security incident.

A SIEM can also serve as a central log storage system and provide capabilities to query the information that it stores. It should also be able to generate reports based on the information that it has acquired from the endpoints, servers, and various software solutions that have been integrated with it. These reports may also provide you with data that you can use to create a timeline.

Most importantly, the SIEM will do a lot of correlation for you so you don't have to pull a lot of data together yourself. This typically requires writing rules to tell the SIEM what data should be put together to trigger an action. As an example, IBM's QRadar considers two types of data that you can create rules from. One is event data, which would be log sources, and the other is flow data. Flow data would be generated from network devices. If you pass flow data to your SIEM, it can be used along with your log data to trigger a result to the rule you have created.

Each SIEM will generate rules in different ways, of course, and some may include some default rules, while others may provide tools that allow you to create complex, fine-grained rules to do much of the work of sifting through a lot of data for you. That is one of the major advantages to making use of a SIEM—letting it do a lot of the grunt work for you and leaving you free to just follow up on something that looks like it may be an incident.

Summary

As you are working on your analysis, you will find that you have a large amount of data to sift through. This may come from logs, packet captures, network flow information, or perhaps even alerts from other systems. To make sense of all of this data, you need to be able to correlate it in some way. One way of doing that is to just simply put everything into a chronological order and see what lines up. You may find data from entirely different sources suddenly makes sense if you can see it all in a coherent way with times lined up correctly.

This can be harder if your systems are in different time zones and their clocks are all set to those time zones. While it can be done, it may be simply easier to set all of your important systems like network devices and servers to UTC time so that no matter where your data comes from, it will all be stamped the same and there is no additional effort required to pull it together into a single timeline. While you are working from a single time zone, you may still have clocks that have drifted. Using NTP to make sure your system clocks are all synched to a time standard, like one of the ntp.org pool clocks, will help ensure that you don't have one system 10 minutes off from the others, which will skew the results of your investigation.

Creating timelines can be challenging work, but you can get some tools that can help you. Plaso, for instance, is an open source program that includes a number of scripts that will ingest data from

different sources and create a coherent timeline of events in different output types. Plaso does require that you are comfortable with the command line and, ideally, Linux, just because of the way it was written and is available. This doesn't mean you can't run it under Windows, but because it is written in the scripting language Python and makes use of a number of libraries that are more commonly found on Linux, it may be a more natural fit there. However, lots of things are possible and many do use it under Windows.

PacketTotal is a website that is capable of generating timelines from a packet capture. You can create a timeline chart using the web interface and that chart can be helpful in visualizing the network communication. Wireshark can also help you visualize the network capture in different ways. The statistics and analysis capabilities of Wireshark are deep.

Finally, get yourself some help. Log management solutions and SIEM solutions can take a lot of the work out of extracting useful information from the number of sources you are going to have. A tool like a SIEM can also be used to create rules that can help you correlate events from different sources, including logs and network devices. Once you have written the rule, you can have the SIEM do the dirty work of digging through the pile of information that you have available. Some of this may also be possible using a log management solution, depending on the solution you have. A lot of feature-rich log management solutions are available.

11

Network Scanning

In this chapter you will learn about:

- How to identify open ports and protocols on remote systems
- How to remotely identify system vulnerabilities
- Passively gathering information about remote systems
- Port knocking and protocol tunneling

He's put his backdoor into place and installed some additional software to make it harder for anyone to find the port that's listening for him to come in. Had he needed to, he could have left a permanent connection to another system he controlled. This would have allowed him to tunnel back into this system, though the firewall. That connection would have been, perhaps, more obvious, but hiding his connections is always necessary. Fortunately, he has tools that can do that for him. Anyone who is also on the system won't see his network connections.

He also worried about someone else coming into the system he now possesses. Fixing some of the broken software packages by patching them would keep others from making use of any vulnerabilities that existed. While he was able to get into this system by way of an e-mail, he has had to use vulnerabilities to get into other systems. There is no need for a turf war here. Just fix the vulnerable software so he would have this system all to himself.

The challenge with host-based analysis is that you can't trust the host operating system if the system has been compromised. We've talked about this in earlier chapters of this book, but it's worth repeating. *The best place to gather information is from the network.* This is true about information on the host, sometimes, as well. You may commonly check for open ports on a system by using tools like netstat or maybe one of the Windows SysInternals tools and using those tools, you can get a list of all of the open ports. If the attacker has installed malware that either replaced the tools or hijacked system calls, he can obscure the details about the network connection. This means you can't trust the tools on the operating system to tell you that something is happening over the network that you should know about.

Fortunately, there are remote ways to see what is happening for at least some of what we are talking about. For example, if there is an existing network connection, the only way to see that is by grabbing packets. But if there are listening backdoors on a system, they should usually be easy to see. However, there are ways to prevent even port scanners from seeing where there may be a backdoor. You can also use a similar technique to port scanning called ping sweeping, which is a way of quickly determining whether there are systems responding at the IP addresses within a given range. This can be used to see if you have the right systems on the network responding. Not every system on the network should necessarily be there.

A common way for attackers to get into systems is through vulnerabilities. While we are talking about probing systems to see what is going on with them, we may as well talk about vulnerability scanners. This may be a way of seeing whether there are vulnerabilities that an attacker has either already taken advantage of, which can help you to identify a way in, or identify an avenue of potential attack. This may be especially useful across entire enterprise networks where a single vulnerability may be common across multiple systems. If you find a vulnerability and it exists across multiple systems, you can be assured that an attacker will find it as well and gain access to those systems.

Attackers can obscure their access attempts in some ways so they don't look like attacks. One way to do this is through *tunneling*. Tunneled traffic can be a way to bypass firewalls, as well, since you may be able to tunnel your attack traffic inside protocols/ports that are allowed through the firewall.

NOTE Scanning can be considered a hostile act in some cases. When you are doing any sort of scanning, whether it's port scanning or vulnerability scanning, make sure you are either working on your own systems or you have permission to scan the systems that you are targeting. There is a possibility of finding yourself in trouble otherwise.

Port Scanning

Port scanning is a way of identifying ports that are listening on the target system. This can be one way of identifying whether ports are open. If you wanted to see what ports were open on a system, you might use the `netstat` command. On newer versions of Linux, you would use the `ss` command. As an example, you can see the output from `ss -l` on a CentOS Linux system in Listing 11-1. It shows all of the instances where there are programs listening for connections.

Listing 11-1: `ss -l` Output from CentOS Linux System

Netid	State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port
n1	UNCONN	0	0	rtln:systemd-network/6929	*
n1	UNCONN	0	0	rtln:kernel	*
n1	UNCONN	0	0	rtln:systemd-network/6929	*
u_dgr	UNCONN	0	0	* 109699	* 15627

```

u_dgr UNCONN 0 0 * 26584 * 15627
u_dgr UNCONN 0 0 * 101548 * 101549
u_dgr UNCONN 0 0 * 22591 * 15627
u_dgr UNCONN 0 0 * 16231 * 15612
u_dgr UNCONN 0 0 * 21842 * 15981
u_dgr UNCONN 0 0 * 380882 * 15981
u_dgr UNCONN 0 0 * 417168 * 15981
u_dgr UNCONN 0 0 * 111079 * 15981
u_dgr UNCONN 0 0 * 101551 * 101550
u_dgr UNCONN 0 0 * 101544 * 15627
u_dgr UNCONN 0 0 * 22480 * 15981
u_dgr UNCONN 0 0 * 101549 * 101548
u_dgr UNCONN 0 0 * 394801 * 15627
u_dgr UNCONN 0 0 * 26431 * 15981
u_dgr UNCONN 0 0 * 109703 * 109704
u_dgr UNCONN 0 0 * 26569 * 15981
u_dgr UNCONN 0 0 * 109704 * 109703
u_dgr UNCONN 0 0 * 412426 * 15627
u_dgr UNCONN 0 0 * 111107 * 15981
u_dgr UNCONN 0 0 * 412545 * 15981
u_dgr UNCONN 0 0 * 101550 * 101551
udp UNCONN 0 0 *:bootpc *: *
udp UNCONN 0 0 127.0.0.1:syslog *: *
tcp LISTEN 0 80 127.0.0.1:mysql *: *
tcp LISTEN 0 128 *:ssh *: *
tcp LISTEN 0 128 127.0.0.1:shell *: *
tcp LISTEN 0 128 :::http ::: *
tcp LISTEN 0 128 :::ssh ::: *

```

This output shows both TCP and UDP as well as interprocess communications (IPC) where programs on the same system can communicate with one another. You can see the listening services for both TCP and UDP from this list. However, just because there are listening services on the system doesn't mean that someone remotely is going to get the same results if they were to check for open ports. First, if the system were compromised, the output you get from `ss` or `netstat` may be corrupted by what the attacker did. Second, there may be a host-based firewall or even a network firewall blocking connection requests, depending on where you were doing the probing from.

While there are a few different port scanning programs, the one that most use is `nmap` (Network Mapper). If you watch TV shows or movies, you may have seen `nmap` in action. Famously, `nmap` was used in *The Matrix Reloaded*. `Nmap` is capable of detecting open ports in both TCP and UDP but it also has a number of other powerful capabilities as well. If you are not familiar with `nmap`, you should become familiar. We're going to go through some of `nmap`'s capabilities here, but we'll just scratch the surface of what `nmap` can do.

First, if you want to just get the open TCP ports, one easy way is through a technique called a SYN scan. The SYN scan takes advantage of the three-way handshake mechanism of TCP in order to determine whether ports are open. In a normal three-way handshake, the client sends a SYN message, which is followed by the server responding with a SYN/ACK, to which the client replies with

an ACK. Because of the way TCP is designed, a system that has an open port will respond with the SYN/ACK, otherwise it will respond with a RST message, indicating that there is nothing there. As a result, a SYN scan is very economical. You can determine whether a port is open with the minimum number of packets sent and received. Of course, if you were to just leave a number of half-open ports on the system, it would be seen as suspicious so nmap politely responds with a RST after the SYN/ACK, which effectively says “never mind” to the server, which then releases the connection it has held half-open.

A SYN scan is done in nmap by using `-sS` on the command line. The results of a SYN scan are shown in Listing 11-2.

Listing 11-2: nmap SYN Scan Results

```
kilroy@oliver:~$ sudo nmap -sS 192.168.86.106

Starting Nmap 7.00 ( https://nmap.org ) at 2017-03-21 16:37 MDT
Nmap scan report for 192.168.86.106
Host is up (0.014s latency).
Not shown: 990 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
427/tcp   open  svrloc
443/tcp   open  https
902/tcp   open  iss-realsecure
5988/tcp  closed wbem-http
5989/tcp  closed wbem-https
8000/tcp  open  http-alt
8300/tcp  open  tmi
9080/tcp  open  glrpc
MAC Address: 68:05:CA:46:70:88 (Intel Corporate)

Nmap done: 1 IP address (1 host up) scanned in 4.33 seconds
```

You may notice that in order to run nmap here, I used `sudo`, which grants me temporary administrative privileges. This is necessary when running a SYN scan. The reason is that nmap isn't just using typical connection mechanisms to do this because stopping after just the SYN isn't considered normal. As a result, it uses something called *raw sockets*, where the program defines what the packet looks like from top to bottom and then the operating system just sends the result out onto the wire without bothering to do anything else about it. Raw sockets require administrative privileges, though. As a result, I need to run this with elevated privileges and `sudo` gives that to me. We could achieve the same result with a *connect scan*, which completes the three-way handshake before tearing the connection down. It requires slightly more packets to accomplish, but the results will be the same. A connect scan from nmap is shown in Listing 11-3.

Listing 11-3: nmap Connect Scan Results

```
kilroy@oliver:~$ nmap -sT 192.168.86.106

Starting Nmap 7.00 ( https://nmap.org ) at 2017-03-21 16:41 MDT
Nmap scan report for 192.168.86.106
Host is up (0.020s latency).
Not shown: 990 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
427/tcp   open  svrloc
443/tcp   open  https
902/tcp   open  iss-realsecure
5988/tcp  closed wbem-http
5989/tcp  closed wbem-https
8000/tcp  open  http-alt
8300/tcp  open  tmi
9080/tcp  open  glrpc

Nmap done: 1 IP address (1 host up) scanned in 4.30 seconds
```

This time, I didn't need `sudo` and I got the same list of open ports. In order to do the connect scan, I used `-sT` instead of `-sS`. You can see the list of open ports on the left-hand side of the results and the name of the service on the right. This is not necessarily the application that is listening, though. This is just the service that is defined by the port number. These are well-known ports, and services have been attached to each one of them. Nmap is just doing a lookup from a services file to get the name of the service, without doing anything beyond that. In order to get the name of the service, we need to do something else with nmap, and we will get to that later on.

You will see that nmap indicates whether the port is open or closed. In most cases, you will only see the open ports listed. In this case, you can see two ports that were identified as being closed. Nmap found two ports that responded to the requests it sent but there is no application listening. Nmap uses closed ports to help with other detections it is capable of. You can also see in the output that nmap only scanned 1000 ports. This is by default. In order to keep scan times down, nmap has defined 1000 well-known ports that it will scan rather than scanning all 65,536 ports that exist in a system.

You can select the ports you want to scan by telling nmap. As an example, if I just wanted to scan web ports, I would say `-p 80,443`. I could also scan ports related to Windows file sharing by using `-p 137-139`. Nmap will take both lists, separated by commas, or ranges, separated by a hyphen. I can also say that I want to scan all ports by either specifying the range like `-p 0-65535` or by just saying `-p-`. The second hyphen just tells nmap to scan everything. In most cases, you are going to be okay scanning the well-known ports that nmap scans by default. However, if you are looking for backdoors, you may want to scan the entire range of ports because they may be hidden on an odd port somewhere.

Nmap can also do some additional scan types that target TCP. In addition to the `SYN` and `ACK` flags, other flags are defined by TCP. Nmap can set different flags and scan using those. Depending on how the system responds, nmap will determine whether the port is open or closed. For instance, nmap can do a `FIN` scan, where it only sets the `FIN` flag. This should only be used on an open connection, and if we are running a port scan we have no open connections. With a `FIN` scan, a `RST` message in response means the port is closed. If there is no response, the assumption is that either the port is open or there is a firewall of some sort just dropping the request and not responding to it. As a result, nmap flags these ports as either open or filtered because it can't tell definitively which may be the case.

A null scan looks for the same behaviors as the `FIN` scan. Instead of setting the `FIN` flag, however, it doesn't set any flags. The target system will send a `RST` on closed ports and ignore the message on open ports. Another scan type that uses TCP flags is the `Xmas` scan. It is referred to as an `Xmas` scan because it lights up the `FIN`, `PSH`, and `URG` flags just like a Christmas tree in the TCP header. The scan types that set specific TCP flags also require administrative privileges because, again, the behavior is different from what would normally be expected. You can see the results of an `Xmas` scan in Listing 11-4 and in most cases, you will see the same results as you did with a `SYN` scan or a connect scan. In this case, the target operating system appears to ignore our `Xmas` scan. The only results we get back are the two ports previously identified as closed.

Listing 11-4: Xmas Scan Results

```
kilroy@oliver:~$ sudo nmap -sX 192.168.86.106

Starting Nmap 7.00 ( https://nmap.org ) at 2017-03-21 16:58 MDT
Nmap scan report for 192.168.86.106
Host is up (0.044s latency).
Not shown: 998 open|filtered ports
PORT      STATE SERVICE
5988/tcp  closed wbem-http
5989/tcp  closed wbem-https
MAC Address: 68:05:CA:46:70:88 (Intel Corporate)

Nmap done: 1 IP address (1 host up) scanned in 12.16 seconds
```

You will also notice that the scan took longer. If you look more closely, you will see that nmap found 998 open or filtered ports. This means it didn't get any response on those ports, which may mean that the request packet got lost. To protect against that, nmap will retransmit the request. These retransmits slow the scan down. Any time you have a scan where you are not getting responses, your scan will be slower.

Along those lines, we can do UDP scans just as well as TCP scans. A UDP scan is a UDP scan. There are no variations because UDP is a very simple protocol. It has no flags to manipulate or any other headers that can be manipulated in any way. In order to perform a UDP scan, you just run

nmap with `-sU` as the flag. UDP scans will also take longer than a typical TCP scan because with UDP, there is no handshake and no way to determine for sure whether the packet was received. If a message is sent to a UDP port, it could just be that the message was ignored by the application. Since UDP datagrams are often considered less valuable than TCP, they can just get lost or dropped. This means that nmap can't guarantee that a UDP message has really gotten through. As a result, it has to retransmit to make sure that the application really has received it and the port is closed. The results of a UDP scan are shown in Listing 11-5.

Listing 11-5: nmap UDP Scan

```
kilroy@oliver:~$ sudo nmap -sU 192.168.86.106

Starting Nmap 7.00 ( https://nmap.org ) at 2017-03-21 17:08 MDT
Nmap scan report for 192.168.86.106
Host is up (0.0075s latency).
Not shown: 997 open|filtered ports
PORT      STATE SERVICE
53/udp    closed domain
161/udp   closed snmp
427/udp   open  svrloc
MAC Address: 68:05:CA:46:70:88 (Intel Corporate)

Nmap done: 1 IP address (1 host up) scanned in 10.15 seconds
```

If you were to scan all UDP ports, you could end up waiting hours, if not days, for the results. There are a number of variables that can get in the way of a UDP scan and certainly network latency is one of those. In my case, I am scanning systems on my own network. This means I have very low latency and fast responses. If you are scanning systems that are not directly on your local network, it will take longer to complete the scan.

Operating System Analysis

In addition to just determining open ports on target systems, you can determine what operating system may be in use. Nmap includes a database of fingerprints for different operating systems. It runs dozens of tests against responses from the target, comparing how the initial sequence number behaves, how packets are ordered, and initial window size selection, among other checks. Once it has all of the information from the target, it can check the results against its database to find a match. Sometimes, nmap can be very specific about the operating system name and version. Other times, it can only make a best guess. While the nmap team has made a deliberate attempt to increase the size of their database, there may still be times when they can't guess the operating system in use.

In order to do an operating system scan, nmap needs to find both open and closed ports in both TCP and UDP. If you have a system you are scanning that has no open TCP ports, nmap will not be able to determine the operating system. This is because it will not be able to determine how the

sequence number behaves. Two scans showing the operating system detection against different operating systems are shown in Listing 11-6.

Listing 11-6: nmap Operating System Scan Results

```
kilroy@oliver:~$ sudo nmap -O 192.168.86.106

Starting Nmap 7.00 ( https://nmap.org ) at 2017-03-21 17:17 MDT
Nmap scan report for 192.168.86.106
Host is up (0.0068s latency).
Not shown: 990 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
427/tcp   open  svrloc
443/tcp   open  https
902/tcp   open  iss-realsecure
5988/tcp  closed wbem-http
5989/tcp  closed wbem-https
8000/tcp  open  http-alt
8300/tcp  open  tmi
9080/tcp  open  glrpc
MAC Address: 68:05:CA:46:70:88 (Intel Corporate)
Aggressive OS guesses: VMware ESXi 5.0 - 5.5 (97%), VMware ESXi 6.0.0
(95%), VMware ESXi 4.1 (92%), Crestron XPanel control system (92%),
FreeBSD 7.0-RELEASE-p1 - 10.0-CURRENT (92%), VMware ESXi 5.5 (92%),
FreeBSD 5.3 - 5.5 (91%), VMware ESX Server 4.0.1 (91%),
FreeBSD 5.2.1-RELEASE (90%), FreeNAS 0.686 (FreeBSD 6.2-RELEASE) or
VMware ESXi Server 3.0 - 4.0 (90%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.55 seconds
kilroy@oliver:~$ sudo nmap -O 192.168.86.1

Starting Nmap 7.00 ( https://nmap.org ) at 2017-03-21 17:17 MDT
Nmap scan report for testwifi.here (192.168.86.1)
Host is up (0.0084s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
5000/tcp  open  upnp
MAC Address: 18:D6:C7:7D:F4:8A (Unknown)
Device type: general purpose
Running: Linux 3.X|4.X
```

```
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.0
Network Distance: 1 hop
```

```
OS detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.55 seconds
```

In the first case, nmap couldn't determine the OS for certain so it made several guesses. It wasn't exactly correct in any of them, though it was closest in the first two. The system was running VMWare ESXi version 6.5. It could be that the database in use is out of date since 6.5 is a newer version of VMWare ESXi. The second system scanned is the router on my network. We did find some open ports and then nmap made some guesses. All it was able to determine was that the kernel was a version of Linux but it doesn't know anything beyond that. This isn't particularly surprising since this would be an embedded device that would only be using the Linux kernel and it wouldn't have any particular distribution.

Since all Linux distributions use the same kernel source, you may find that Linux distributions are harder to determine. In order to get differences, the kernel would need to be modified in some way by the distribution maintainers. Otherwise, there would need to be some other signature that is unique to a particular distribution. Remember that the networking stack on any system lives inside the kernel (operating system) and not in any program that sits on top of that software. If all the kernels are the same, there is no way to differentiate from one distribution to another.

Scripts

Nmap includes a scripting engine, based on the Lua programming language. The Nmap scripting engine (NSE) can be used to probe the target system more deeply. Nmap provides functions that can be used to quickly develop your own scripts. You tell nmap which ports you care about so when one of those ports is identified as being open, your script gets called so you can do additional investigation.

In addition to being able to write your own scripts, nmap comes with a library of scripts that can be used against your targets. When you run nmap, you would typically specify which script to use, but you can also run a number of scripts. Because each script defines categories that it belongs to, you could select all of the scripts in a particular category. You can also select by patterns. As an example, Listing 11-7 shows nmap running all of the scripts matching the pattern `smtp*`. This means that all of the script filenames that start with `smtp` will be executed.

Listing 11-7: nmap SMTP Scripts

```
kilroy@zulu:~$ sudo nmap --script "smtp*" 192.168.86.111
```

```
Starting Nmap 7.01 ( https://nmap.org ) at 2017-03-21 22:08 EDT
Nmap scan report for proxy.lan (192.168.86.111)
```

```

Host is up (0.0073s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
25/tcp    open  smtp
|_smtp-commands: proxy.washere.com, PIPELINING, SIZE 10240000, VRFY,
ETRN, ENHANCEDSTATUSCODES, 8BITMIME, DSN,
| smtp-enum-users:
|   root
|   admin
|   administrator
|   webadmin
|   sysadmin
|   netadmin
|   guest
|   user
|   web
|_ test
|_smtp-open-relay: Server is an open relay (16/16 tests)
| smtp-vuln-cve2010-4344:
|_ The SMTP server is not Exim: NOT VULNERABLE
8080/tcp  open  http-proxy
MAC Address: 00:0C:29:02:2B:82 (VMware)

```

Nmap done: 1 IP address (1 host up) scanned in 40.04 seconds

The scripts that were run here were `smtp-enumusers`, `smtp-strangeport`, `smtp-commands`, `smtp-vuln-cve2011-1720`, `smtp-vuln-cve2011-1764`, `smtp-brute`, `smtp-vuln-cve2010-4344`, and `smtp-open-relay`. They enumerate any users that may be available, look for an SMTP server running on a non-standard port, look for specific vulnerabilities, and determine if the server is an open relay. An SMTP open relay would allow unauthorized users to send mail through it. The current version of `nmap` includes more than 500 scripts. Listing 11-8 shows a script run against an SSH server, as another example.

Listing 11-8: nmap Script Output

```

kilroy@zulu:~$ sudo nmap --script=ssh-hostkey.nse 192.168.86.111

Starting Nmap 7.01 ( https://nmap.org ) at 2017-03-21 22:00 EDT
Nmap scan report for proxy.lan (192.168.86.111)
Host is up (0.0086s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
| ssh-hostkey:
|   2048 44:45:f9:0b:bd:37:5c:be:8e:89:38:ea:e1:e8:bc:16 (RSA)

```

```
|_ 256 51:b1:cb:43:34:8c:03:cd:e8:0d:f1:d2:f8:93:98:5c (ECDSA)
25/tcp open  smtp
8080/tcp open  http-proxy
MAC Address: 00:0C:29:02:2B:82 (VMware)
```

Since the SSH server uses public key encryption, we can extract the public key. This is the key anyone can have because it requires the private key to decrypt anything encrypted with the public key. The public key does provide a fingerprint that can be used to identify this server since encryption keys are not supposed to be used across multiple systems. This doesn't mean that people always follow that guideline. Sometimes keys can be shared across multiple systems since they are just files that sit on the disk and the server gets configured to know where they are. Using *nmap*, you can obtain the fingerprint from any server that is using SSH.

If there isn't an existing NSE script for your purposes, all of the scripts are available on any system where *nmap* is installed and any of those scripts could be used as a template to create a new one. With a little programming experience, it shouldn't be too complicated to make adjustments to one of the scripts to get a new one that does what you want it to do. However, with over 500 scripts currently available, you may well be able to find a script that will gather a lot of information from the remote host.

Banner Grabbing

Banner grabbing is the process of connecting to a service and probing it to get the service banner. Different protocols support different ways of conveying information about themselves. Listing 11-9 shows the banner that is provided from an SSH server. Even though SSH is an encrypted protocol, the initial banner on connection is in plaintext, ahead of the encryption negotiation. This banner provides us with the protocol version (2.0) as well as the name of the software and the version in use. Additionally, we can see that the operating system this software is running on is Ubuntu. This information is one of the very reasons that we use banner grabbing—services can be overly communicative about themselves.

Listing 11-9: SSH Banner

```
kilroy@oliver:~$ nc 192.168.86.83 22
SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.1
```

The way we obtained this banner was through the use of a program named *netcat* (sometimes referred to as *nc* and also *ncat* in the *nmap* distribution), which provides us with a raw TCP connection to the service port. Once the connection has been negotiated, *netcat* gets out of the way and the port is just there to send messages to. In some cases, you need to be able to communicate with the underlying protocol in order to get a response. Other services, like SSH, require no initial message to get a response. As an example of a service that requires a message from the client side before getting

a response, you can see a connection to a web server. There is no indication that you are even connected using netcat. Instead, the web server waits patiently for the client to initiate a request. The request shown in Listing 11-10 is for the primary index page at the root of the web server. If a web server has an index page, commonly `index.html` or `index.htm`, and the server is configured to offer up that index page, which most servers are configured to do by default, you don't have to specify the index page, just the directory.

Listing 11-10: HTTP Message for Web Server Banner

```
kilroy@oliver:~$ nc 192.168.86.83 80
GET / HTTP/1.1
Host: foo

HTTP/1.1 302 Found
Date: Sat, 25 Mar 2017 20:41:57 GMT
Server: Apache/2.4.18
Set-Cookie: PHPSESSID=t5goj0t77d4smjj30537t7t513; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=t5goj0t77d4smjj30537t7t513; path=/; HttpOnly
Set-Cookie: security=impossible; HttpOnly
Location: login.php
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

The request is to GET the page located at / using HTTP protocol version 1.1. Version 1.1 supports virtual servers so in order to get the right root page back, we have to indicate the hostname we are looking for. Since I know that this particular server has no virtual hosts configured, I know it doesn't matter what I specify in the host field because there is only one host. The server here provides us with the server identification details, telling us the type of web server (Apache) and the version number (2.4.18). Additionally, we know that this server supports PHP, which is a programming language often used in web application development. In fact, not only do we know that the server supports PHP, we know that the default page for this directory is actually `login.php`. We can tell that because 302 is a redirect message and the `Location:` field tells us where the correct page can be located. This is something your browser would take care of automatically and you won't even see it happening.

I did both of these by hand but that is a two-step process, because you first need to identify an open port and then you need to connect to the port and try to get it to respond. This is another area where we can get nmap to help us. Using a version scan, nmap will not only identify open ports, but it will retrieve the banners and provide just the relevant information about the version of software being used. Listing 11-11 shows a version scan using nmap. Again, nmap only scans about 1000 ports by default so if you want to check for anything on a non-default port, you need to specify a wider

range of ports. Backdoors or other services that have been installed by attackers may show up on oddball ports because of the tendency of port scanners like `nmap` to just look on well-known ports. If all you are doing is scanning the default set of ports, you may well be missing a service that was installed by someone else, and the attackers are aware of these behaviors.

Listing 11-11: `nmap` Version Scan

```
kilroy@oliver:~$ sudo nmap -sV 192.168.86.111

Starting Nmap 7.00 ( https://nmap.org ) at 2017-03-25 14:52 MDT
Nmap scan report for proxy.lan (192.168.86.111)
Host is up (0.0082s latency).
Not shown: 995 closed ports
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          vsftpd 3.0.2
22/tcp    open  ssh          OpenSSH 6.6.1 (protocol 2.0)
25/tcp    open  smtp         Postfix smtpd
80/tcp    open  http         Apache httpd 2.4.6 ((CentOS))
8080/tcp  open  http-proxy   Squid http proxy 3.5.20
MAC Address: 00:0C:29:02:2B:82 (VMware)
Service Info: Host: proxy.washere.com; OS: Unix

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.80 seconds
```

Using the version scan, `nmap` was able to identify some information we had already identified, like the fact that the target was running Apache version 2.4.6. In addition to that information, `nmap` was able to determine that Apache is running on CentOS. Even if you are able to determine that ports are open, there is no guarantee that the service you connect to will respond to anything. What we have identified here are a number of ports that have well-behaved service applications listening. The protocols they are using to communicate may even specify that they need to provide some amount of information in their banners and communications with their clients. The reason for this is because clients and servers may determine they need to communicate differently depending on capabilities that are in use by the server. One example is shown in Listing 11-12. The mail server responds to an initial message with a list of capabilities that it supports. This lets the client know how it should go about engaging with the server.

Listing 11-12: SMTP Capabilities

```
kilroy@oliver:~$ telnet 192.168.86.111 25
Trying 192.168.86.111...
Connected to proxy.lan.
```

```

Escape character is '^]'.
220 proxy.washere.com ESMTP Postfix
EHLO blah.com
250-proxy.washere.com
250-PIPELINING
250-SIZE 10240000
250-VERFY
250-ETRN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN

```

You'll notice that I used the telnet client in place of netcat. The telnet client can also offer a raw TCP connection and I have a habit of using telnet. I was using it before netcat was available, and because the telnet client is more likely to be installed on a Linux system than netcat, I usually don't have to worry about installing another package before opening the connection.

A service that has been installed to provide an attacker access that the attacker doesn't want anyone to know about will likely be less forthcoming when it comes to providing information to anyone connecting.

Ping Sweeps

A *ping sweep* is a way of identifying all of the hosts that are discoverable on a network. There are a number of ways to do this. One way is using what nmap calls *host discovery*. You can see an example of host discovery using nmap in Listing 11-13. For this, I have told nmap that I don't want any information other than just the list of hosts that show as being up and available. I do that by using `-sn`, telling nmap not to scan anything.

Listing 11-13: Host Discovery Using nmap

```

kilroy@oliver:~$ nmap -sn 192.168.86.0/24

Starting Nmap 7.00 ( https://nmap.org ) at 2017-03-25 15:13 MDT
Nmap scan report for testwifi.here (192.168.86.1)
Host is up (0.0086s latency).
Nmap scan report for milobloom.lan (192.168.86.23)
Host is up (0.0082s latency).
Nmap scan report for billthecat.lan (192.168.86.24)
Host is up (0.0082s latency).
Nmap scan report for NPI110654.lan (192.168.86.28)
Host is up (0.0064s latency).
Nmap scan report for Google-Home.lan (192.168.86.31)
Host is up (0.015s latency).
Nmap scan report for SonosZP.lan (192.168.86.45)
Host is up (0.035s latency).

```



```

Nmap scan report for 192.168.86.51
Host is up (0.0070s latency).
Nmap scan report for oliver.lan (192.168.86.65)
Host is up (0.00033s latency).
Nmap scan report for zulu.lan (192.168.86.83)
Host is up (0.0013s latency).
Nmap scan report for 192.168.86.106
Host is up (0.015s latency).
Nmap scan report for boingers.lan (192.168.86.110)
Host is up (0.014s latency).
Nmap scan report for proxy.lan (192.168.86.111)
Host is up (0.0050s latency).
Nmap scan report for overbeek.lan (192.168.86.112)
Host is up (0.0061s latency).
Nmap done: 256 IP addresses (13 hosts up) scanned in 2.41 seconds

```

You can also do a ping scan using `nmap`, which just requires that you use `-sP` for your scan type. You will get the same results back, because `nmap` is doing the same thing. We are just being more explicit in the case of `-sP`.

Other tools like `fping` could be used to do a ping sweep. Using `fping`, you would need to either create a list of IP addresses you want to scan or you would have `fping` generate the list for you. An example of having `fping` generate the list from a CIDR notation can be seen in Listing 11-14. The output here is very straightforward. It just tells you which hosts respond as up. Where `nmap` has provided additional information like the network latency from the response, `fping` is very straightforward and matter-of-fact, indicating hosts that are alive. You will also get a list of all of the IP addresses that were scanned that did not respond. This shows up after the list of those that were deemed to be alive. You can see a partial list of those in the listing. Since we were scanning 254 addresses, the entire list is quite a bit longer.

Listing 11-14: `fping` Output

```

kilroy@oliver:~$ fping -g 192.168.86.0/24
192.168.86.1 is alive
192.168.86.23 is alive
192.168.86.24 is alive
192.168.86.28 is alive
192.168.86.31 is alive
192.168.86.45 is alive
192.168.86.38 is alive
192.168.86.49 is alive
192.168.86.51 is alive
192.168.86.65 is alive
192.168.86.83 is alive
192.168.86.85 is alive
192.168.86.60 is alive

```

```
192.168.86.106 is alive
192.168.86.110 is alive
192.168.86.111 is alive
192.168.86.112 is alive
192.168.86.2 is unreachable
192.168.86.3 is unreachable
192.168.86.4 is unreachable
192.168.86.5 is unreachable
192.168.86.6 is unreachable
192.168.86.7 is unreachable
192.168.86.8 is unreachable
192.168.86.9 is unreachable
192.168.86.10 is unreachable
192.168.86.11 is unreachable
192.168.86.12 is unreachable
192.168.86.13 is unreachable
192.168.86.14 is unreachable
```

Ping sweeps or ping scans are very simple techniques of identifying systems that are on your network that respond to ICMP requests. If a host has somehow blocked messages or just ignores ICMP requests, they won't show up using these techniques. If that's the case, you may need to do a full-blown nmap scan of the network to see whether there are some hosts that respond on ports without responding to ICMP messages. Even then, though, nmap prefers to use ICMP to determine aliveness before bothering to do any port scanning. If you are using nmap to do a scan to see if there are hosts that are not responding to ICMP messages, then you would need to specify `-Pn` on the command line to tell nmap not to ping first.

Vulnerability Scanning

While there are a number of ways for an attacker to gain access to systems, one of those ways is through vulnerabilities. As a result, it can be beneficial to have an idea of what your systems look like from a vulnerability perspective from the outside. Since vulnerabilities are constantly changing as new ones are discovered and old ones get fixed with updates from the software vendors, it's useful to check on the vulnerabilities on a regular basis. There are scanners that are capable of doing this. In fact, there is a good market for scanners that can identify vulnerabilities within your systems.

A vulnerability scanner works with a database of signatures, and this database needs to be updated regularly. Without a signature, the scanner won't be able to detect a vulnerability. Vulnerability scanners do not attempt exploits in order to verify that there is a vulnerability. They also do not do any brute force or fuzzing attacks in order to identify unknown vulnerabilities. That is the function of other tools. Vulnerability scanners should be safe to run against most systems, since the intention is not to cause any damage but instead to just identify vulnerabilities.

NOTE Although vulnerability scanners are not designed to cause damage to systems and their availability, sometimes fragile systems or services may crash or otherwise become unavailable. When you are running vulnerability scanners, be careful, and ensure you have notified system owners that you are running scans.

Vulnerability scanners run across the network and as such, they will generally identify network-based vulnerabilities. This is most useful on servers that are going to be more likely to have applications that are designed to listen on the network. Desktops will also have externally exposed services, though, so don't think that vulnerability scanners are not going to be valuable against desktops. Also, vulnerability scanners can be configured to connect to systems remotely using known credentials. This means that a vulnerability scanner can check for local vulnerabilities like those on programs that only exist on the local system and never have network connections. Adobe Flash and various PDF viewers have histories of local vulnerabilities, as does Java and a large number of other programs users make use of on a regular basis. A vulnerability scanner can connect to a system remotely and run the local checks against those. These local vulnerabilities can sometimes provide an attacker extra privileges than they might otherwise have, which is one reason they are valuable.

While a number of commercial scanners are available, there are also some that are free. If you happen to have a copy of Kali Linux, which is useful for a wide variety of network security and forensics tasks, you already have a vulnerability scanner. If it isn't installed by default, it is easy to add using the apt package manager. Once you install OpenVAS and get it configured using `openvas-setup`, you can connect to the web interface to start scans. One advantage to using OpenVAS, in addition to the fact that it is open source and free to obtain and use, is that there is a quick start capability. All you need are targets and OpenVAS will do some quick scanning of them without you having to do any additional configuration of your scan. You can see the starting page for OpenVAS in Figure 11-1. OpenVAS uses the Greenbone Security Assistant as its client interface and this is the Web implementation of that.

As mentioned, OpenVAS is open source. This means that anyone who is interested can contribute to it. This includes contributing to the codebase for OpenVAS itself but also signatures to go into the database for vulnerability checks. OpenVAS maintains a database of Network Vulnerability Tests (NVTs), which are written in a scripting language called the Nessus Attack Scripting Language (NASL). Nessus is another vulnerability scanner that also has a product that you can use at home to play around with. Nessus is developed by Tenable Security and they also have commercial offerings.

NOTE OpenVAS is a fork of a much earlier version of Nessus. Nessus was once open source but went closed source. The developers of Nessus were seeing others take what they had done, repack-age it, and sell it at a profit. The open source community had also not been contributing to Nessus, while reaping the benefits of Nessus being open source. As a result, the Nessus developers decided to close their source code, pursuing a more commercial path.

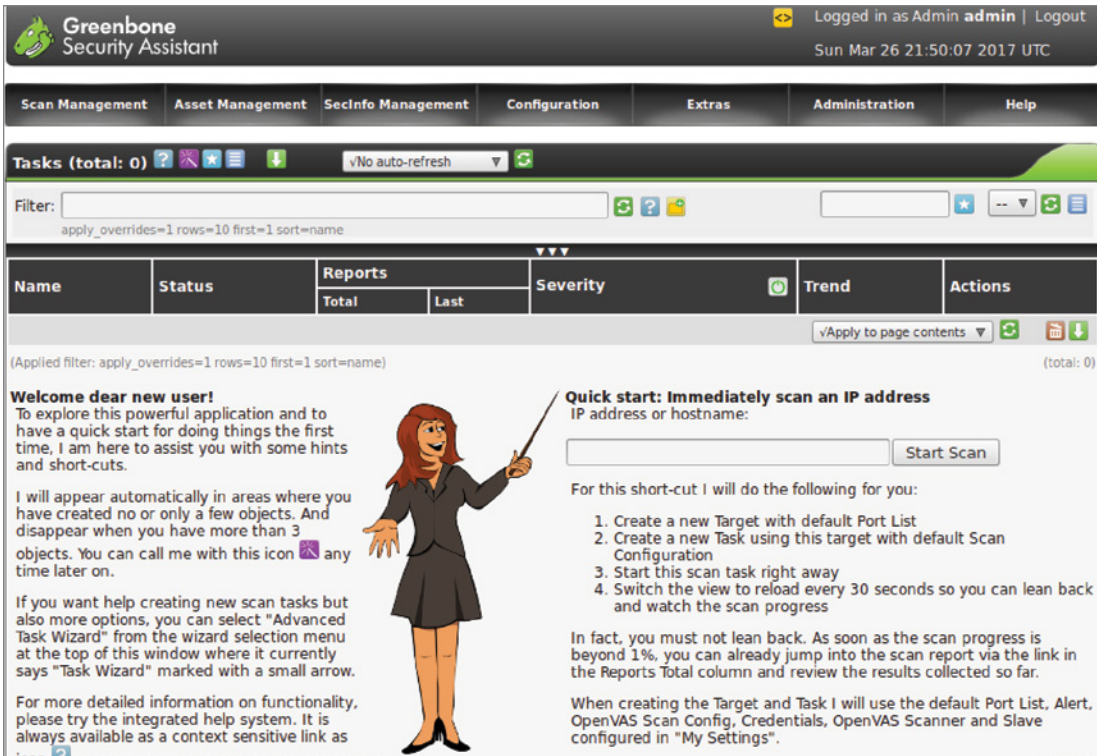


Figure 11-1: OpenVAS start page.

Using the OpenVAS quick start, you can get results back almost immediately. Scanning the entire network where I am resulted in a list within a very short period of time, which can be seen in Figure 11-2.

Results 1 - 10 of 89 (total: 106) vRefresh every 30 Sec.

Filter: severity>Error and task_id=a1da8950-8791-4385-bed6-0bd574a5aa sort=nvt first=1 rows=10

Vulnerability	Severity	QoD	Host	Location	Created
SSH Protocol Versions Supported	0.0 (Log)	95%	192.168.86.83	22/tcp	Sun Mar 26 21:58:45 2017
Dnsmasq Detection	0.0 (Log)	80%	192.168.86.1	53/tcp	Sun Mar 26 21:57:14 2017
Dnsmasq Detection	0.0 (Log)	80%	192.168.86.113	53/tcp	Sun Mar 26 22:00:01 2017
Determine which version of BIND name daemon is running	0.0 (Log)	80%	192.168.86.83	53/tcp	Sun Mar 26 21:58:28 2017
HTTP Server type and version	0.0 (Log)	80%	192.168.86.1	5000/tcp	Sun Mar 26 21:57:37 2017
HTTP Server type and version	0.0 (Log)	80%	192.168.86.83	80/tcp	Sun Mar 26 22:03:01 2017
HTTP Server type and version	0.0 (Log)	80%	192.168.86.49	1443/tcp	Sun Mar 26 22:05:17 2017
HTTP Server type and version	0.0 (Log)	80%	192.168.86.49	1400/tcp	Sun Mar 26 22:05:18 2017
HTTP Server type and version	0.0 (Log)	80%	192.168.86.45	1443/tcp	Sun Mar 26 22:05:24 2017
HTTP Server type and version	0.0 (Log)	80%	192.168.86.45	1400/tcp	Sun Mar 26 22:05:30 2017

(Applied filter: severity>Error and task_id=a1da8950-8791-4385-bed6-0bd574a5aa sort=nvt first=1 rows=10) 1 - 10 of 89 (total: 106)

Backend operation: 0.79s Greenbone Security Assistant (GSA) Copyright 2009-2016 by Greenbone Networks GmbH, www.greenbone.net

Figure 11-2: OpenVAS results.

Rapid 7, which develops the popular exploitation framework Metasploit, also has a vulnerability scanner called Nexpose. Nexpose is oriented toward an end-to-end vulnerability management. As a result, in order to start off a scan in Nexpose, you create an organization, a site, and a collection of assets. Once you have all of this in place, you can start a scan. You can see part of the creation of a site in Figure 11-3. Once you have the site created, you can start the scan. Nexpose, like OpenVAS, doesn't wait for the entire scan to be complete before starting to show results. As soon as there is anything to show, including identified assets, Nexpose will begin presenting results through the web interface.

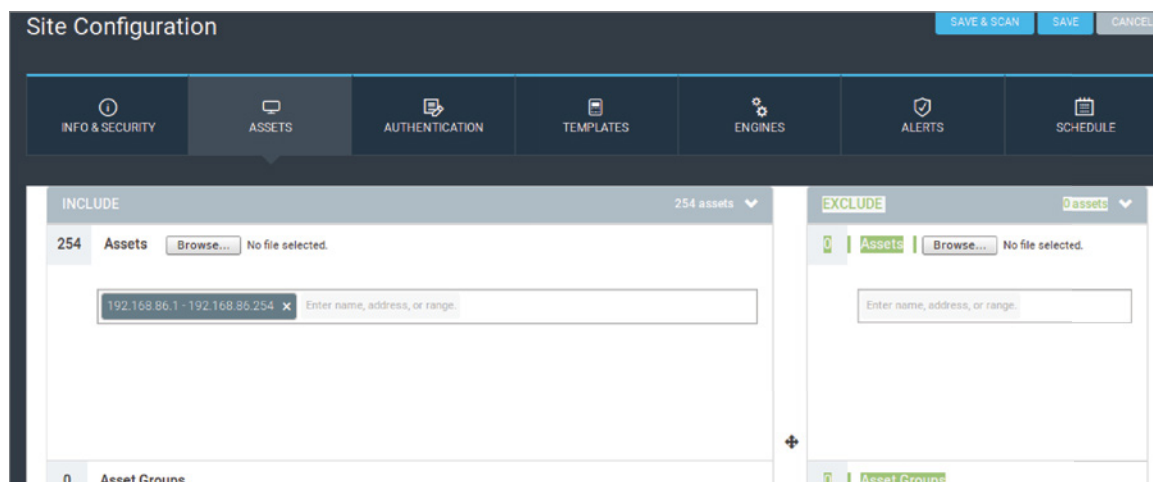
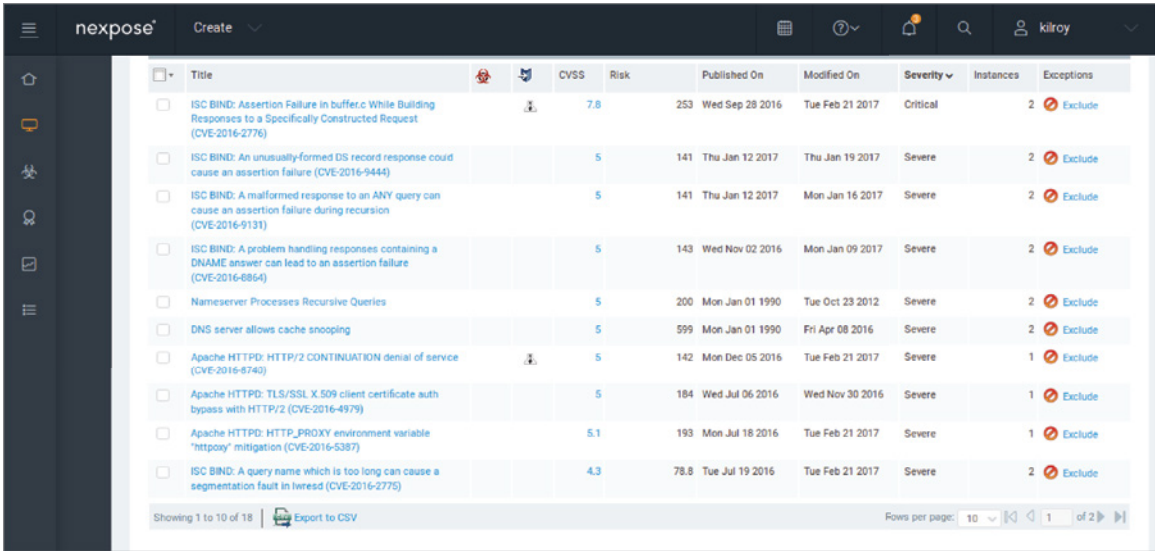


Figure 11-3: Nexpose site creation.

Once the scan has completed, you can see a list of vulnerabilities across the site. Additionally, Nexpose will calculate a risk score, based on the number and severity of vulnerabilities that have been identified. An Ubuntu Linux system on my network was found to have the vulnerabilities identified in Figure 11-4, and these vulnerabilities resulted in a total risk score of 2685 according to Nexpose. The number of the risk score will vary based on the severity of the vulnerabilities, and the number of them, so just looking at the risk score by itself isn't that valuable. You should look at each of the vulnerabilities found.

According to the output from Nexpose, two of the vulnerabilities identified had exploits that were available in the Exploit DB, which is a website used to store exploits and proof of concept code to determine whether vulnerabilities are real or not. You can determine whether there are exploits available in either Exploit DB or Metasploit by the little icons in each line. The existence of exploit code would increase the potential risk to the system because it would mean the hard work had been done already for an attacker. They would be able to use that exploit at least as a starting point for an attack on the system.



The screenshot shows the Nexpose web interface. At the top, there's a navigation bar with the 'nexpose' logo, a 'Create' dropdown, and icons for calendar, help, notifications, search, and user profile (kilooy). On the left is a sidebar with icons for home, reports, assets, alerts, and settings. The main content area displays a table of vulnerabilities. The table has columns for Title, CVSS, Risk, Published On, Modified On, Severity, Instances, and Exceptions. There are 10 rows of vulnerability data, each with a checkbox on the left and an 'Exclude' button on the right. At the bottom, it says 'Showing 1 to 10 of 18' and 'Export to CSV'.

	Title	CVSS	Risk	Published On	Modified On	Severity	Instances	Exceptions
☐	ISC BIND: Assertion Failure in buffer.c While Building Responses to a Specifically Constructed Request (CVE-2016-2776)	7.8	253	Wed Sep 28 2016	Tue Feb 21 2017	Critical	2	Exclude
☐	ISC BIND: An unusually-formed DS record response could cause an assertion failure (CVE-2016-9444)	5	141	Thu Jan 12 2017	Thu Jan 19 2017	Severe	2	Exclude
☐	ISC BIND: A malformed response to an ANY query can cause an assertion failure during recursion (CVE-2016-9131)	5	141	Thu Jan 12 2017	Mon Jan 16 2017	Severe	2	Exclude
☐	ISC BIND: A problem handling responses containing a DNAME answer can lead to an assertion failure (CVE-2016-6864)	5	143	Wed Nov 02 2016	Mon Jan 09 2017	Severe	2	Exclude
☐	Nameserver Processes Recursive Queries	5	200	Mon Jan 01 1990	Tue Oct 23 2012	Severe	2	Exclude
☐	DNS server allows cache snooping	5	599	Mon Jan 01 1990	Fri Apr 08 2016	Severe	2	Exclude
☐	Apache HTTPD: HTTP/2 CONTINUATION denial of service (CVE-2016-6740)	5	142	Mon Dec 05 2016	Tue Feb 21 2017	Severe	1	Exclude
☐	Apache HTTPD: TLS/SSL X.509 client certificate auth bypass with HTTP/2 (CVE-2016-4979)	5	184	Wed Jul 06 2016	Wed Nov 30 2016	Severe	1	Exclude
☐	Apache HTTPD: HTTP_PROXY environment variable "httpoxy" mitigation (CVE-2016-5387)	5.1	193	Mon Jul 18 2016	Tue Feb 21 2017	Severe	1	Exclude
☐	ISC BIND: A query name which is too long can cause a segmentation fault in Inresd (CVE-2016-2775)	4.3	78.8	Tue Jul 19 2016	Tue Feb 21 2017	Severe	2	Exclude

Showing 1 to 10 of 18 | [Export to CSV](#) Rows per page: 10 | 1 of 2

Figure 11-4: Nexpose site creation.

While Rapid 7 offers a community edition that can be used for free, it does have some limitations. One of those limitations is that you can only store a small number of IP addresses in the scanner. They do this to encourage people who are doing real scanning to purchase one of the commercial offerings. Someone who is scanning entire networks, particularly on a regular basis, would benefit from being able to store information about a larger number of hosts than those allowed in the community edition. You could certainly do your scanning in smaller-sized blocks, but that becomes difficult to manage because you have to keep deleting the results in the database before starting up the next block of addresses. Additionally, you don't get the historical information in Nexpose, though you can certainly export reports and store them for reference. That's a lot of work, though.

A significant advantage to using Nexpose is integration with Metasploit. If you are looking to determine whether a vulnerability is real and not just a false positive, Metasploit can be used to do the testing necessary to determine that. This is where you would actually exploit the vulnerability. If you were successful, you know for sure the vulnerability exists. You also know that if you can exploit the vulnerability using a readily available tool, attackers can also exploit the vulnerability. If you have identified easily exploitable vulnerabilities, it may be worth investigating to see if anyone else has similarly identified and made use of the vulnerability.

NOTE Like antivirus and other security-related products, vulnerability scanners are prone to false positives. A false positive is anything that is identified as legitimate when, in fact, it is not. In the case of a vulnerability scanner, a false positive would be an identified vulnerability when the

vulnerability doesn't exist on that system. More damaging than false positives are false negatives. A false negative is a vulnerability that exists but that has not been identified. False negatives can lead to a false sense of protection.

Along the same lines, if you have identified a system that doesn't have a vulnerability in a network full of essentially identical systems that do have the vulnerability, it may be worth exploring. Attackers will sometimes close holes they came in through in order to keep others out. It could be they have found one system and closed the hole without yet identifying and exploiting other systems on the network. In that case, it would just be a matter of time.

Port Knocking

Port knocking is a method of hiding ports from view. In order to gain access to the service you want, you need to go through what amounts to a network authentication. You would need to “knock” on some number of ports in the correct order before the real port would reveal itself to you. Harry Potter fans can think of Hagrid tapping bricks to get into Diagon Alley, as a physical example. For an actual example, you can look at the configuration file for knockd, which is a program designed to provide port knocking capabilities. The default configuration that comes with knockd on Kali Linux is shown in Listing 11-15.

Listing 11-15: knockd.conf Contents

```
[options]
    UseSyslog

[openSSH]
    sequence      = 7000,8000,9000
    seq_timeout   = 5
    command       = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22
-j ACCEPT
    tcpflags      = syn

[closeSSH]
    sequence      = 9000,8000,7000
    seq_timeout   = 5
    command       = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22
-j ACCEPT
    tcpflags      = syn
```

This particular configuration setting makes use of the firewall in Linux to only open a hole after a particular set of ports has received connection attempts. Once the appropriate “code” has been used, the firewall will be configured to grant access only to your IP address. Once you are done, you would knock in a different sequence to close the hole in the firewall so no one else would be able to

get through. In Listing 11-15, the sequence is ports 7000, 8000, and 9000, then in reverse to close the hole—9000, 8000, and 7000.

By default, you would initialize a connection attempt, which could be done using the telnet client or netcat. Should you wish to use something more complicated, a tool like hping3 can be used to send specially crafted messages to a target. You could change the `tcpflags` setting to something like `FIN`. This would require that instead of the SYN message that is used by default here, you would be expected to send a message with just the FIN flag set in the TCP header.

In the case of the configuration in Listing 11-5, knockd requires the use of a firewall to open and close ports for access and only grants access to the IP address that initiated the knocking. While we have covered port scanning in enough detail for you to be able to scan systems for open ports, be aware that you are still at the mercy of whatever is happening on the local system. This is especially true if there is a firewall running on the target. Similarly, a port scanner won't be able to determine if the target has a port knock daemon in place, listening for the right connection sequence. One of the points of a port knock daemon and hiding services behind it is to defeat port scanners. The daemon will hide the existence of the service you are looking to discover.

Tunneling

There are different ways to obscure data being sent over the wire. I have said that there is no way to hide what happens on the network. This is true from a network perspective and you will always be able to see what is going on at the network and transport layers. The same is not true higher up the stack than that, though. While you will always be able to see the communication happen if you are looking in the right place, you will not always be able to see inside that communication. We'll get into encryption later but for now, we are going to talk about another way of hiding information. This is through a technique called *tunneling*. In tunneling you send one type of data inside another protocol.

There are different tunneling protocols that are designed to carry data from one location to another. The Point to Point Tunneling Protocol (PPTP) is one of these. This used to be used regularly when the Internet was accessed by most people over dial-up connections. There are still locations that will use PPTP to get data from one location to another because it's necessary in cases where the endpoints may not otherwise be reachable to one another. It may also be the case that you don't want the packets being routed over the open Internet but instead carried to another location and have the routing done there. Another protocol that can be used for the same sort of purpose is the General Routing Protocol (GRE). Both of these protocols are designed to encapsulate data with a set of headers that are specific to the protocol doing the encapsulation, and those headers are used to transport the data to the endpoint specified in the encapsulation headers.

While protocols like PPTP and GRE were designed to encapsulate and transport data, other protocols can be used to carry data. When you see PPTP or GRE in a packet capture, you know you are seeing encapsulated data. The same isn't always true with other protocols. One protocol that can be used to carry data from one location to another and often be entirely unseen is the Hypertext

Transfer Protocol (HTTP). HTTP is a protocol used to transfer web pages and other related content like images. Where GRE and PPTP can be easily blocked by a firewall and readily identified as carrying tunneled traffic, HTTP is a different story altogether. The default ports used for HTTP and its associated protocol HTTP Secure are commonly allowed right through firewalls.

A number of programs can be used to tunnel any type of traffic over HTTP. One way to do it is to use the `CONNECT` method specified in HTTP. This method tells an HTTP proxy server to initiate a connection to a remote system using a specified protocol and port. The proxy server takes care of the connection and the client can then send data through the proxy server to the endpoint specified in the `CONNECT` request. This does not mean that all HTTP proxies can be used to perform this tunneling, though. Not all proxies support the `CONNECT` method and even if they do, they may limit the types of protocols and ports that they will connect to in order to prevent misuse.

There are ways to do HTTP tunneling without using `CONNECT` or an HTTP proxy. In these cases, though, you would need a special server as well as a special client. The two would be paired to transmit your information between them. The client gets the information off your system, and the server that the client connects to would then get your real network traffic out into the rest of the world. This is something you can obtain as a service. There are businesses that offer access to their farms of servers and the client that connects to them.

Other protocols can be used for tunneling. Secure Shell (SSH) will open connections to remote systems that would normally be used for remote, terminal logins but once the encrypted session is open, users can send data down the tunnel. This effectively turns an SSH session into a tunnel for any type of traffic. In fact, you can specifically turn an SSH session into a type of proxy for Secure Sockets (SOCKS).

In addition to using SSH and the clients that are available for it, there is another program, called `stunnel`, that will do encrypted transmission of data. Once you have the encryption in place, the entire stream looks like a tunnel because from the outside, you can't see what's on the inside. `stunnel` doesn't care what type of data it carries so you can send anything you want down one of these tunnels.

Passive Data Gathering

While you may want to actively gather data using techniques like port scans, it is also possible to passively gather data about hosts. You won't find the same type of information since you can't passively identify all ports that are open on a system, but you can gather a fair amount of information regarding remote systems by just sitting and listening to traffic that is going by. Because you are passively gathering data, it is only about systems that are communicating with you—whether you have initiated the communication or they have. The program `p0f` will listen to packets coming across the wire and extract relevant information from them. You could also do this using `tcpdump` and/or Wireshark. However, pulling the data out of the packet and putting it into a coherent form can be very time consuming. Having a tool to do it for you automatically can be very convenient. You can see some sample output from `p0f` in Listing 11-16.

Listing 11-16: p0f Output

```

.-[ 192.168.86.65/52542 -> 64.233.169.100/443 (syn) ]-
|
| client    = 192.168.86.65/52542
| os        = Mac OS X
| dist      = 0
| params    = generic
| raw_sig   = 4:64+0:0:1460:65535,5:mss,nop,ws,nop,nop,ts,sok,eol+1:df,
id+:0
|
|-----

.-[ 192.168.86.65/52542 -> 64.233.169.100/443 (mtu) ]-
|
| client    = 192.168.86.65/52542
| link      = Ethernet or modem
| raw_mtu   = 1500
|
|-----

.-[ 192.168.86.65/52542 -> 64.233.169.100/443 (uptime) ]-
|
| client    = 192.168.86.65/52542
| uptime    = 15 days 18 hrs 44 min (modulo 49 days)
| raw_freq  = 994.08 Hz
|
|-----

.-[ 192.168.86.65/52542 -> 64.233.169.100/443 (syn+ack) ]-
|
| server    = 64.233.169.100/443
| os        = ???
| dist      = 20
| params    = tos:0x08
| raw_sig   = 4:44+20:0:1380:mss*31,7:mss,sok,ts,nop,ws::0
|
|-----

.-[ 192.168.86.65/52542 -> 64.233.169.100/443 (mtu) ]-
|
| server    = 64.233.169.100/443
| link      = generic tunnel or VPN
| raw_mtu   = 1420
|
|-----

```

This may look like a packet capture in some regards because in essence, that's what it is. P0f has captured the packets and extracted relevant information. It also puts it into context. As an example,

the very first block in the listing indicates which end of the communication stream is the client. P0f was also able to identify the operating system of the client end. It was able to do this by paying attention to the messages that were going back and forth. This is just a sample of what p0f can do, however. In order to get a full taste of what sorts of information you can extract from network communications using passive data gathering, you should get a copy of the software yourself and run it for a while. Using p0f, you will get a new appreciation for the types of information that can be available to you by just listening to what is going on around you.

Summary

You can learn a lot about what is going on by looking at the host and gathering information there. You can also watch network traffic. However, sometimes, you need that outside perspective. Just watching traffic pass across the wire doesn't tell you much about what the host may be doing from a network perspective if all it is doing is sitting there listening. There may be connections coming in but they may be very sporadic, which may mean you miss them. It's best to get an idea from the outside whether there are ports that are listening. This includes knowing what is listening, and not just knowing whether there is anything listening. There are different ways of accomplishing that goal.

While there are a number of port scanners around, the de facto scanner is nmap. Even other scanners, like masscan take the command-line parameters that nmap has been using for the last two decades. Since people are so familiar with nmap, it makes sense to function more or less like nmap does. At a minimum, even if the back-end works differently, getting the interface to behave similarly means that people who have been using nmap have no learning curve coming to a new tool.

Nmap will do scans of UDP and TCP, looking for open ports. In the case of TCP, it can use different scanning techniques because TCP has flags that guide the behavior of the systems that are communicating. This means nmap will set different TCP flags in order to determine the behavior of the target host, which will help it determine whether the port is open. In the case of UDP, it's much easier, though UDP scans can take longer because there is nothing that guarantees the behavior of a UDP system. This means that retransmissions may be required to ensure the message gets to the target. A lack of response isn't a guarantee of anything when it comes to UDP systems.

Nmap also supports scripting so you can engage with services in a way that you choose. You get several hundred scripts when you install nmap, and using these scripts, you can gather a lot of information about your target and the services that are running on that target. If you are looking for more details than what you get from the scripts nmap provides, you can write your own scripts since they are written in a well-defined programming language, Lua.

One thing you may want to do is get details from the service banners from the applications listening on the open ports. While nmap can do this, you can also use tools like netcat and the telnet client to engage directly with the port. Netcat supports both TCP and UDP connections but telnet is limited to just TCP. It does require that you know something about the protocol being used by the application you are engaging with.

Attackers can sometimes get into systems by exploiting vulnerabilities. Vulnerability scanners are available that can probe systems to determine known vulnerabilities. Vulnerability scanners can only get information about known vulnerabilities. They do not attempt to exploit systems, nor do they attempt to identify vulnerabilities that are not previously known. They don't programmatically probe applications to identify previously unknown vulnerabilities. This means that the scanner uses a database of signatures that need to be updated on a regular basis.

While port scanning is valuable, there are other techniques that attackers may use that are useful to know about. One of them is tunneling. Using tunneling, an attacker can hide attack traffic inside something that may appear to be innocuous. This can be done using protocols like GRE or PPTP or more common protocols like HTTP, which means it can be harder to identify when what you are looking for is buried inside something that is very commonplace.

12

Final Considerations

In this chapter, you will learn about:

- Encryption and its ramifications
- Cloud computing implications
- What The Onion Router (TOR) and the so-called “dark web” is

While looking through the Internet history on the latest system he was able to compromise, he discovered accounts with Dropbox, Google, and Amazon. The accounts led him to a fair amount of data stored with these providers. Additionally, he was able to retrieve the cached passwords for the accounts, which meant he was able to directly access the accounts from other systems. The Amazon account led to a system that was running, presumably for business purposes, and that was another account that he might be able to use. Amazing that he was able to turn up so many accounts from this one system with data stored off-site. These were just the sorts of finds that kept him going. The best part about using these sites is that they were all encrypted, which meant it wasn't easy to determine what was being sent without direct access to the endpoint.

One area that I have deliberately steered away from until now, though it has tremendous impact on network traffic capture and analysis, is encryption. One reason I've stayed away from discussing it is that if you run across encrypted network traffic, it can be difficult, if not impossible, to get at what is inside the messages. At best, you may be left with the metadata that you get out of the conversation from the headers. You can get the IP information and the TCP information but none of the application information, which may be the most important data you are looking for. Because of the challenge associated with encryption, it's important to understand it, its capabilities, and its limitations. Encryption has become a fact of life, not only from the standpoint of host analysis with whole disk encryption, but also because as more and more sensitive data is stored with service providers, network encryption is far more prevalent.

Cloud computing, despite its ambiguous name, has become another fact of life. More businesses are moving to service providers to not only take care of data storage but also handle applications.

The workforce is increasingly mobile and geographically dispersed, which means workers need to be able to access business data from wherever they are. Service providers can make this possible, by allowing businesses to outsource a lot of the hard and expensive stuff while empowering their workers. This creates issues when it comes to forensic investigations because it requires working with the service provider to acquire data rather than acquiring it directly. Unlike a local business where you can go in, seize the computer, and then get an image of a drive, or where you could install a network sensor to capture packets, service providers have a far more complex setup and also won't provide you with permission to just monitor their network.

If you watch TV or movies, especially the so-called police procedurals, you will likely have heard of the dark net or the dark web. These terms are commonly and colloquially used to refer to sites that are not reachable by way just firing up a web browser like Chrome, Edge, Internet Explorer, Safari, or Firefox and heading out to the Internet in the way you are used to. Instead, you would use a TOR browser to gain access to these hidden sites or reconfigure your current browser to use the TOR network. You can also browse regular websites without being identified because of the way traffic is passed around to get to its destination. This is another way that life of a forensic investigator can be challenged over the course of a network investigation. Understanding how this works will help you to know where you may be able to look next.

Encryption

Almost as long as there has been sensitive data in need of protection, people have been finding ways to hide that information. They do this by taking information and altering it such that someone would need to understand how it was altered in order to read the original message. The process of altering a message using a piece of information that should only be known by the sender and recipient, rendering the original information unreadable is called *encryption*. Restoring encrypted information to the original message, commonly called *plaintext*, is called *decryption*. Once a message is encrypted, it is called *ciphertext*. We normally think of encryption as something that requires a computer, but in fact, we were taking plaintext and creating ciphertext out of it centuries before we had computers.

One method of encryption that doesn't require a computer is sometimes referred to as the Caesar cipher, and sometimes it's just called a rotation cipher. The way it works is you write out the alphabet and then write it out a second time but shifted some number of letters. You can see this in the following example. To encrypt a message, you find the letter in the plaintext in the top alphabet, then locate the letter directly below it and that becomes your ciphertext. You repeat the process for all letters in your message until you have a completely encoded message.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ  
DEFGHIJKLMNOPQRSTUVWXYZABC
```

The word "hello," for example, becomes "lipps." To restore the original word, find the letters in the ciphertext in the lower alphabet and replace them with the letters directly above them in the upper

alphabet. Of course, you don't have to write your alphabet out in order to perform this encryption and decryption, but it does help to be able to see what is going on and makes wrapping around the end of the alphabet a little easier. All we are doing is rotating the letters by 3. If you see an A, you skip ahead three letters to get D and that's your ciphertext letter. You reverse the process by skipping back 3 letters.

NOTE One way of protecting sensitive readers, avoiding spoilers, or sometimes just being humorous on the Internet and its precursor networks was using a weak form of encryption called rot13. This is a form of a Caesar cipher where you rotate the alphabet by 13 letters. Since it's easy to reverse, nothing is really well-protected but it required a step to be able to read it.

Encryption is a complicated business, and not least of all because of the mathematics involved in doing modern encryption. Protecting information is not as simple as converting it to a ciphertext; if that were the case, we'd still be using a rotation cipher. The problem is that as fast as people can come up with schemes to encrypt information, other people are working just as fast to figure out how to decrypt that information. As a result, a lot goes into encryption and we're going to talk about the different ways encryption happens and how the information gets to be protected.

Keys

When it comes to encryption, the key has always been the key. Even if we are talking about a simple rotation cipher, you still need to know how many characters to rotate before you can start to do the decryption. Using a simple rotation cipher, you should be able to do something like a frequency analysis if you have enough text to look at. A frequency analysis will let you determine the most used characters in the message. This will let you map what you have found to a normal frequency distribution of letters. Once you have the frequency of letter occurrences, you can start the process of reversing the encryption. The most frequently used letter in the English language is e. After that, it depends on whose analysis you want to believe. It could be t or a. According to Robert Lewand, who did a frequency analysis for his book *Cryptological Mathematics* (The Mathematical Association of America, 2000), the letter e is used more than 12% of the time. This is followed by the letter a, which is used slightly more than 8% of the time.

Once you have determined the letter mapping, you can get to what is referred to as the key. The key is the piece of information that is needed to decipher a piece of encrypted text. In the case of the rotation cipher, the key could be thought of as 3. This means that the alphabet is rotated three positions. In modern encryption, the key is quite a bit more complex and larger, because modern encryption makes use of complex math and the key is used to feed the equations that do the encryption and decryption.

Based on this simple example, though, you can see how important it is to protect the key. Once the key is known, no matter how complex or how simple it is, the encrypted data can be decrypted. Protecting the key is one of the biggest challenges in modern cryptography. If you and I were to try

to communicate sensitive information, we would need a key. How do we come up with a key that both of us know so we can encrypt and decrypt the information? Do I call you and tell you what the key is? No, because someone could overhear it—all they need is the ciphertext and they can convert it to plaintext.

The phone mechanism for transmission is entirely simplistic, especially considering keys are very long and often not even readable, which means they need to be converted to something that is readable. This is typically hexadecimal, which can make the key more manageable, but if there is any mistake at all in telling you what the key is, you are not going to be able to decrypt what I send you.

Fortunately, a number of researchers and cryptographers came up, roughly at the same time, with a way of sharing keys that protects the key. The method that is most well-known—in part because some of the other teams were working under government intelligence secrecy and weren't allowed to publish like Whitfield Diffie and Martin Hellman—is the *Diffie-Hellman key exchange protocol*. Diffie and Hellman came up with a way of allowing both sides to independently derive the key that would be used. They do this by starting at a common point and sharing a piece of information that is fed into an algorithm with another piece of information they create themselves.

Let's say, by way of an entirely unrealistic example, that we wanted to create an animal that could carry our information. We both agree to start with a tiger. You choose to cross your tiger with a wolf, creating a wolf-tiger. I choose to cross my tiger with an elephant, creating an elephant-tiger. I send you my elephant-tiger and you send me your wolf-tiger. As soon as I add in my elephant to your wolf-tiger, I get a wolf-tiger-elephant. You take my elephant-tiger and add in your wolf and you also get a wolf-tiger-elephant.

Imagine a mathematical process that equates to this really bad genetic process and you have a sense of how Diffie-Hellman works at creating a key that we can both use, and that has not been transmitted. The assumption is that taking the wolf-tiger and determining that both a wolf and a tiger went into it is a very expensive process. If someone could extract the wolf-tiger into its component pieces and the elephant-tiger into its component pieces, that someone would know what went into creating the key, and since the mathematical process to create the key is well-known, that person could create the key that you and I are using to encrypt information. Because you have the key, you can decrypt the data.

Now, hold onto the idea of keys and the knowledge that keys are used to encrypt and decrypt data. We are going to talk about different ways that those keys can be used.

Symmetric

The process where you and I are using the same key to encrypt and decrypt information is called *symmetric encryption*. The same key is used on both ends of the communication. If you have done any exchanges on the Internet that have been encrypted, you are using symmetric encryption, whether you realize it or not. You may even be familiar with the names of the encryption ciphers. DES, 3DES, and AES are all encryption ciphers that use symmetric encryption.

NOTE You will see DES and AES referred to but in fact, those are not the names of the encryption algorithms. Those are the names of the standards. DES is the Data Encryption Standard and AES is the Advanced Encryption Standard. The actual name of the algorithm DES is based on is LUCIFER, an encryption cipher developed by IBM in the early 1970s. AES is the result of a contest that was designed to find a strong successor to DES. The algorithm that was chosen was named Rijndael, developed by a team of researchers.

Symmetric key encryption tends to be faster than asymmetric and takes less computing power. Additionally, key sizes tend to be smaller. Don't be fooled by this, though. You can't compare key lengths and assume that a cipher with a larger key is necessarily stronger. The strength of an encryption process is related to the algorithm and the algorithm determines the size of the key based on how it works. As an example, AES is a block cipher. This means that it takes in chunks of data in 128-bit blocks. That's 16 ASCII characters, to give you a sense. Your data is chunked into blocks of 128 bits and sent into the algorithm. If what you send isn't a multiple of 128 bits, your data will be padded out to make sure that it's a multiple so the block size doesn't have to be compromised or adjusted.

Using symmetric key encryption, one algorithm is used to encrypt while another is used to decrypt and both sides use the same key, which means that protection of the key is important, as indicated previously. Because both sides need to use the same key, there needs to be a way of either uniquely deriving the same key on both sides, or a way to get the key from one side to the other. One of the challenges of symmetric key algorithms is that after prolonged use, the key may be prone to attack because an attacker can acquire a large amount of ciphertext that he can analyze to attempt to derive the key that can be used for decryption. The solution to this is to rekey periodically, meaning replace the key you were using with a different key. However, that does bring up the issue of how do both sides know when to rekey and how do they both get the new key?

Asymmetric

Unlike symmetric encryption, asymmetric uses two separate keys. One of the keys is used to encrypt while the other is used to decrypt. A common implementation of asymmetric encryption is sometimes called *public key encryption*, because the two keys are referred to as the public key and the private key. Not surprisingly, the *public key* is available for public use while the *private key* is the one you keep protected. This is by design. The more people who have your public key, the more people who can send encrypted messages to you. If I don't have your public key, I can't encrypt a message. On the other side of that coin, even if I have your public key, I can't decrypt any message that has been encrypted with your public key. The public key is only good for encrypting messages because the private key is required to decrypt them.

In the case of asymmetric keys, the two keys, which are just numbers after all, are related mathematically. You can think of them as two halves of a whole because they are linked. The way asymmetric algorithms work is to take advantage of the mathematical relationship between the two numbers.

Without getting too deep into the math, unless you feel like doing some research, it's sufficient for our purposes here to know that the two keys are related and the algorithms make use of the two keys to encrypt and decrypt messages.

The whole system works because you have a public key or you have the ability to obtain a public key. There are two primary systems for key management. One is centralized and the other is decentralized. The centralized approach uses a certificate authority to generate the keys and validate that the owner of the keys is who that entity (it could be a system or a person) claims to be. The keys are stored in a data object called a *certificate* and use of the keys is typically protected by password because, unlike symmetric keys, these keys will stick around for a long time. While you do want your public key getting out, you do *not* want your private key to be accessed or used by anyone but you. As a result, since the key is just a chunk of data that is stored in a file, you password protect access to the key so that anyone wanting to use your private key needs to know the password.

The second approach is decentralized. Pretty Good Privacy (PGP) and Gnu Privacy Guard (GPG) both use this approach. Unlike the certificate authority, PGP and GPG rely on the users to validate one another's identities. Public keys are stored on key servers and a key server can be anywhere and available to anyone since the idea is to make public keys accessible. Until you are able to get my public key, you are unable to encrypt a message to me.

NOTE Enterprises may choose to store keys or certificates on behalf of their employees. As an example, Microsoft's Exchange Server can store a certificate in its Global Address Book (GAL). If you have a certificate associated with your e-mail address in the GAL, anyone can encrypt messages to you.

This all sounds awesome, right? Asymmetric keys tend to be considerably longer than symmetric keys, though that's not always the case. However, they are also processor-intensive and the algorithms used to encrypt and decrypt tend to be slow. As a result, they are not great for real-time communication or even near-real-time communication. Asymmetric encryption is good for e-mail where it doesn't much matter how fast or slow it is, because we aren't as worried about additional milliseconds for encryption and decryption as we would be for data that was constantly flowing.

We now have two different algorithms. One is slow and not great for real-time communication, while the other is fast but suffers from the need to rekey to protect itself. What if we were able to use them together?

Hybrid

What you will commonly see in communication systems that require encryption is a hybrid approach. Using a hybrid approach, we use symmetric encryption for the actual communication but we use asymmetric encryption to send the symmetric key. This allows us to use the faster communication of a symmetric cipher while also providing the protection that the asymmetric encryption offers. It also

solves the problem of key sharing because with asymmetric encryption, I am entirely okay with you getting access to my public key. You and everyone else can have it. We don't need to protect it at all like we do with the symmetric key. As a result, we can use that encryption to send symmetric keys back and forth and they should be well-protected, assuming the private keys of both parties are protected.

When it comes to real-time communications, or even near-real-time, both parties will generate a symmetric key and then send that key to the other side. This symmetric key is called the *session key* because it is used to encrypt and decrypt messages during the course of the communication session between the two parties. Since the messages using the symmetric key are still flowing over open networks, which is why we're encrypting to begin with, we still have the issue of rekeying. When the session is established, a rekeying timer may be set. At some point, when the timer triggers, the session keys will be regenerated and shared using the asymmetric encryption. This maintains the integrity of the communication stream.

Web-based communications, such as those with Amazon or any other website that uses encryption to protect your data, will use a hybrid cryptosystem. This means that servers you communicate with will have certificates associated with them.

SSL/TLS

To support the increasing desire for businesses to engage with consumers over the World Wide Web in the mid-1990s, Netscape, the company, developed the Secure Sockets Layer (SSL). The initial version wasn't considered ready to release so the first version that was available was version 2.0, released in 1995. By 1996, version 3.0 was out because of the security issues with version 2.0. Both versions of SSL have since become deprecated or even prohibited because of issues with the security they provide to the connection. As a result, the current standard for offering encryption between a client and a web server is Transport Layer Security (TLS). TLS was developed in 1999 and while it doesn't vary greatly from SSL 3.0, the changes were significant enough that the two mechanisms were not considered compatible.

SSL/TLS provides a way for the two ends of the conversation to first agree on encryption mechanisms, then decide on keys, and finally send encrypted communication back and forth. Since the different versions of SSL are considered unusable due to issues in the way they work, we'll talk about TLS and how it functions.

To determine how the two ends are going to communicate, they need to agree on several things, not least of which are the keys they will use. To do this, they need to perform a handshake. This is separate from the three-way handshake that happens with TCP. Since TLS runs over TCP, the three-way handshake has to happen first. The first stage of the TLS handshake happens as soon as the three-way TCP handshake has completed. It starts with a ClientHello, which includes the TLS versions supported as well as the cipher suites that it can support. You can see a list of cipher suites supported by www.microsoft.com in Listing 12-1, which shows the variety that are available, as determined by the program SSLScan.

NOTE A *cipher suite* is a combination of encryption protocols along with hashing algorithms to verify the integrity of the communication. It also provides the key exchange mechanism that will be used.

Listing 12-1: SSLScan Output Showing Ciphersuites

```
Supported Server Cipher(s):
Preferred TLSv1.2 256 bits ECDHE-RSA-AES256-GCM-SHA384 Curve P-256
DHE 256
Accepted TLSv1.2 128 bits ECDHE-RSA-AES128-GCM-SHA256 Curve P-256
DHE 256
Accepted TLSv1.2 256 bits ECDHE-RSA-AES256-SHA384 Curve P-256
DHE 256
Accepted TLSv1.2 128 bits ECDHE-RSA-AES128-SHA256 Curve P-256
DHE 256
Accepted TLSv1.2 256 bits ECDHE-RSA-AES256-SHA Curve P-256
DHE 256
Accepted TLSv1.2 128 bits ECDHE-RSA-AES128-SHA Curve P-256
DHE 256
Accepted TLSv1.2 256 bits AES256-GCM-SHA384
Accepted TLSv1.2 128 bits AES128-GCM-SHA256
Accepted TLSv1.2 256 bits AES256-SHA256
Accepted TLSv1.2 128 bits AES128-SHA256
Accepted TLSv1.2 256 bits AES256-SHA
Accepted TLSv1.2 128 bits AES128-SHA
Preferred TLSv1.1 256 bits ECDHE-RSA-AES256-SHA Curve P-256
DHE 256
Accepted TLSv1.1 128 bits ECDHE-RSA-AES128-SHA Curve P-256
DHE 256
Accepted TLSv1.1 256 bits AES256-SHA
Accepted TLSv1.1 128 bits AES128-SHA
Preferred TLSv1.0 256 bits ECDHE-RSA-AES256-SHA Curve P-256
DHE 256
Accepted TLSv1.0 128 bits ECDHE-RSA-AES128-SHA Curve P-256
DHE 256
Accepted TLSv1.0 256 bits AES256-SHA
Accepted TLSv1.0 128 bits AES128-SHA
```

SSL Certificate:

Signature Algorithm: sha256WithRSAEncryption
RSA Key Strength: 2048

Subject: www.microsoft.com

Altnames: DNS:privacy.microsoft.com, DNS:c.s-microsoft.com,

```

DNS:microsoft.com, DNS:i.s-microsoft.com, DNS:www.microsoft.com,
DNS:wwwqa.microsoft.com
Issuer: Symantec Class 3 Secure Server CA - G4

Not valid before: Oct 26 00:00:00 2016 GMT
Not valid after: Oct 27 23:59:59 2018 GMT

```

The very first line, which is one of the preferred cipher suites, indicates that the key exchange would happen with the elliptic curve Diffie-Hellman key exchange, using RSA as the asymmetric key algorithm. They are also specifying AES for the symmetric key algorithm with a 256-bit key. With AES, the server is offering to use Galois/Counter Mode (GCM) as the form of block cipher. Finally, the server is offering Secure Hash Algorithm for data integrity checks using a 384-bit hash result. When the client sends its ClientHello to the server, it would send a list of cipher suites similar to what you see in Listing 12-1. The list may not be nearly as long as that one but it would present choices to the server.

The server would follow up by selecting the cipher suite it preferred from the list provided by the client. It would send its selection back to the client in a ServerHello message. In addition to the selection of the cipher suite, the server would send its certificate. The certificate would include its public key that could be used to encrypt messages to send back to the server. The server would have to have a certificate. Without a certificate, it couldn't be configured to use TLS. The same would have been true with SSL. The challenge at this point is that the client can encrypt messages to the server but the server can't encrypt messages to the client. The server could ask the client if it had a certificate, in which case, the server could use the public key from the certificate to encrypt the message back to the client. Barring the certificate, they would have to use Diffie-Hellman to derive the symmetric key that will be used going forward.

Once the client has received the preferred cipher suite from the server, it either chooses to agree with it or it suggests another cipher suite. This would not be typical. Instead, it would normally begin the key exchange process, sending a message to the server indicating that. Both sides would go through the Diffie-Hellman exchange, ending up with a key at the end that could be used during the communication session. In Figure 12-1, you can see a packet capture in Wireshark demonstrating the handshake taking place between the browser on my system and a web server on the other end.

Partway down the packet capture is a message (frame number 9603) that is labeled ChangeCipherSpec. This is a message used to indicate to the other side that the negotiated cipher suite and keys will be used going forward. In short, it says this is the last message you will receive before receiving nothing but encrypted messages. You will notice that both sides send the ChangeCipherSpec message in frames 9603 and 9610. These messages indicate that all subsequent messages will be encrypted. After that, everything you see captured will be Application Data or just fragments of packets. You'll get nothing in the Info column from Wireshark other than TCP-related data because that's the last thing it can see in the packet. Everything beyond the TCP header is encrypted.

No.	Time	Source	Destination	Protocol	Length	Info
9593	40.469231	e673.e9.akamaiedge.net	oliver.lan	TCP	66	https → 57313 [ACK] Seq=14466718 Ack=372033...
9594	40.469241	e673.e9.akamaiedge.net	oliver.lan	TLSv1.2	1514	Server Hello
9595	40.469245	e673.e9.akamaiedge.net	oliver.lan	TCP	1514	[TCP segment of a reassembled PDU]
9596	40.469502	oliver.lan	e673.e9.akamaiedge.net	TCP	66	57313 → https [ACK] Seq=3720339497 Ack=1446...
9597	40.470782	e673.e9.akamaiedge.net	oliver.lan	TCP	1266	[TCP segment of a reassembled PDU]
9598	40.470785	e673.e9.akamaiedge.net	oliver.lan	TCP	1514	[TCP segment of a reassembled PDU]
9599	40.470836	oliver.lan	e673.e9.akamaiedge.net	TCP	66	57313 → https [ACK] Seq=3720339497 Ack=1447...
9600	40.512939	e673.e9.akamaiedge.net	oliver.lan	TLSv1.2	1283	Certificate
9601	40.513013	oliver.lan	e673.e9.akamaiedge.net	TCP	66	57313 → https [ACK] Seq=3720339497 Ack=1447...
9602	40.524379	oliver.lan	e673.e9.akamaiedge.net	TLSv1.2	141	Client Key Exchange
9603	40.524441	oliver.lan	e673.e9.akamaiedge.net	TLSv1.2	72	Change Cipher Spec
9604	40.524441	oliver.lan	e673.e9.akamaiedge.net	TLSv1.2	111	Encrypted Handshake Message
9605	40.530595	oliver.lan	testwifi.here	DNS	87	Standard query 0xb04e PTR 116.86.168.192.in...
9606	40.530703	oliver.lan	testwifi.here	DNS	83	Standard query 0xb04e PTR 1.74.13.31.in-add...
9607	40.532490	testwifi.here	oliver.lan	DNS	113	Standard query response 0xb04e PTR 116.86.1...
9608	40.546795	testwifi.here	oliver.lan	DNS	131	Standard query response 0xb04e PTR 1.74.13...
9609	40.565347	e673.e9.akamaiedge.net	oliver.lan	TCP	66	https → 57313 [ACK] Seq=14473479 Ack=372033...
9610	40.565350	e673.e9.akamaiedge.net	oliver.lan	TLSv1.2	117	Change Cipher Spec, Encrypted Handshake Mes...
Random						
Session ID Length: 32						
Session ID: b0933803a20e90490959726c54b6e7423f0dead720e1c34...						
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)						
Compression Method: null (0)						
Extensions Length: 13						
▶ Extension: renegotiation_info						
▶ Extension: ec_point_formats						
0000	f4 5c 89 b7 2c 09 18 d6	c7 7d cb 53 08 00 45 20	.\.....).S..E			
0010	05 dc a1 a0 40 00 36 06	fb d7 17 c1 72 d9 c0 a8	@.6....f...			
0020	56 41 01 bf db e1 00 dc	be 9e dd bf e0 29 80 10	VA.....)...			
0030	03 ab 59 a2 00 00 01 01	08 0a 26 39 1f 02 05 b3	..Y......69...			
0040	63 50 16 03 03 00 59 02	00 00 55 03 03 4e 68 19	cP.....Y...U.Nh.			
0050	b5 d6 11 7f 15 36 64 a8	3a 89 b1 e7 a6 e9 af da	6d.:.....			
0060	a8 dd ac 8a d6 01 d4 f9	bc 60 d5 02 94 20 b0 93	I.V...LT..B7.			
0070	38 03 a2 8e 90 49 09 59	72 6c 54 b6 e7 42 3f 90	8.....I.V...LT..B7.			
0080	de ad 72 0e 1c 34 d1 39	87 fb fb 23 33 a9 c0 30	..f..4.9...#3..0			

Figure 12-1: TLS handshake in Wireshark.

Certificates

We’ve been talking about certificates without providing a lot of information about them. The *certificate* is a data structure that has been defined by the X.509 standard. X.509 falls underneath the X.500 directory protocol suite. X.500 is a way of structuring information about organizations and individuals within the organization. A simplified implementation of X.500 is the Lightweight Directory Access Protocol (LDAP), which is how Microsoft accesses information stored for its domains including resources like users, printers and even encryption keys. Other companies also use LDAP. As part of the directory, each entry could have encryption information. The certificate is organized with fields providing information about the owner of the certificate; a certificate could have the Organizational Unit or Common Name fields, for example.

Additionally, the certificate will include information about its issuance: the organization that issued the certificate, the date it was issued and the date it expires, the algorithm that was used to sign the certificate, and a number of other fields that not only describe the certificate but also provide important information to judge its validity.

If you are using a web browser, you can look at certificates that have been provided by the web server you have connected to over HTTPS. Figure 12-2 is an example of a certificate, as shown in Google Chrome. Different browsers will present the information in different ways, but ultimately, what you see is all included in the certificate.

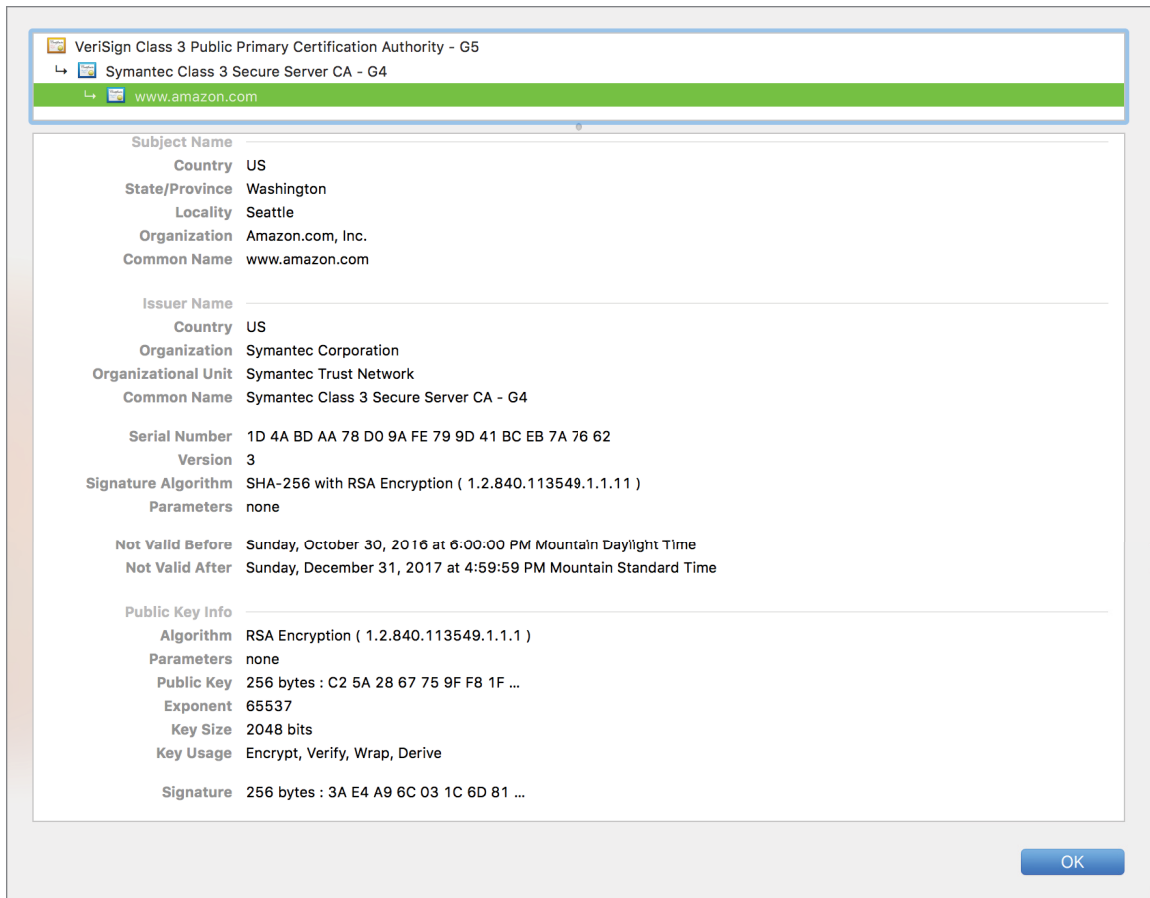


Figure 12-2: Amazon's web server certificate.

In Figure 12-2, you can see not only the name of the website, `www.amazon.com`, but also the organization that provided the certificate to Amazon, Symantec. Toward the bottom, you can see the algorithm that was used to sign the certificate. Signing the certificate involves a known and trusted entity, in this case Symantec, indicating that the certificate is valid and the entity that has the certificate, Amazon's web server, is known to Symantec. If you trust Symantec to do its job correctly and well, and Symantec trusts Amazon and that the web server named `www.amazon.com` is the correct website, then by the transitive property, you trust Amazon's website.

Certificates work by not only providing a means for encryption to happen but also by providing a way for users to trust the sites they are going to. If a known and trusted authority like the Symantec certificate authority has issued a certificate, it must be to the right place, as indicated in the certificate. If all that is true, you can trust the site and not worry that data sent to it will get into the wrong hands.

One of the ways that you indicate that you trust a certificate authority is by having the certificate for that authority installed on your system. When certificates are issued, they come with a chain. At the very top is the certificate authority. This is called the *root certificate*. The root certificate is the only certificate in the chain of certificates you will see that is self-signed because there is nothing above it. The way you prove that the certificate you are looking at is trustworthy is to check it against the certificate you have installed for the certificate authority. If the certificate authority has signed the certificate and it's a valid signature, meaning it was definitely signed by the certificate you have installed for the certificate authority, then the certificate is considered valid, meaning it's trustworthy. It does mean that you have to have the certificate for the certificate authority installed. Most of the well-known root certificates for certificate authorities like Verisign, Symantec, Google, and others are installed for you by the operating system or browser vendor. If there are other certificate authorities you know about and can verify that they are legitimate, you can install their certificate for yourself. Once you have done this, all certificates issued by that authority would be, again by the transitive property, trusted by you.

A chain of certificates, which may be several deep depending on how many intermediate systems were also involved in generating your certificate, is called a chain of trust. The trust anchor—the entity that all trust is tied to—is the root certificate, which belongs to the certificate authority. If you trust the certificate authority, all of the cascading certificates would also be considered trustworthy. In the case of PGP, which is not used in web communications but which also uses certificates, it would not be a certificate authority but instead would be a number of users that sign the certificate, providing its validity. If you know Milo Bloom and you trust him to sign certificates for others, and you see that Milo has signed the certificate of Michael Binkley, you, by extension, trust that the certificate of Michael Binkley is legitimate and really belongs to Michael Binkley.

Cipher Suites

We talked briefly about cipher suites earlier in this section. It is worth going into a little more detail rather than just passing quickly over a single example. The cipher suite is a defined collection of the following components of the entire encryption process. These components are used through TLS 1.2.

- Authentication
- Encryption
- Message authentication code (MAC)
- Key exchange

All of these components represent different algorithms. The authentication piece could use either RSA or DSA, which are both examples of asymmetric encryption algorithms. RSA is a proprietary algorithm developed by the men it is named for—Ron Rivest, Adi Shamir, and Leonard Adleman. DSA, on the other hand, is the Digital Signature Algorithm and is an open standard endorsed by the National Institute of Standards and Technology (NIST). Both of these are used to verify identities in

certificates. If you are presented a certificate, you can use that certificate to verify an identity because of the trust relationship discussed earlier. If a certificate authority you trust says a certificate belongs to `www.amazon.com`, it belongs to `www.amazon.com`.

Over the course of a communication stream between a client and server the encryption would use a symmetric algorithm like AES, IDEA, DES, or another symmetric algorithm. Taking an entire communication stream and encrypting it is sometimes called bulk encryption. When negotiating the bulk encryption to be used, one of the elements would be the key length. Some algorithms, like AES, include the ability to support multiple key lengths, so it's important that both ends understand the length of the key to use. It's no good having one side generate a 128-bit key and encrypting with that when the other side is using a 256-bit key. If the keys don't match exactly, the process of communication won't work.

The message authentication code (MAC) is typically a message digest or hashing algorithm. As an example, the Secure Hash Algorithm (SHA) can be used to take variable length input and generate a fixed-length value from that. This value can then be used to compare against as a way of ensuring messages were not tampered with. If I send an encrypted message to you and a hash value, no one would be able to intercept that and send you a different message or even a modified version of the original message because the hash values would not match. Since a cryptographic hash like SHA cannot be reversed (they are considered one-way functions), meaning you can't take a hash value and derive the original message from it, an attacker would have no way of generating a message that matched the hash value without a lot of time and effort. If he did generate a second value that matched the hash, it would be considered a collision, which is something these hashing algorithms are designed to avoid. Longer hashing algorithms like the 384-bit SHA384 would be considerably harder to get a collision in than the much smaller MD5, which only uses 128-bits, or even SHA-1, which uses 160-bits.

Finally, the key exchange algorithm that would be used has to be selected. Diffie-Hellman would be a common approach to exchange keys in the open, meaning without encryption to protect them. Diffie-Hellman doesn't exchange keys directly. Instead, it allows both sides of a communication stream to generate the key to be used independently. Both sides end up with exactly the same key without ever actually exchanging the key itself.

All of these individual components are essential to the successful encryption of messages. TLS is the primary protocol in use for transmitting encrypted messages over network communications. Other protocols, including SMTP, POP3, and FTP might make use of TLS to transmit encrypted messages. The more people who use the Internet and the more malicious users there are, the more formerly cleartext protocols need encryption over the top in order to protect the users of those protocols.

SSLScan, SSLStrip, and Encryption Attacks

There are two open source programs that you should be aware of. You saw the first earlier in this section in a very limited capacity. SSLScan can be used to obtain information about the cipher suites that are in use by any server that supports SSL or TLS. In addition, SSLScan can determine whether

a server is vulnerable to the HeartBleed bug that had the potential to take an encrypted communication stream and offer a way to decrypt it, exposing the information in the clear. Listing 12-1 shows partial output from SSLScan, run from a Kali Linux system, checking the Microsoft website, `www.microsoft.com`. You can see from the output that the underlying library, OpenSSL, isn't a version that supports SSLv2, which was deprecated and prohibited in 2011. As a result, there were no checks done against SSLv2. You'll also note that SSLScan did run checks to see if the server was vulnerable to the HeartBleed bug.

Listing 12-2: SSLScan Output

```
kilroy@oliver:~$ sslscan www.microsoft.com:443
Version: 1.11.8
OpenSSL 1.0.2k 26 Jan 2017

OpenSSL version does not support SSLv2
SSLv2 ciphers will not be detected

Testing SSL server www.microsoft.com on port 443

  TLS Fallback SCSV:
Server supports TLS Fallback SCSV

  TLS renegotiation:
Secure session renegotiation supported

  TLS Compression:
Compression disabled

  Heartbleed:
TLS 1.2 not vulnerable to heartbleed
TLS 1.1 not vulnerable to heartbleed
TLS 1.0 not vulnerable to heartbleed
```

You can use SSLScan to determine the capabilities that are supported by a server. This doesn't get you to the point where you can see any data, however. For that, you need a tool like `sslstrip`. This is a program that takes advantage of weaknesses in some of the versions of SSL to decrypt messages. It does, however, require that the program can get all of the messages from a communication stream in order to be able to do what it does. This requires some understanding of manipulating networks using Linux. The following steps are required to get `sslstrip` to work and have the potential to see decrypted information:

1. Install `sslstrip`, which may be available in the package repository for your Linux distribution
2. Enable forwarding of packets on your Linux system (`sudo sysctl -w net.ipv4.ip_forward=1`).

3. Use an ARP spoofing attack (Ettercap, arpspoof) to get all of the messages on your network to come to your machine. If you don't enable forwarding, the communication stream between the endpoints will break because you will be getting messages destined for someone else and they will never reply.
4. Configure a port forwarding rule using iptables to make sure messages to port 443 are sent to a port you want sslstrip to listen on (e.g., `iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT --to-ports 5000`).
5. Start sslstrip, telling it which port to listen on. This may look something like `sslstrip -l 5000` or `sslstrip --listen=5000`, which is the same command expressed a different way. This needs to be the port you redirected traffic to in your iptables rule.
6. Sit and wait for sslstrip to do what it does.

This is referred to as a man in the middle (MiTM) attack and it's a common approach to going after encrypted messages. If you are unable to sit in the middle of the communication stream, you will be unable to gather enough information to determine how the encryption is happening. Without any of that information, you are going to be utterly blind. Once you are sitting in the middle gathering information, though, you may be able to make some changes.

One way of implementing a man in the middle attack, though it will generate authentication errors because you can't pretend to be someone you are not without being caught, is to terminate one end of the communication stream and re-originate it on the other side. This means that when a client attempts to communicate, you pretend to be the server and respond as such. On the other side, you pretend to be the client, originating messages to the server. This is not nearly as easy as it sounds and it is prone to error, especially because any certificate you send to the client pretending to be Amazon simply won't pass the sniff test. Of course, certificate errors have gotten to be so common that users may simply click through any error they receive.

Another attack is called a bid down attack. If you can, again, get in the middle, you may be able to get the server and client to agree to a lower strength cipher suite. The lower strength may allow you to go after the ciphertext and decrypt it. Weaker encryption may be susceptible to different attacks that can lead to decryption. This is because either the encryption algorithm has already been chosen or because the key strength is so low that it may be susceptible to something like a brute force attack. Smaller key sizes mean far fewer possible keys that can be used. A 40-bit key, for instance, only has the potential for 2^{40} possible keys. This is considerably fewer, by several orders of magnitude, than a 128-bit key, which has 2^{128} possible values.

Encryption protocols, algorithms, and ciphers are constantly under attack. The longer they're around, the more time people have to really look at them to identify potential weaknesses. Additionally, the longer they are in use, the more computing power comes into play. Computers today are vastly more powerful than the ones 20 years ago when the AES algorithm was being selected from the many prospective algorithms under consideration. This means that attacks that require faster processors are feasible now when they would have been unthinkable then. This is why key sizes keep increasing,

as do the sizes of the hash value used for message authentication. Bigger keys and hash values make for many more possibilities, which increases the computation needed to go after them.

NOTE Earlier there was a reference to obtaining network traffic, which can be done on a local network using a technique called *ARP spoofing*. While you can do this using tools like arpspoof and Ettercap, it does manipulate the flow of traffic, causing it to pass through a system it wasn't meant to pass through.

Cloud Computing

Cloud computing is an ambiguous term. It covers, as they say, all manner of sins. The term originates from the fact that large, open networks like the Internet are commonly depicted as clouds on network diagrams. Clouds are used because they are amorphous, insubstantial, and almost entirely opaque. You can't see inside a cloud from the outside, and you also may not be able to see inside the cloud if you are in it. Objects that pass into the cloud can pass out the other side entirely unaltered, and we just don't know what happens inside.

When you take the idea of a cloud as a stand in for the Internet and you add in service providers and accessibility through the Internet, you get *cloud computing*. However, even with that bit of clarity, the term cloud computing is still vague and not especially transparent. The reason for this is there are multiple types of cloud computing. Ultimately, what cloud computing means is that a computing service is being provided somewhere in a network, often using web-based technologies, rather than locally on individual systems. Because these are the ones that are most important for our purposes, we're going to talk about Infrastructure as a Service (IaaS), Storage as a Service (SaaS), and Software as a Service (SaaS).

Infrastructure as a Service

Imagine that you have a way to make a computer very, very small. So small that you can then stuff a lot of them into a single piece of hardware that you would normally think of as a computer system. Imagine that you can install Windows or Linux on every single one of those computers and then you can get access to every one of those operating systems individually. You have a way to dump a lot of systems into a single container, such that every one of your "tiny" systems believes it is running on a regular piece of computer hardware.

All of these smaller systems are crammed into one piece of hardware by virtual machines. A *virtual machine* looks to an operating system like actual, physical hardware. In fact, the OS is interacting with software and a set of extensions within the central processing unit (CPU) of the computer system. This makes it possible to run a number of virtual computers within a single computer system. One reason this is possible is because current CPUs are so powerful that they are often idle. Even running

multiple operating systems with all of their associated programs and services, modern CPUs are not highly stressed. Virtual machines make efficient use of modern, fast, powerful hardware by allowing much more to run on a single computer system and CPU.

Why do we bother doing this? Well, computer hardware is comparatively expensive. If you can run half a dozen systems on every piece of computer hardware and save the extra hardware expenditure, lots of companies can save a lot of money. Why would a company want to buy separate machines for a web server, a mail server, a database server, and a file server when they can buy a single machine? Additionally, every computer system purchased costs power, floor space, cooling, and maintenance. The fewer computer systems needed, the more money the company saves. You can see why virtualization would be popular for companies that can make use of it. Managing virtual systems isn't free, however. It does require systems powerful enough to run all of the services the company needs and the software, called *hypervisors*, needed to run the virtual machines. That software needs management and maintenance, which means paying someone who knows how to run it.

What does all of this have to do with Infrastructure as a Service (IaaS)? There are companies that can offer up access to virtual machines at a moment's notice. Some of this started with companies like Amazon that had an excess of capacity and the infrastructure in place to manage these services on behalf of customers. However, this is just an extension of the outsourcing that has long been going on. Companies have been buying outsourced computing power for decades. Even virtual machines have been around for decades. Finally, the cheaper computing power and easier delivery came together to allow companies like Amazon, Google, and many others to offer virtual infrastructure to companies.

One significant advantage to going with one of these providers is that they have all of the deployment, configuration, and management automated. You can very quickly set up an entire infrastructure for anything you need, including complex web applications, in a very short period of time without all of the hardware expenditures that would otherwise be necessary. You can see how this would be appealing to a lot of companies. This would be especially true of smaller companies that get quick and easy access to a lot of computational power without having to put up a lot of money up front.

Where there have been companies that would put up hardware for you and allow you access to it or even allow you to put up your own hardware at their facilities, management could be done physically unless you had remote management configured. In the case of IaaS providers, everything is done through a web interface. As an example, Figure 12-3 shows that you can quickly set up a number of virtual machines that can support web applications, mobile application back ends, or other requirements you may have.

Even if all you want to do is just set up a single virtual machine, that process is very simple and you can have a choice of a number of operating systems that will already be installed on the system you get access to. You aren't being given just a hunk of bare metal that you need to do a lot with to make it useful. You are being presented with a working, completely configured operating system in a matter of minutes. In Figure 12-4, you can see a small sample of the number of operating systems that are available. The figure shows the Ubuntu Server, but other flavors of Linux server are available as well.

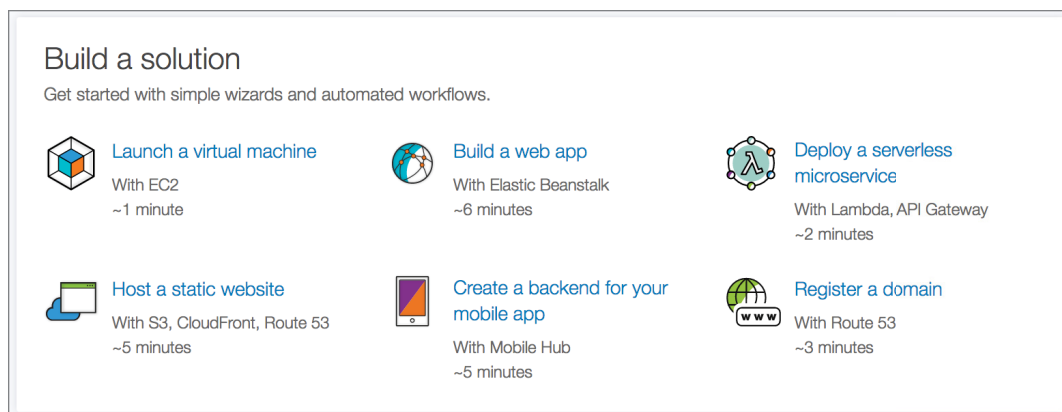


Figure 12-3: Amazon EC2 workflows.

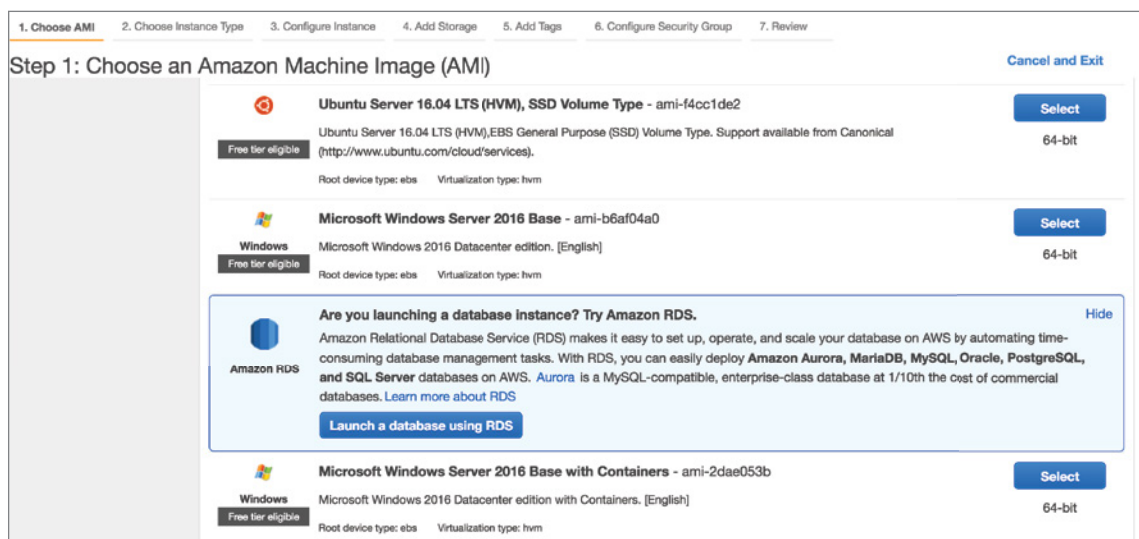


Figure 12-4: Amazon EC2 virtual machine choices.

Of course, Amazon is not the only company in this business. Google, Microsoft, and several others offer this. What this means, though, is that when a company stands up a set of systems using any of the service providers, any data that is being managed by those servers is stored not on the company's systems but instead on storage owned and managed by the service provider. What it also means is that every interaction with this infrastructure happens over the network. This is no longer just the local network. Management and application traffic all happens over the same set of wires, where normally they are separated when companies use their own networks.

Storage as a Service

Not everyone needs entire computing systems. Sometimes all you need is a place to store some things for a while, or even permanently. When a service provider puts together what is needed to service customers' infrastructure needs, one thing it has to do is make sure it has a lot of storage capacity. Virtual machine capacity and storage capacity are not directly related since different virtual machines have different storage needs. It's not as simple as 10 virtual machines will need 10 terabytes of storage. One of the virtual machines may need only 10 gigabytes of storage while another may want 20 terabytes. This is why service providers that offer up infrastructure services will also have storage capacity as well.

Where formerly we made use of File Transfer Protocol (FTP) servers to share information across the network, these days we are more likely to use storage devices that are accessible through a web interface. You are probably even familiar with many of the storage services that individuals and companies are likely to use. Google's is Drive, Microsoft has OneDrive, Apple has iCloud, and there are also companies that do nothing but offer storage like Dropbox, Box.Com, and others. Access to all of these is through a web interface or maybe a mobile application that uses the same web-based protocols that your browser uses to access the web interface. You may also be able to get a service that presents the cloud storage as just another folder on your computer. This is true of services from Dropbox, Box, Microsoft, Google, and Apple and perhaps others as well.

In most cases, you will find that the web interface looks more or less like any other file browser that you would see, whether it's the Windows File Explorer or the Mac Finder or any of the different file managers that are available for Linux. You can see an example in Figure 12-5 from Microsoft OneDrive. There is a list of files and folders, just as you would see in any other file browser.

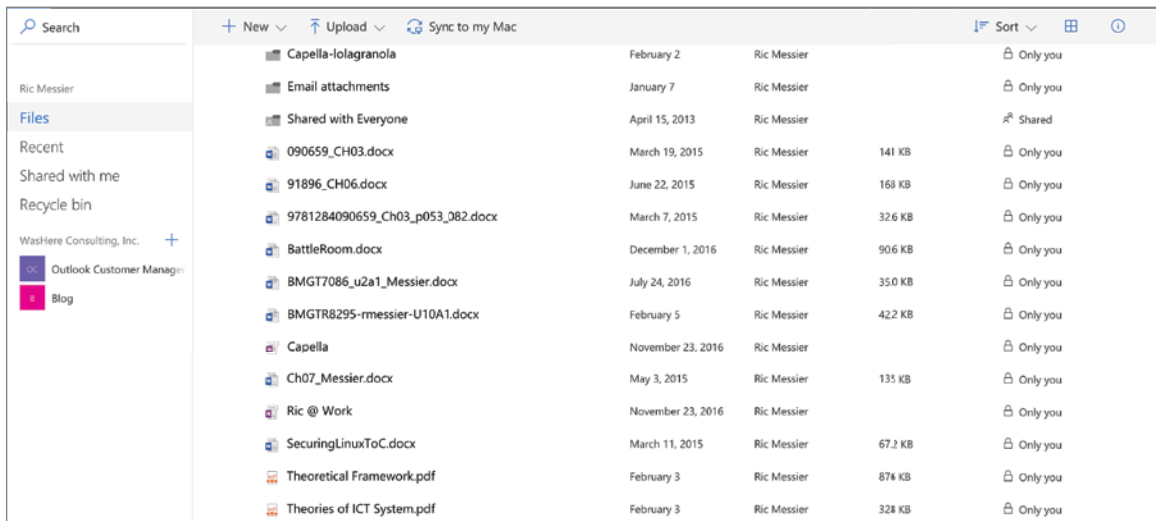


Figure 12-5: Microsoft OneDrive interface.

One thing you don't see from the image in Figure 12-5, though, is the address bar and the fact that the communication is secured. You can't see that the communication is encrypted using AES with 256-bit keys. The certificate in use has a 2048-bit public key that is used, as noted earlier, to protect the 256-bit session key. This is Microsoft and it has the infrastructure to be able to support strong encryption on communications with its services. Not all providers will have that capability, but since it's really easy to do even basic encryption, especially over a web interface, it would be rare to find a storage service provider that didn't encrypt its network transmissions.

Software as a Service

Over the course of decades, the information technology industry has gone from centralized with mainframes to decentralized with personal computers, back to centralized with so-called thin clients, then back to decentralized as computing power and storage got to be cheap. We are now back to centralized again. As infrastructure became easy to do with service providers, along with even more powerful processors and much cheaper storage, it was cost efficient to start providing access to common software and services through a web interface.

There is a significant advantage to providing software through a service provider. As companies become geographically more dispersed, picking up talent where the talent lives, they need to be able to provide their employees access to business data and applications. This is where Software as a Service (SaaS) becomes so beneficial. Previously, remote employees required virtual private networks (VPNs) to get access to corporate facilities. The VPN provided an encrypted tunnel to protect business data. If you access data by way of HTTP and encrypt it with TLS, the effect is the same. The web interface becomes the presentation layer and all protocol interactions happen either over HTTP to the user or over whatever protocol is necessary on the back end.

You may be familiar with a lot of these SaaS solutions. Some companies have developed their business around them. If a company needs a contact management or customer relationship management solution, it may well be using Salesforce.com. This is an application that stores contacts and all information related to engagement with those contacts. Storing information with an application service provider helps any employee gain access to that information remotely. It also relieves the problems of syncing data for anyone who may be making changes offline. Since the access is always online, there are no issues with syncing. All changes are made live and you can have multiple people working within the data set at any point. Previous solutions relied on a central database that everyone synchronized their data with. They may have worked online or they may not have.

Other solutions are tied directly to the storage solutions just mentioned. With the ability to store and share documents online, companies like Microsoft and Google offer office productivity suites entirely online. You no longer need Word and Excel installed locally. You can just open them up in a web browser. Word looks and behaves very similarly to the one you would have installed locally. Figure 12-6 shows the page where you can select from different templates in Word to create a new document. This is the same as current versions of native Word (the application designed to run locally). An advantage to online versions of these applications is the ability to share the document.

Rather than having to e-mail it out every time it changes or editing from a file share or any other way of handling this task previously, every change is made in real time. If you are editing a document and someone opens it up, they can see the changes you have just made. They can also edit simultaneously.

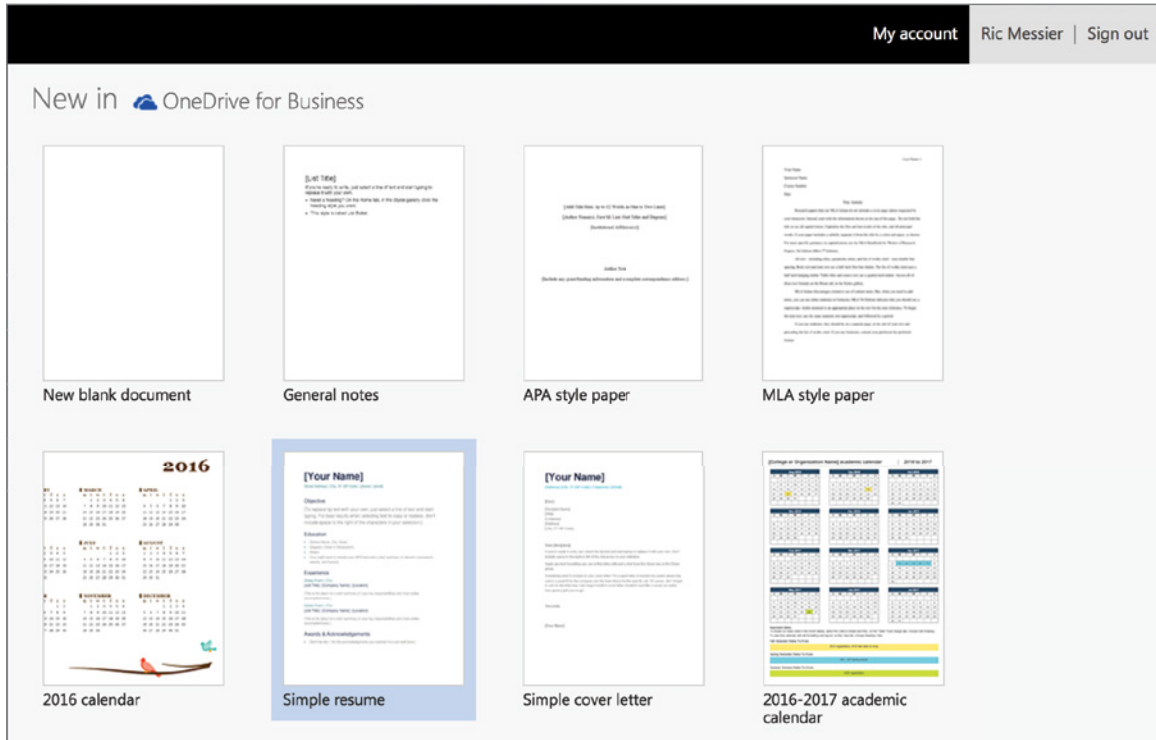


Figure 12-6: Microsoft Word new document.

With a solution like this, layered on top of a storage solution, you are gaining access to the documents stored in the storage solution. You are able to make edits and changes to those documents, leaving the documents local to the storage provider. Everything is done over the network, rather than leaving any artifact on the local system. Solutions like this have a number of benefits. One of them is that people can work wherever they are, on whatever system they have access to. Since there are no local artifacts other than Internet history and potential memory artifacts, most of the artifacts will be either network-based or they will reside with the service provider.

Other Factors

Data in cloud-based solutions, regardless of what the solution is, resides with the service provider. Even in the case of IaaS, where the only thing the service provider is providing is remote access to virtual machines and the underlying storage, gaining access to the artifacts quite likely requires

going to the service provider. As an investigator, you could, of course, gain the same remote access to the service that the customer has but you would be probing live systems and, in the process, potentially altering that data. In order to get raw data (files, logs, etc), you would need to go to the service provider, with appropriate legal documentation, and ask for the information. This is a very common approach. To get a sense of how common, you can review the transparency reports from service providers like Google or Microsoft. Google's transparency history graph is shown in Figure 12-7. In the first half of 2016, which is the last set of data available at the time of this writing, Google received nearly 45,000 requests.

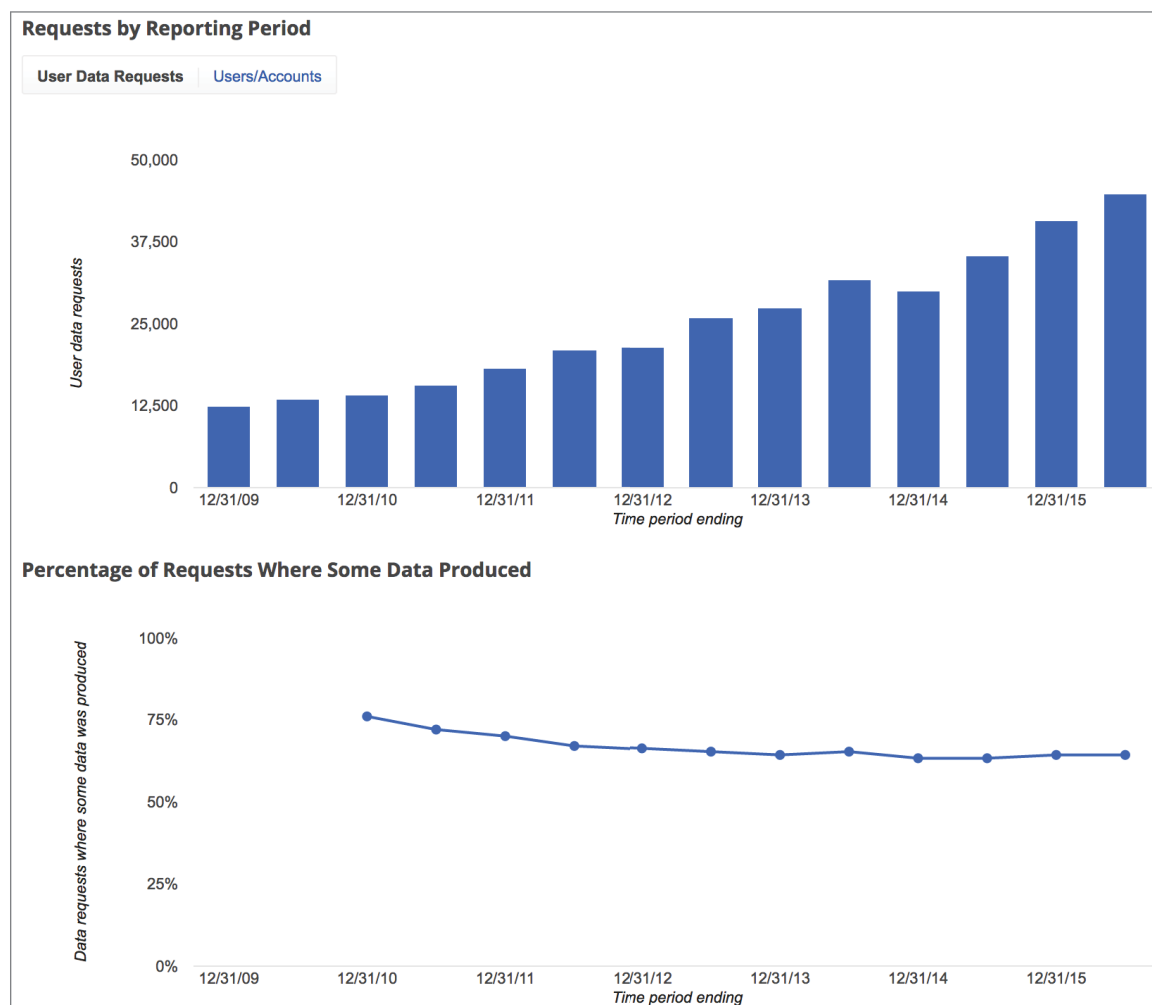


Figure 12-7: Google transparency report.

The bottom chart shows that Google is able to provide data in roughly 75% of the cases. What is not clear from any of this is why exactly Google was not able to provide data in the other 25% of cases. Microsoft sees similar numbers in terms of legal requests. In Microsoft's case, it provides more specific graphs. You can see in Figure 12-8 what it disclosed from the more than 35,000 requests it received in the first half of 2016. The number of requests received is based on everything they get from around the world. Just over 5,000 of those requests came from the United States, and nearly 5,000 requests came from the United Kingdom. What you can see clearly from the pie chart is that the majority of cases only provide subscriber information. What isn't clear from the graph is whether that was all that was asked for.

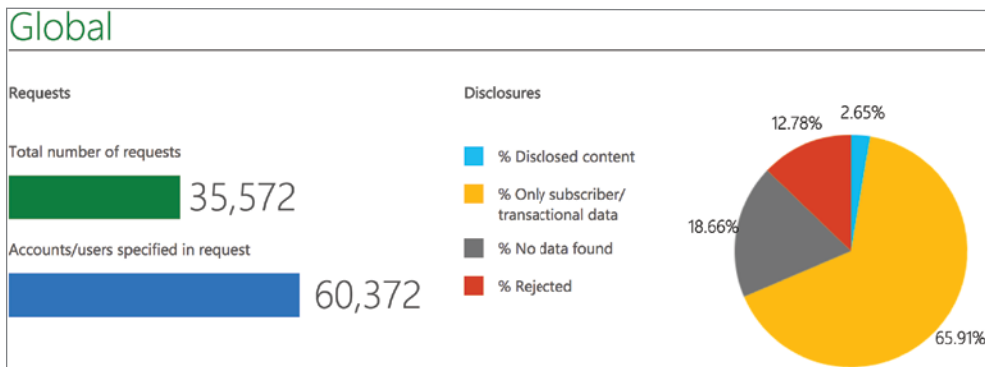


Figure 12-8: Microsoft transparency report.

This brings up the importance of network investigations. We need to be able to perform investigations on the network events as they happen, even if the events represent interaction with service providers, rather than relying on the ability to have quick and easy access to all data files and relevant information. Being able to at least observe interactions with these service providers, even if we may not be able to see what is being transacted, at least gives us information about when users were gaining access and what services they were gaining access to. In most cases, unless you have something sitting in between the user and the outside of the network, you are going to have a hard time seeing the actual content of the data because it will be encrypted. Larger service providers will have their encryption house well in order, but there are constantly attacks on encryption, resulting in regular changes to what is allowed and what isn't.

Even when you can see how users are engaging with service providers, they may be using Asynchronous JavaScript and XML (AJAX). What AJAX provides is the ability for the server to reach out to the client without the client requesting a page. The way HTTP was initially developed, it assumed clients would make requests of servers that would respond. As applications delivered over HTTP have become far more complex, a reverse route was needed, which meant the server could

autonomously send data to the client, which would be displayed to the user. When AJAX is used to communicate to the client or even for the client to send data to the server without the user's involvement, it may be done in small chunks rather than entire documents at a time. You may get a single packet containing a handful of characters that are sent to the server. This may then require a lot of work to piece everything together.

There is a slim possibility of getting data from a proxy server in this case, but it's not common. SSL/TLS are supposed to guarantee end-to-end encryption, which means that the browser should send a `CONNECT` message to the proxy server. This essentially turns the connection into a raw TCP connection rather than an HTTP connection. However, it is possible for the proxy to see the message by allowing it to sit in the middle of the request, providing encryption to the proxy and then again from the proxy to the end server. Some proxies will allow this behavior, but there is the potential for it to cause problems with the browser. The browser will want to verify the certificate, but the certificate can't match if the browser is trying to compare names in the certificate with the name of the server being requested.

As more and more services go to the web, one of the drivers continues to be the use of smartphones. People are transacting business less on traditional computers like desktops and laptops and more on mobile devices like smartphones or tablets. Either way, the use of these small form-factor devices is driving more uptake of web-based communications. As mobile applications are developed, it's far easier to just use web protocols since the access to those are built into the programming libraries used to develop them. Because it's easy, available, and well-understood, it makes sense to use it. Also, no additional infrastructure is required to support these mobile applications. A company can just make use of its existing web infrastructure to support these applications. In the end, they are exposing functionality that is already available in the primary web application.

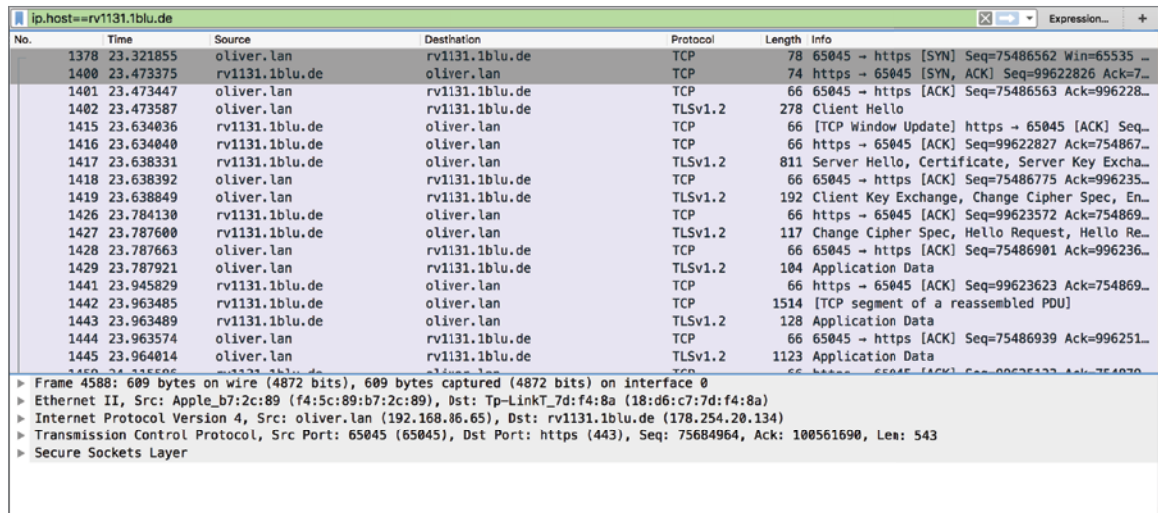
The Onion Router (TOR)

There's one last topic to cover: dark nets. They are overlays to other networks, like the Internet. While you gain access to them over the Internet, meaning you can get to it from your cable modem, DSL connection or mobile device, the services require that you access them using services that lie over the top of those you would normally access through the Internet. One common dark net is The Onion Router (TOR). TOR was developed by United States Naval Research Laboratory employees in the mid-'90s and was conceived as an anonymity project. Tor (no longer an acronym) sits on top of the Internet but provides services to users in a way that the user sees no difference from normal Internet access. What is different is what happens on the other end of the connection.

In normal Internet use, if you want to gain access to a service, you would send a request to figure out the address of that service. You probably know a hostname and your application knows a port to connect to, whether it's a mail client or a web browser, but what you need is an IP address. Your computer sends out a request to a DNS server to get the IP address from the hostname you provide. Your computer then sends out a request to the server. All along the way, you are leaving

bread crumbs—your IP address is leaving a trail that can be followed, because you left it with the DNS server you are trying to talk to in order to get to a website. Additionally, your IP address has traveled through multiple routers to get to the DNS server and then to the web server. Those routers have dutifully sent your request along and then the response back to you, based on the IP addresses in the IP headers.

In the case of Tor, your requests are sent through a peer-to-peer network. The network itself is Tor. Your request is routed through the Tor network and eventually exits looking as though it came from someone else entirely. This is how you obtain anonymity. No request to an end server ever appears to originate from you. Instead, it comes from the network and some departure node. You also need to have an entry node to match the departure node. This means that when you connect to the Tor network, you aren't just throwing messages out to the Internet. You are communicating directly with what is, in essence, an application-layer router. You can see the initial connection to the entry node (rv1131.bl.u.de) to the Tor network in Figure 12-9. (This is a TLS connection.)



The image shows a Wireshark packet capture window with the filter 'ip.host==rv1131.bl.u.de'. The packet list shows a series of TCP and TLSv1.2 packets between 'oliver.lan' and 'rv1131.bl.u.de'. The packet details pane for packet 1445 shows the following information:

- Frame 4588: 609 bytes on wire (4872 bits), 609 bytes captured (4872 bits) on interface 0
- Ethernet II, Src: Apple_b7:2c:89 (f4:5c:89:b7:2c:89), Dst: Tp-Link_7d:f4:8a (18:d6:c7:7d:f4:8a)
- Internet Protocol Version 4, Src: oliver.lan (192.168.86.65), Dst: rv1131.bl.u.de (178.254.20.134)
- Transmission Control Protocol, Src Port: 65045 (65045), Dst Port: https (443), Seq: 75684964, Ack: 100561690, Len: 543
- Secure Sockets Layer

Figure 12-9: Wireshark capture of Tor communication.

The application I used to get access is the Tor Browser. The Tor Browser is just a browser, like Google Chrome or Microsoft Edge. However, underneath the browser interface is the ability to gain access to the Tor network. The browser takes care of all of the exchanges with the network and all of the encryption necessary to make it work. The Tor Browser is available from the Tor Project directly for Windows, Mac OS X, Linux, and Android. This allows you to connect to the network and then configure your own applications to make use of that network.

While you can use Tor as your browser and it will work with websites just as other browsers do, Tor also provides more than just anonymity to clients. Additionally, there are servers that are only

accessible within the Tor network and it's really these servers and services that comprise the "dark web." Over the years, there have been a number of sites selling products and services through the Tor network. One of the most notorious was Silk Road, which was a site dealing drugs through the Tor network. It was shut down in 2013. Many other sites that have previously been available are now defunct.

A disadvantage to Tor is that while it provides anonymity for people who have legitimate need for it, it can also be a hiding place for criminals and criminal activity. Although there are projects focused on rooting out the criminal activity, it does become a bit like whack-a-mole, much like other attempts to deter criminal activity. Tor makes it hard to identify where people are because of the way the network operates. It is intentionally that way. Since anonymity is the point, you can see why criminals would be drawn to a service like Tor. This is not at all to say that Tor is the only place where criminals operate, because there are plenty of criminal activities on the open Internet. It's also not to say that only criminals utilize the Tor network. There are often attempts to self-police the service. You can see in Figure 12-10 that the search engine Ahmia searches for hidden services on the Tor network but also that it doesn't tolerate abuse. It keeps a blacklist of services and encourages users to report abuse so that abusive services can be blocked.

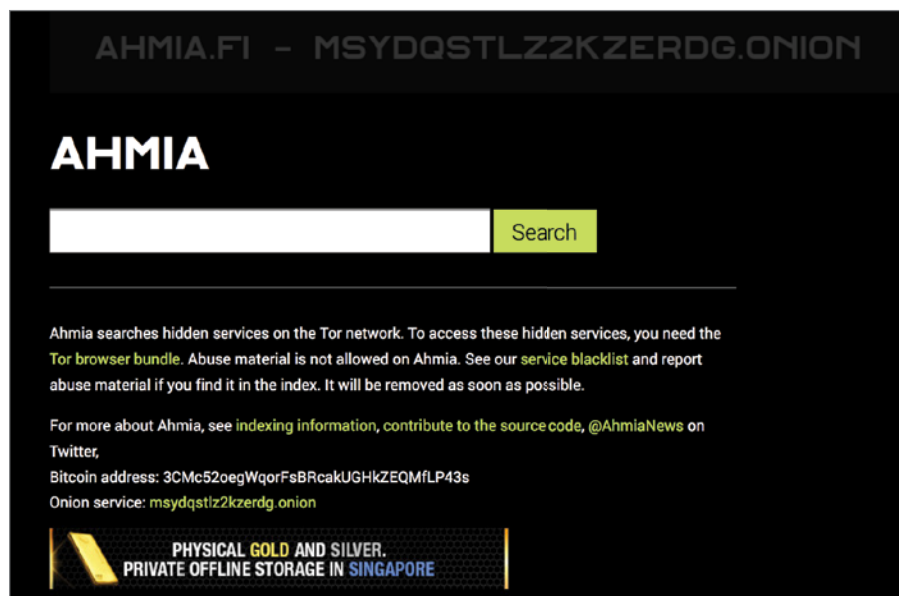


Figure 12-10: Ahmia search site.

It is important to recognize that Tor uses common web ports. You can see this when you look closely at a packet capture. This means that Tor traffic can't easily be blocked. It looks and behaves

just like regular web traffic. If you were to see a packet capture of someone using Tor, it may be very difficult to distinguish between the Tor traffic and another capture of web traffic.

One piece of data you can use to determine if you're seeing Tor traffic is the name of the host on the other end of the connection, though this is not always the case. `rv113.1blu.de` doesn't look much like a common Internet address to be communicating with. Looking a little further, as shown in Listing 12-3, it appears that the domain name `1blu.de` belongs to an individual rather than a business or an organization.

Listing 12-3: Whois Request

```
[Tech-C]
Type: PERSON
Name: Johann Dasch
Organisation: 1blu AG
Address: Stromstr. 1-5
PostalCode: 10555
City: Berlin
CountryCode: DE
Phone: +49 30 20181000
Fax: +49 30 20181001
Email: info@1blu.de
Changed: 2006-02-27T14:24:03+01:00
```

This also is not necessarily suspicious, since anyone can get a domain name and set up a website. This sort of digging may be needed in order to determine whether someone is using a Tor browser or just a regular browser to connect to websites over the Internet. One thing you would note from a Tor connection is the duration of the connection and the amount of traffic transmitted. Over the course of a few minutes, I visited a few websites and the Wireshark capture showed that the host I was exchanging information with never changed. This doesn't mean that it never changes, but it certainly doesn't based on individual transactions over a period of a few minutes.

Summary

One of the biggest challenges associated with network investigations is encryption. Over the course of the book, I've mostly skirted the issue, demonstrating a lot of network traffic that takes place in the clear. It would be disingenuous to pretend that encrypted traffic doesn't happen. This has a profound effect on forensics because it is difficult, if not impossible, to obtain decrypts of the ciphertext that has been transmitted. Even if you are able to obtain a complete packet capture, encryption mechanisms are designed to not be reverse engineered. You can see the key exchange happening but you can't get the key. It is never transmitted. You might have the public key used to transmit the data, but that's

insufficient for decryption. Without the private key, the public key is meaningless. Encryption will make your life very difficult. However, what you will always get, even when the data is encrypted, is the metadata, which is the source and destination of the traffic as well as the timing of the connection. While the time isn't transmitted as part of the packets, you do know when packets arrived relative to the start of the capture. This means you know when the network communication took place, assuming the timestamp on the file is accurate, which is a reason to ensure all of your systems are synchronized to a reliable time server.

Cloud-based services are available over the Internet, typically offered by a service provider, and you will encounter them very commonly in investigations. Whether the individual or organization is using a service provider for virtual machine access (IaaS) or just to host files (SaaS) or maybe even entire applications (SaaS), you will be able to see users gaining access to these services. Unfortunately, this is another area where you are likely to be foiled by encryption. Most service providers will offer their services using TLS, because these cloud-based services are offered using web-based technologies for the most part. Doing encryption through a website is simple and well-known. It's also cheap. Even strong encryption is no problem, so finding a service provider that is still offering services without encryption is going to be a challenge.

Savvy users, concerned with their privacy, may be using Tor as a way of communication. The Tor network provides services just as the open Internet does. Users can get e-mail over the Tor network and they can do peer-to-peer communication just as they could with Google Hangouts, Skype, Yammer, Slack, and other messaging applications. The communication happens, completely encrypted and anonymous, across the Tor network. Using the Tor network, you are anonymous because your address is hidden by the network. It's important to remember that Tor is just an overlay on top of the Internet—it is a logical construct rather than a physical one. All of the systems that are on the Tor network exist, by definition, on the Internet. Tor is just the way they have all connected themselves using the Tor program.

Index

Symbols and Numerals

- # (pound sign)
 - for decimal encoding, 136
 - for Snort comments, 196
- \$ (dollar sign), in Snort configuration, 194
- % (percent sign), in URL-encoding strings, 135
- & (ampersand), for decimal encoding, 136
- 802.11 (WiFi), 20

A

- Abstract Syntax Notation One (ASN.1), 121, 123
- abuse contact, 24
- abusive services, 316
- access control lists (ACLs), 177
- ACK flag, in TCP header, 35
- ACK message, 37
- acknowledgment number, in TCP header, 34
- Acon Timeline, 258
- Address Resolution Protocol (ARP), 48–49, 77–78, 94
- address space, IPv6 and, 32
- Adleman, Leonard, 302
- Admin log (Windows), 226
- administrative privileges, sudo for temporary, 268
- Advanced Research Projects Agency (ARPA), 14, 20
- AES (Advanced Encryption Standard), 295, 299
- African Network Information Center (AfriNIC), 23
- Ahmia search engine, 316
- alerts, 196
 - from IDS, 207–208
 - SYN scan, 208
- AlienVault, 184
- Amazon, 291
 - EC2 virtual machine choices, 308
 - EC2 workflows, 308
- American Registry for Internet Numbers (ARIN), 23
- ampersand (&), for decimal encoding, 136
- analyzer programs, 238
- anomaly-based detection, 189–190
- anonymity, Tor and, 315–316
- antivirus programs, 180–181
- Apache web server, 240–243
 - log storage, 170
- Apple, iCloud, 309
- application attacks, 113–114, 136–140
- application layer (OSI), 18
- Application layer (TCP/IP), 19
- Application log (Windows), 224, 225
 - clearing, 231
- ARP poisoning, 95
- ARP request, in tcpdump, 87
- ARP spoofing, 49, 94–96, 305, 306
- Arpanet, 16
 - hosts file, 42
- artifacts, 3. *See also* host-side artifacts
- ASCII-printable string, in packet search, 201
- Asia-Pacific Network Information Center (APNIC), 23
- asymmetric encryption, 295–296
- Asynchronous JavaScript and XML (AJAX), 313–314
- Asynchronous Transfer Mode (ATM), 20
- attack. *See also* correlating attacks
- attack preparation, 159–185
 - logging, 165–180
 - NetFlow, 160–165
 - syslog, 166–171
- attack types, 113–141
 - application attacks, 113–114, 136–140
 - denial of service (DDoS) attack, 114–130
 - amplification attacks, 124–126
 - backscatter, 128–130
 - distributed attacks, 126–128

- malformed packets, 118–122
- SYN floods, 115–118
- UDP floods, 122–124
- evasion, 114, 134–136
- insider threats, 132–134
- vulnerability exploits, 130–132
- authentication, in cipher suite, 302
- authenticity, cryptographic hashes as test of, 5
- authoritative server, 43
- authorization messages, logging, 220–221

B

- backdoor, 54, 265, 266, 277
- background, program running in, 54
- backscatter, 128–130
- bandwidth
 - proxy servers and, 237
 - UDP flood to consume, 122–124
- Base64 encoding, 135, 137
- baseline, in anomaly-based system, 190
- Basis Service Set Identifier (BSSID), 157
- Berkeley Packet Filters, 89
- Best Current Practice, in RFC, 21
- best-effort delivery model, 31
- bid down attack, 305
- binary operation codes, 131
- binary protocol, 121, 122
- birthday paradox, 5
- black hole, 128
- Bolt, Beranek, and Newman (BBN), 20
- bootstrap protocol (Bootp), 48
- bot, 127
- botnet, 126–127
- Box.Com, 309
- British Broadcasting Corporation (BBC),
 - attack against, 127
- British Summer Time (BST), 247
- Bro, 187, 191, 203–205
 - event script, 203–204
- bro_init event, 204
- broadcast address, 30, 46
 - pinging, 125
- broadcast domain, 134
- broadcast MAC address, 94
- broadcast traffic, 129
- BSD Packet Filter (BPF), 191
- buffer overflow, 131–132
- bulk encryption, 303

- burst rate, 104
- byte, 36
 - count in Wireshark, 103

C

- C programming language, 132
- cache, for proxy server, 237
- caching DNS, 43–44
- Caesar cipher, 292
- Calce, Michael (Mafiaboy), 8, 127
- Carbon Black, 182
- centralized logging, 216–219, 251
 - servers for, 165, 169
- certificate authority, trust in, 302
- certificates, 296, 300–302
 - Amazon web server, 301
 - from server, 299
- chain of custody, 8
- chain of trust, 302
- ChangeCipherSpec message, 299
- checksums, 7, 28
 - in TCP header, 35
 - from Wireshark, 100
- cipher suite, 298, 302–303
- ciphertext, 292
- civil law system, in Louisiana, 4
- Class A/B/C/D addresses, 29
- Classless Internet Domain Routing (CIDR) notation, 30
- classtype keyword, in Snort, 199
- clearing Event logs, 231–233
- client, 40–41, 200
 - configuring with syslog-ng, 254
 - vulnerabilities, 136
- closed ports, nmap use of, 269–270
- CLOSED TCP connection state, 61, 62
- CLOSE-WAIT TCP connection state, 61
- CLOSING TCP connection state, 61
- cloud computing, 306–314
 - Infrastructure as a Service (IaaS), 306–308
 - Software as a Service (SaaS), 310–311
- cloud storage, 291
- Cogswell, Bryce, 69
- collector, configuring for logs, 252
- collision, 5, 303
 - MD5 algorithm vulnerability to, 7
- collision domain, 134
- combined log format, for web servers, 241
- Combs, Gerald, 89

- command injection, 139
- comma-separated value (CSV) format,
 - for NetFlow data, 162
- comments, in Snort, 196
- common law legal system, 4
- common log format (CLF), 240–243
- CONNECT method, for tunneling over HTTP, 287
- connect scan, 268–269
- connectionless protocol, 38, 39–40
- connection-oriented transport, 36–38
- connections, 60–62
 - programs listening for, 266
 - tearing down, 60–61
- content addressable memory (CAM), 89
- Conversations statistics (Wireshark), 261–262
- Coordinated Universal Time (UTC), 247, 250
- correlating attacks, 245–264
 - log aggregation and management, 251–257
 - Network Time Protocol (NTP), 247–248
 - packet capture times, 249–250
 - PacketTotal, 259–261
 - Plaso software, 258–259
 - Security Information and Event Management (SIEM)
 - system, 262–263
 - syslog and, 252–254
 - time synchronization, 246–248
 - timelines, 257–262
- CPU, dynamic graphs on performance, 75
- Create Custom View dialog box (Event Viewer),
 - 227, 228
- criminals, Tor and, 316
- Crocker, Steve, 20
- cron, 220
- cross-site scripting attack (XSS), 138
- CrowdStrike, 183
- cryptographic hashes, 5–7
 - table of, 205
- Cryptological Mathematics* (Lewand), 293
- cybersecurity, 10
- cyclic redundancy check (CRC), 7

D

- daemon, 55
- dark net, 292, 314
- dark web, 292
- data, 19
- Data Acquisition (DAQ) layer, in Snort, 191
- data link layer (OSI), 16–17
- data loss prevention (DLP) software, 133
- data offset, in TCP header, 34
- data theft, 133
- database programs, exporting NetFlow data to, 162
- datagram, 19
- Daubert vs. Merrell Dow Pharmaceuticals, Inc., 4
- daylight saving time (DST), 247
- decimal encoding, 136
- decryption, 292
- default gateway, 29
- deleting log files, 233
- denial of service (DDoS) attack, 8, 114–130
 - from inside network, 133
- Department of Defense (DoD) model, 18
- dependencies, for configuration file, 59
- depth keyword, 200
- DES (Data Encryption Standard), 295
- destination interface, for NetFlow data, 163–164
- destination IP address, 28
- destination port, in TCP header, 41
- Device Association Framework Provider Host, 70
- DF field, in firewall log entries, 236
- DHCP ACK message, 47
- DHCP Discovery message, 46
- Diffie-Hellman key exchange protocol, 294, 299, 303
- dig tool, 45
- digital forensics, 3
- disk, dynamic graphs on performance, 75
- dissectors, in Wireshark, 89
- distance keyword, 200
- distcc attack, 130–131, 134–135
- distributed attacks, 126–128
- DNS hostname, vs. NetBIOS name, 66
- DNS lookup, 43–44
- documentation
 - of chain of custody, 8
 - RFCs for, 20
- Domain Name System (DNS), 42–45, 125
 - query trace, 45
 - UDP transport for requests, 40
- domains, 23
- dotted quads, 28
- dp-ip.com lookup, 155
- Dropbox, 291, 309
- dropped messages, firewall log of, 175
- DSA (Digital Signature Algorithm), 302
- Dynamic Host Configuration Protocol (DHCP),
 - 32, 46–48
 - message, 87–88

E

- Elastic Stack (ELK), 219, 255
- ELK (Elastic search, Logstash, and Kibana), 255
- e-mail
 - asymmetric encryption for, 296
 - logs, 223
 - with malicious attachments, 3
- encapsulation, 15–16
 - PPTP or GRE and, 286
- encryption, 6, 108, 190, 291, 292–306
 - asymmetric, 295–296
 - hybrid, 296–297
 - keys, 293–294
 - symmetric, 294–295
- enhanced 911 (E-911) services, location information, 153–156
- Enterprise Security Manager (McAfee), 262
- ephemeral ports, 41
- Epoch time, in Wireshark, 250
- error messages
 - ICMP to transmit, 31
 - in mail system, logging, 215
- established connection, 175, 200
- ESTABLISHED TCP connection state, 61, 62
- /etc/init.d/ directory, 58
- /etc/localtime file, 145–146
- Ethernet, 20
- Ethernet frames, 83
- Ettercap, 95
- evasion, 114, 134–136
- events
 - and Bro, 203
 - vs. incident, 9
 - querying logs, 227–230
- evidence, 3
 - handling, 4–5
- Expert Information in Wireshark, 262
- Exploit DB website, 283
- extortion, 3
 - attack against password input to server, 131–132
 - Bro support for, 205
- files, gathering for packet capture, 107–108
- Filesystem Hierarchy Standard (FHS), 215
- filters
 - for log forwarding, 252
 - for logs, 228
 - for tcpdump, 88
 - for Wireshark, 101–102, 261
- FIN flag
 - nmap use of, 270
 - in TCP header, 35, 38
- fingerprints database, for operating system analysis, 271–273
- FIN-WAIT TCP connection state, 61
- Firefox web browser, 136–137
- Firepower, 191
- firewalls, 60, 127, 189, 212
 - logs, 173–177, 233–240
 - state table for, 62
- flags
 - in IP header, 27
 - nmap use of, 270
 - in TCP header, 34–35
- Flash applet, 136
- flooding, 115
- flow designation rule, 200
- foreground program, 54
- forensic practitioners, 9
- forensics, 3
 - basics, 3–8
- format string attack, 132
- Forwarded Events log, 252
- four-tuple, 62
- fping, 279–280
- fragmentation, of packets, 118
- fragmentation of packets, 134–135
- frames, 20, 82–83
- frameworks, in Bro, 204
- FRE (Federal Rules of Evidence), 4
- Frye vs. United States (1923), 4
- fully-qualified domain name (FQDN), 42
- fuzzing, 120, 139

F

- false negatives, from vulnerability scanners, 285
- false positives, from vulnerability scanners, 284–285
- Federal Rules of Evidence (FRE), 4
- Fiber Distributed Data Interface (FDDI), 203
- File Checksum Identity Verifier (FCIV), for generating
 - hash values, 7
- File Transfer Protocol (FTP), 17

G

- Galois/Counter Mode (GCM), 299
- General Routing Protocol (GRE), 286
- geolocation, 153–156

- Geolocation API, 156
- Get-Eventlog cmdlet, 228
- Get-WinEvent cmdlet, 173
- gli (get log information) command, 228
- Global Address Book (GAL), 296
- global positioning systems (GPS), 143–144
- Gnu Privacy Guard (GPG), 296
- Google, 291
 - as certificate authority, 302
 - Drive, 309
 - productivity suites, 310–311
 - transparency report, 312
- Google Rapid Response (GRR), 182, 183
- gratuitous ARP, 94
- Greenbone Security Assistant, 281
- Greenwich Mean Time (GMT), 145, 247
- Greenwich Observatory, 144
- guestbook, vulnerability, 139
- Guidance Software, 183

H

- hackers, 3, 159
 - cloud storage, 291
 - examples, 1–2, 13, 53, 81
 - backdoor, 265
 - misdirection, 113
 - time difference, 143
 - time settings, 245
 - logs and, 211
- handshake. *See also* three-way handshake
 - packet capture for, 299–300
 - for TLS, 297
- hashes, cryptographic, 5–7
- hashing algorithms, 6
- header
 - for TCP, 33–35
 - for UDP, 39
- hearsay, 5
- HeartBleed bug, 304
- heuristic detection system, 188, 189–190
- hex editor, with times from PCAP, 251
- hexadecimal dump, 90
- hexadecimal format
 - for NetFlow data, 163
 - in packet search, 201
- hop distance, 110
- host address, in IP address, 28
- host discovery, 278

- host-based firewalls, 233
- host-based IDS, vs. network-based, 190–206
- hostname, resolution to IP address, 43–44
- hosts, passively gathering data about, 287
- hosts file, 42
- host-side artifacts, 53–79
 - connections, 60–62
 - services, 54–59
 - tools, 62–79
 - /proc filesystem, 78–79
 - address resolution protocol (ARP), 77–78
 - ifconfig/ipconfig, 68–69
 - nbstat utility, 66–68
 - netstat utility, 63–66
 - ntop utility, 73–75
 - sysinternals, 69–73
 - Windows Task Manager, 75–77
- HTML encoding, 135
- hubs, 89
- hybrid encryption, 296–297
- Hypertext Transport Protocol (HTTP), 17, 287
 - applications delivered over, 313–314
 - attacks, 119–120
 - Bro support for, 205
 - timestamp in headers, 145
 - Web application firewall and, 238–240
- hypervisors, 307

I

- IBM
 - nxlog, 254–255
 - QRadar, 262
- ICMP protocol, Snort rule for, 198–199
- ID field, in firewall log entries, 236
- identification field, in IP header, 26
- ifconfig/ipconfig, 68–69
- incident, defined, 9
- incident detection systems (IDS), 159–160
- incident response, 8–10
- incident response preparation, 181–183
- incident response software, commercial offerings, 182–183
- incident response specialist, 3
- incident response team (IRT), 9, 181
- Infrastructure as a Service (IaaS), 306–308
- init program, 56, 57–58
- initial sequence number (ISN), 36
- IN/OUT field, in firewall log entries, 235

- Interface Message Processor (IMP), 20
- International Organization for Standardization (ISO), 16
- International Telephone and Telegraph Consultative Committee, 16
- Internet, cloud computing, 306–314
- Internet Assigned Numbers Authority (IANA), 41
- Internet Control Message Protocol (ICMP), 27, 31
 - Snort rule on, 199–200
- Internet Corporation for Assigned Names and Numbers (ICANN), 23
- Internet Engineering Task Force (IETF), 14, 21, 166
- Internet layer (TCP/IP), 18
- Internet Packet Exchange (IPX) protocol, 17
- Internet Protocol (IP), 14, 17, 18, 25–33
- Internet Protocol (IP) version 6 (IPv6), 31–33, 155
 - header, 26
- Internet registries, 23–25
 - location-related information on IP addresses, 147
- Internet Relay Chat (IRC) server, 126
- interprocess communications (IPC), 267
- intrusion detection systems (IDS), 187–209
 - alerts, 207–208
 - architecture, 206–207
 - Bro, 203–205
 - host-based vs. network-based, 190–206
 - log monitoring, 165
 - OSSEC, 206
 - rules, 187
 - styles, 188–190
 - Suricata and Sagan, 201–203
 - Tripwire, 205
- IP address, 28–31
 - hostname resolution to, 43–44
 - location, 147–148
 - lookups to MAC address, 77
- IP headers, 20, 25–26
 - Wireshark display, 99–100
- IP statistics view (Wireshark), 104–105
- IPSec (IP Security), 32
- iptables
 - firewall, 175
 - log entries, 174
 - port forwarding rule with, 305

J

- Java, 136
- JavaScript, for GeoIP information, 156

K

- Kali Linux system, 64
 - Plaso package on, 258–259
 - SSLScan output, 304
 - vulnerability scanner, 281
- key management, centralized vs. decentralized, 296
- keys, for encryption, 293–294
- key/value pairs, 57
- knockd program, configuration, 285
- Krebs Brian, 127

L

- laser source, 92
- latency, 76
- Latin American and Caribbean Network Information Center (LACNIC), 23
- latitude, 145
- launchd macOS startup program, 56
- law enforcement, 3
- layers, 15
- LEN field, in firewall log entries, 236
- length field, in UDP header, 39
- Lewand, Robert, *Cryptological Mathematics*, 293
- libpcap library, 191
- light emitting diode (LED) source, 92
- Lightweight Directory Access Protocol (LDAP), 300
- Link layer (TCP/IP), 18
- link-local addresses, 32
- Linux systems
 - command-line utilities, 7
 - firewall logs, 234–235
 - flow-tools package for, 161–162
 - service scripts, 58
 - services management, 57–58
 - syslog, 212–223
- listening mode
 - ports in, 65, 76
 - rsyslog configuration for, 169
- LISTENING TCP connection state, 61, 62
- local area network denial (LAND) attack, 119
- local DNS, 43–44
- localhost address, 30
- location awareness, 143–158
 - geolocation, 153–156
 - location-based services, 156
 - time zones, 144–147

- Traceroute, 150–153
- WiFi positioning, 157–158
- log files
 - deleting, 233
 - generating hash value for, 7
 - PowerShell for investigating, 228
 - querying, 227–230
 - storage in Windows, 227
 - tampering with, 216
- log management package, 178
 - vs. SIEM, 262
- log messages
 - reading, 220–222
 - routing for, 219
- log2timeline program, 246, 258
- Log::create-stream function, 204
- logging, 165–180
 - aggregation and management, 251–257
 - antivirus programs, 180–181
 - authorization messages, 220–221
 - centralized, 216–219
 - common log format (CLF), 240–243
 - firewall logs, 173–177, 233–240
 - Google Rapid Response (GRR), 182, 183
 - incident response preparation, 181–183
 - log servers and monitors, 178–180
 - mail system error messages, 215
 - management offerings, 254–257
 - monitoring for, 206
 - network traffic, and firewall overhead, 234
 - proxy logs, 236–238
 - router and switch, 177–178
 - Suricata and, 202
 - web application firewall (WAF), 238–240
 - Windows Event logs, 171–173
- LogWatch, 222–223
- longitude, 145
- loopback address, 30
- Louisiana, civil law system in, 4
- lpvar configuration keyword (Snort), 194
- Lua programming language, 273
- LUCIFER encryption cipher, 295

M

- MAC address
 - broadcast, 94
 - lookups from IP address to, 77
 - in NetworkMiner, 110
- MAC field, in firewall log entries, 236
- macOS, 55–57

- command-line utilities, 7
- launch daemons, 57
- Mafiaboy (Calce, Michael), 8, 127
- mail system, logging error messages, 215
- malformed packets, 118–122
- malicious attachments, e-mail with, 3
- malware, 180
 - and information access, 133
- man in the middle (MTM) attack, 305
- max_active_responses setting (Snort), 193
- maximum transmission unit (MTU), 26, 67
- MaxMind, 155
- Maxwell, James Clerk, 56
- Maxwell's Demon, 56
- “may,” in RFC, 21
- McAfee, Enterprise Security Manager, 262
- Media Access Control (MAC) address, 17, 18
- memory, dynamic graphs on performance, 75
- memory capture, 62
- message authentication code (MAC), 303
- Message Digest 5 (MD5), 6–7
- Metasploit, 135
 - Nexpose integration with, 284
- Microsoft
 - OneDrive, 309
 - productivity suites, 310
 - transparency report, 313
- Microsoft Excel, NetFlow output in, 163
- Microsoft Exchange Server, certificate storage, 296
- Microsoft Windows. *See* Windows
- Mirai botnet, 127
- Mixer, 127
- Modsecurity, 238–240
- monitor mode, 83
- multicasting, 129
 - Class D addresses for, 29
- multiplexing, 19
 - ports for, 40
- Multipurpose Internet Mail Extensions (MIME) type, 108
- MySQL, 162

N

- Nagios, 178, 179
- National Center for Supercomputing Applications (NCSA), 240
- National Institute of Standards and Technology (NIST), 302
- nbstat utility, 66–68

- NCSA HTTP daemon (httpd), 240
- Neighbor Discovery Protocol, 77
- Nessus Attack Scripting Language (NASL), 281
- NetBIOS name, vs. DNS hostname, 66
- NetBT (NBT), 66
- netcat program, 275
- NetControl framework, in Bro, 205
- NetFlow protocol (Cisco), 130, 160–165
 - Collector, 160–161
- Netscape, 297
- netstat utility, 63–66, 266
 - output, 64
- network address, 18, 28
- network address translation (NAT), 29, 31
- Network Basic Input Output System (NetBIOS), 66
- network communications, 212
- network devices, time synchronization with, 248
- network forensic practitioners, need for, 10–11
- network forensics, 3
- network intrusion detection systems (NIDS), 190.
 - See also* intrusion detection systems (IDS)
- network layer (OSI), 17
- network operations center (NOC), 24
- network stack, attacks on, 114
- network statistics, netstat for displaying, 64, 65
- network telescope, 128
- Network Time Protocol (NTP), 247–248
- Network Vulnerability Tests (NVTs), 281
- network-based firewalls, 233
- network-based IDS, vs. host-based, 190–206
- NetworkMiner, 82, 109–110
 - Hosts tab, 110
- networks
 - basics, 13–51
 - compromised, 2–3
 - connections, 60–62
 - dynamic graphs on performance, 75
 - forensic analysis, information loss and, 211
 - memory for information, 62
 - scanning, 265–290. *See also* port scanning
 - passive data gathering, 287–289
 - port knocking, 285–286
 - tunneling, 286–287
 - for vulnerabilities, 280–285
- Nexpose scanner (Rapid 7), 283–284
- nibble, 26
- “nice to have” feature, in RFC, 21
- nmap (network Mapper), 267–268
 - for host discovery, 278–279
 - operating system analysis, 271–273

- scripts, 273–275
- UDP scan, 270–271
- nslookup utility, 45
- ntinternals, 69
- ntop utility, 73–75
- null scan, 270
- NXLog, 218–219
- nxlog (IBM), 254–255

O

- octet, 28, 36
- offset keyword, 200
- oinkcode, 198
- one-way functions, 6
- The Onion Router (TOR), 314–317
- open ports, viewing, 266
- Open Systems Interconnect (OSI) model, 14, 16–17
- OpenVAS, 281–282
- operating system, nmap for analysis, 271–273
- Operational log (Windows), 226
- organizationally unique identifier (OUI), 17, 99
- OSSEC, 206
- OSSIM, 184–185
- outliers, in timeline, 246
- outside attacks, 3
- outsourcing, 307
- oversubscription, 93, 207

P

- p0f scanner, 96–97, 287–289
- packet, 20
 - analysis with Wireshark, 98–108
 - total length, 26
- packet capture, 3, 81–112
 - analysis, 259–261
 - ARP spoofing, 94–96
 - gathering files, 107–108
 - generating hash value for, 7
 - malformed packets, 118–122
 - NetworkMiner, 82, 109–110
 - passive scanning, 96–97
 - port spanning, 93
 - and saving, 88
 - taps, 91–93
 - tcpdump, 84–89
 - timeline information and, 246
 - times, 249–250
 - Wireshark, 89–91

- packet counts, in Wireshark, 103
- packet switching, 27
- PacketTotal, 259–261
 - Analytics page, 260
- PAM (pluggable authentication modules), 221
- passive data gathering, 287–289
- passive scanning, 96–97
- passive tap, 92–93
- pathping utility, 151
- pattern matching, 201
- pcap file format, 191
 - timestamps in Epoch time, 250–251
- Peach, 120
- percent sign (%), in URL-encoding strings, 135
- persistent XSS attack, 138
- PFSense firewall, 174–175
 - state table, 176
- PHP, 276
- physical layer (OSI), 16
- ping sweeping, 266, 278–280
- ping utility, 31, 124
- plaintext, 292
- plists, 57
- Point to Point Protocol (PPP), 20
- Point to Point Tunneling Protocol (PPTP), 286
- pointer (PTR) record, 151
- polygraph test, 4
- port forwarding rule, with iptables, 305
- port knocking, 285–286
- port mirror, 206
- port scanning, 266–280
 - banner grabbing, 275–278
 - operating system analysis, 271–273
 - ping sweeping, 278–280
 - scripts, 273–275
- port spanning, 93, 206
- Portable Document Format (PDF) reader, 136
- ports, 19, 40–42
 - listeners binding to, 60
 - in listening mode, 65
 - number display, 64
 - in rsyslog configuration, 218
 - setting Snort variable for, 195
 - for syslog, 169
 - for TCP, 33
- portvar configuration keyword, 195
- Post Office Protocol version 3 (POP3) server, 121–122
- Postel, Jon, 22, 23
- pound sign (#), for decimal encoding, 136

- PowerShell
 - for investigating log files, 228
 - to view Event Logs on Windows, 172–173
- Prelude, 185
- preparation stages, in incident response, 9
- preprocessors, in Snort, 192–198
- presentation layer (SSH), 17
- Pretty Good Privacy (PGP), 296
- Prewikka, 185
- printf function, attacks using, 132
- private addresses, 29–30, 152
- private key, 295
- privileged ports, 33
- /proc filesystem, 78–79
- Process Explorer, 72
- process identification number (PID), 70
 - netstat for displaying, 65
- processes, monitoring for, 206
- promiscuous mode, 83, 90
- property list, 57
- PROTO field, in firewall log entries, 236
- protocol analyzer, 89
- protocol data units, 19–20
- protocol field, in IP header, 27–28
- Protocol Hierarchy view, 104
- protocols, 14–20
- proxy ARP, 94
- proxy logs, 236–238
- proxy servers, 314
- PsFile, 70–72
- PSH flag, in TCP header, 35
- PsPing, 73
- public key encryption, 275, 295
- pulledpork, 198
- Python, 230
 - scripts in Plaso, 258

Q

- QRadar (IBM), 262
- queries, 137
 - of Event logs, 229–230

R

- ransom, 3
- Rapid 7, Nexpose scanner, 283–284
- raw packet data, from Wireshark, 100–101

- raw sockets, 268
- reference keyword, in Snort, 199
- referrer header, in web server log, 241, 242
- reflected attack, 138
- Regional Internet Registries (RIRs), 23
- regular expressions, 201
- Rekall (memory analysis), 182
- related connection, 175
- reliability of NTP time, 248
- remote access trojan (RAT), 133
- Remote Administration Tool (RAT), 133
- request for comments (RFCs), 14, 20–22
 - 1918 on private addresses, 30, 129
 - 3164, on syslog, 212
 - 5424, on syslog, 167–168, 212, 214
- requirements, in RFC, 21
- RES field, in firewall log entries, 236
- Reséaux IP Européens Network Coordination Centre (RIPE), 23
- Resource Monitor (Windows), 76
- reverse address, 151
- reverse path verification, 129
- risk, system deployments commonality and, 10
- Rivest, Ron, 302
- Roesch, Martin, 191
- root certificate, 302
- root name servers, 44
- rootkit, 54
 - monitoring for, 206
- rot13 encryption, 293
- rotation cipher, 292
- router, functionality of, 20
- routers
 - Bro interface with, 205
 - logs, 177–178
- routing, in NXLog, 219
- routing table, 29
 - netstat for displaying, 64
- RPaxson, Vern, 203
- RSA encryption algorithm, 302
- RST flag, 37
 - in TCP header, 35
- RST message, 268
- rsyslog, 254
 - configuration, 215
 - for receiving log messages, 216
 - for remote logging on clients, 218
 - for remote systems, 217
 - properties, 217
 - for Ubuntu Linux installation, 168–169

- rules
 - for IDS, 187, 189
 - for SIEM, 263
 - for Web application firewall, 238
- Russinovich, Mark, 69

S

- Sagan, 202–203
- Salesforce.com, 310
- Samhain, 205
- scanning. *See also* networks, scanning
 - passive, 96–97
- scientific evidence, 4
- scripts, in nmap, 273–275
- searches, in log management, 255
- second-level domains, 42
- Secure Hash Algorithm (SHA), 7, 299, 303
- Secure Hash Algorithm 1 (SHA-1) hash, 7
- Secure Shell (SSH), 17
 - tunneling with, 287
- Secure Sockets Layer (SSL), 108, 297, 298–299
- Secure Sockets (SOCKS), 287
- Security Information and Event Management (SIEM)
 - system, 183–185, 262–263
- Security log (Windows), 224
 - clearing, 231
- segment, 19
- Sendmail, 212
- sensors, in intrusion detection systems, 206
- sequence number, in TCP header, 34
- Sequenced Packet Exchange Protocol (SPX), 17
- servers, 40–41
 - centralized log, 165, 169
 - configuring with syslog-ng, 253
 - for logs, 178–180, 212
- Service Control Manager (Windows), 55
- service providers, 292
- services, 54–59, 114
- Services logs (Windows), 224, 225
- Session Initiation Protocol (SIP), 22
- session key, 297
- session layer (OSI), 17
- Session preprocessor (Snort), 192–193
- severity levels, in syslog, 167, 214
- Shamir, Adi, 302
- “should,” in RFC, 21
- signature framework, in Bro, 204–205
- signature-based detection, 188–189
- signature-based IDs, processing needs, 190

- signatures database, for vulnerability scanner, 280
- smartphone applications, global positioning systems (GPS), 143–144
- SMTP, capabilities, 277–278
- Smurf attack, 124
- snoop (Solaris), 84
- Snort, 187, 191–201
 - # for comments, 196
 - configuration, 194–198
 - preprocessors, 192–198
 - rules, 198–201
- Snort conf file, output section, 197
- Snort ID, 199
- Software as a Service (SaaS), 310–311
- source interface, for NetFlow data, 163–164
- source IP address, 28
- source port, in TCP header, 41
- Sourcefire, 191
- spam, 25
- span port, 206
- special characters, and URL, 138
- Splunk, 178, 179, 219, 256
 - creating listener, 256
 - displaying logs, 257
 - patterns, 180
- spoofing
 - addresses, 118
 - protecting against, 37
- SPT/DPT fields, in firewall log entries, 236
- SQL (Structured Query Language), 137
- SQL injection attack, 137–138
- Squid (proxy server), logs, 237–238
- SRC/DST fields, in firewall log entries, 236
- ss command, 266
- SSH server, banner from, 275
- sshd (secure shell daemon), 221
- SSLScan, 303–306
- sslstrip, 304–305
- Stacheldraht, 127
- stacks, 15
- standards, vs. RFCs, 22
- Start of Authority (SOA) record, 44
- state machine, 60
- stateful communication, 60
- stateful firewall, 233
 - SYN message in, 175
- stateless firewall, 233
- Statistics menu (Wireshark), 261
- storage space
 - for log data, 234
 - for NetFlow data, 164
- stream index, 106
- stream5_global preprocessor, 193–194
- streams, in Wireshark, 105–106
- subdomains, 43
- subnet mask, 28–29
- subscriptions for log forwarding, 252, 253
- sudo, 170, 221
 - for running nmap, 268
- superprocesses, 58
- Suricata, 187, 201–203
 - configuration fragment, 202
- Switch Port Analyzer (SPAN) port, 93
- switches, 89
 - Bro interface with, 205
 - and IDS, 206
 - logs, 177–178
- Symantec, 301, 302
- symmetric encryption, 294–295
- SYN field, in firewall log entries, 236
- SYN flag, in TCP header, 35
- SYN flood attack, 37, 115–118
- SYN message, 36, 60
 - in stateful firewall, 175
- SYN scan, 268
- SYN scan alert, 208
- SYN/ACK message, 37, 268
- synchronization
 - across time zones, 245
 - of time, 246–248
- SYN-RCVD TCP connection state, 61
- SYN-SENT TCP connection state, 61
- sysinternals, 69–73
- syslog, 166–171, 212–223
 - for centralized log system, 216
 - configuration file, 214–215
 - correlating attacks and, 252–254
 - facilities, 166–167
 - forwarding entries, 169
 - message, 220
 - severity levels, 167
- syslog-ng
 - client configuration, 254
 - server configuration, 253
- system compromise, 114, 130
- system deployments, commonality,
 - and risk, 10
- System log (Windows), 224
 - clearing, 231–233
- System Network Architecture (SNA), 16
- systemd process, 59
- Sytek, 66

T

- taps, 91–93
 - Bro configured as, 207
- TCP header, Wireshark display, 100
- TCP stream, reconstructing, 192
- tcpdump, 82, 84–89, 191
 - ARP request in, 87
 - ARP spoofing output, 94–95
 - capture with verbose setting, 85–86
 - filters, 88
 - output with time, 249–250
 - protocol decode with, 86–87
 - of SYN flood, 117
- tcpflags setting, 286
- TCP/IP protocol suite, 18–19
- TCPView, 70–71
 - Console program (TCPVCon), 70
- Teardrop attack, 118–119
- technical contact, 24
- technical intrusion, 3
- Telnet, 17
 - clients, 278
- test access point (tap), 92
- The Onion Router (TOR), 314–317
- three-way handshake, 35, 36–38, 60
 - SYN flood and, 115
 - SYN scan and, 268–269
- time, synchronization, 246–248
- time server, synchronization with, 248
- time to live (TTL), 27
 - in firewall log entries, 236
 - IP header field for, 150
- time zones, 144–147, 245, 246–247
 - Windows configuration, 146–147
- timelines, 245–246, 257–262
 - PacketTotal for generating, 260–261
 - in Wireshark, 261–262
- timestamp
 - and e-mail logs, 223
 - in syslog messages, 220
- TIME-WAIT TCP connection state, 61
- top utility, 73
- top-level domains (TLDs), 42
- TOR (The Onion Router), 314–317
- TOR browser, 292
- TOS/PREC fields, in firewall log entries, 236
- Traceroute, 150–153
- tracert utility, 151
- Transmission Control Protocol (TCP), 18, 33–38

- connections for, 60–62
- nxlog listening on, 255
- open port detection, 267–268
- ports, nmap and, 271
- Snort rule on, 199
- Transmission Control Protocol/Internet Protocol (TCP/IP), 14, 17
- transport headers, 19
- transport layer (OSI), 17
- Transport Layer Security (TLS), 108, 297
- Transport layer (TCP/IP), 19
- Tribe Flood Network (TFN), 127
- trier of facts, role of, 5
- Trin00, 127
- Tripwire, 205
- Trivial File Transfer Protocol (TFTP), 192
- trunk ports, spanning, 207
- trust, in certificate authority, 302
- tshark, 82
- TTL. *See* time to live (TTL)
- tunneling, 66, 266, 286–287
- Type of Service (ToS) byte, in IP header, 26

U

- Ubuntu Linux
 - installation with rsyslog, 168–169
 - vulnerabilities, 283
- Unicast messages, 129
- United Kingdom, common law legal system, 4
- United States Naval Research Laboratory, 314
- Unix-like operating system, 55–57
 - syslog for, 166–171
- unsolicited commercial e-mail (UCE) messages, 25
- URG flag, in TCP header, 34–35
- urgent pointer, in TCP header, 35
- URGP field, in firewall log entries, 236
- URL encoding, 135
- user agent
 - logging, 241
 - string for, 242
- User Datagram Protocol (UDP), 17, 38–40, 192
 - firewall and, 234
 - floods, 122–124
 - for NTP, 248
 - nxlog listening on, 255
 - open port detection, 267–268
 - scans, 270–271
 - source port and, 42
- UTC (Coordinated Universal Time), 247, 250

V

- variables, in Snort, 195
- Verisign, 302
- version number, in IP header, 26
- version scan, with nmap, 276–277
- virtual machine, 306–307
- virtual private networks (VPNs), 310
- Voice Over IP (VOIP), 22
 - location awareness for, 153–156
- vulnerabilities, 266
 - in client, 136
 - exploits using, 130–132
 - guestbook as, 139
 - MD5 algorithm vulnerability to collision, 7
 - scanning, 280–285

W

- web application firewall (WAF), 212
 - logs, 238–240
- web applications, 137
- web browsers, 136
- web interface, for IaaS providers, 307
- web proxy, 236
- web servers
 - combined log format for, 241
 - default port for, 136
 - HTTP message for banner, 276
 - primary index page, 276
- Webalizer, 238
- well-known ports, scanning, 269
- wevtutil program, 228–229
 - clearing log, 232
- who-has request, 94
- whois utility, 23–24, 43, 147–150
- WiFi (802.11), 20
 - positioning, 157–158
- WiGLE (wireless hotspot database), 157
- WINDOW field, in firewall log entries, 236
- window field in TCP header, 35, 38
- Windows, 7
 - Event logs, 171–173
 - APIs for interacting with, 230
 - clearing, 231–233
 - log file storage, 227
 - NetBT (NBT), 66
 - services start-up options, 54–55
 - sysinternals, 69–73
 - time zone configuration, 146–147
 - Windows Event Viewer, 212, 224–233, 251–252

- actions, 231
- applications and services, 226
- categories, 226
- creating subscription, 253
- Windows events
 - forwarding, 251–252
 - IDs, 227–228
- Windows logs, 224
- Windows registry, for configuration
 - settings, 57
- Windows Task Manager, 75–77
- Winternals, 69
- Wireshark, 82, 89–91
 - Conversations statistics, 261–262
 - Conversations view, 102–103
 - for data capture, 191
 - decode options, 106
 - Endpoints view, 102
 - Expert Information in, 262
 - exporting files, 107
 - filters using protocols, 101–102
 - GeoIP lookup with, 156
 - Name Resolution view, 99
 - packet analysis, 98–108
 - packet decoding, 98–101
 - packet capture, for handshake, 299–300
 - statistics from, 102–105
 - Statistics menu, 261
 - streams in, 105–106
 - time as offset, 250
 - Tor communication capture, 315

X

- X.509 standard, 300
- Xmas scan, 270
- XML
 - for log information storage, 171
 - for Windows logs, 224–225
- XML attacks, 139
- XPath, 229

Y

- YAML (Yet Another Markup Language), 201–202

Z

- zeroconfig addresses, 32
- zones, for domain records, 44