© SANS Institute 2021

503.5

Modern and Future Monitoring: Forensics, Analytics, and Machine Learning

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

https://t.me/learningnets

©SANS Institute 2021

© 2021 David Hoelzer. All rights reserved to David Hoelzer and/or SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE “USER”) AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE,USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, USER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree, User may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There’s an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP® and PMBOK® are registered trademarks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

https://t.me/learningnets

©SANS Institute 2021

SEC503.5 | Intrusion Detection In-Depth

SANS

Modern and Future Monitoring: Forensics, Analytics, and Machine Learning

© 2021 David Hoelzer | All Rights Reserved | Version G01_01

©SANS Institute 2021

Table of Contents | Page

Table of Contents	Page
NetFow Forencics	
➢ Network Traffic Forensics	4
➢ Cyber Kill Chain and You	12
Practical NetFlow Applications	
➢ What Is NetFlow?	16
➢ Using SiLK	23
SiLK Exercises: SiLK and NetFlow	39
Modern and Future Monitoring	
➢ Being Data Driven	40
Basic Analytics Exercise	56
➢ What is Machine Learning	57

SANS | SEC503 | Intrusion Detection: In Depth

This page intentionally left blank.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Table of Contents	Page
Learning vs. Understanding Exercise: IMDB Experiment	85
➢ Experiments with Analytics and Machine Learning	88
Is It IP? Exercise: IPv4 Binary Classification	90
Is It IP? Exercise (Take Two): Binary Classification with Better Features	93
Finding Different Exercise	96
Section 5 Bootcamp and Real-World Scenarios	101

SANS | SEC503 | Intrusion Detection: In Depth

This page intentionally left blank.

https://t.me/learningnets

©SANS Institute 2021

Section 5: Putting it all together & dealing with large scale data

- Guided scenarios to reinforce all skills
- Elaboration on effective deployment, collection, and analysis at large scale
- Moving beyond alerts to data driven analysis & detection

- Network Forensics
- Practical NetFlow Applications
- Modern and Future Monitoring

Modern and Future Monitoring

Network Traffic Forensics

Cyber Kill Chain and You

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Network Forensics Analysis

- Examination of traffic that traverses your network
- Correlation of indicators of noteworthy or anomalous activity
- For us, this will encompass:
  - Examination of Snort alerts
  - Generation and inspection of Zeek logs
  - Network flow analysis using SiLK (introduced later in this section)
  - Use of tcpdump for overview, large file inspection, and filtering packets
  - Use of Wireshark on smaller pcaps
  - Use of tshark for command-line environment
  - Inspection of host-based logs

SANS SEC503 | Intrusion Detection: In Depth 5

Most of our focus so far has been identifying signatures or interesting log entries traffic through an IDS (in our case, Snort or Zeek). This really only gives us a snapshot of an instance in time. We have already broadened our view through the use of Zeek for scripting correlations and building smarter anomaly detection, but is there more that we can do? In this book, we have two goals: draw together everything covered so far into a cohesive scenario and introduce you to the most up-to-date methods that are being researched in this field.

In the morning, we will employ Snort alerts to attempt to focus our analysis. We continue to embellish the analysis using Zeek, mostly for logs that Zeek generates, along with scripts and Zeek's file extraction capabilities. The Zeek logs contain artifacts of interest for various protocols. For instance, the **http.log** contains the request URL, method, and hostname. The **dns.log** contains DNS queries and responses in a single file of easily readable text data. Zeek offers connection data as well as the capability to correlate data in protocol logs for a given connection. We will examine the generation and inspection of Zeek logs as a means of comparing its recorded activity with that of other logs and alerts. Zeek is capable of traffic analysis in ways that other tools are not, such as default inspection of any files that traverse the network.

We'll continue to use our old favorite tools. We'll use tcpdump, mostly to get an overview of particular traffic, for large packet inspection and for filtering packets at times to a pcap that has a more manageable number of records for Wireshark inspection when the original pcap is too large for Wireshark. Wireshark distinguishes itself from all the other tools because it reassembles packets into more coherent sessions and because Wireshark has an abundance of protocol dissectors that offer us a view into the protocol fields and values.

We will place more emphasis on tshark, the command-line version of Wireshark. Tshark can do some of the same analysis as Wireshark, yet without the overhead of Wireshark's graphical interface and process-intensive rendering of protocols. Finally, we will use some logs that contain Windows Events, firewall activity, and SMTP traffic.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Before Jumping In...

When do network engineers and security people run sniffers?

- Isn't it the same answer as, "When do administrators look at logs?"

If you only ever look at your sniffer output when something's wrong...

- Grab some engineers and security analysts.
- Grab a 15-, 30-, or 60-minute packet capture from a sensor (or a live tap!).
- Book a conference room with a projector (bring bagels).

Task:

- What is it?
- Does it belong here? (If we don't know, who will find out?)
- If not, who's going to make it go away?
- Add a filter to get it out of our view.

Before we jump right into reconstructing events, we'd like to give you a suggestion that is directly related to raising the awareness of network engineers and security analysts. If you think about it, we only tend to run sniffers in the same way that administrators look at logs; when there's something wrong. There is something fundamentally wrong with this model! If you only look when there's something wrong, you have no baseline, no context, no notion of what "normal" looks like for your network!

To change this (and dramatically improve your ability to identify "wrong" when investigating an incident), we want to recommend a useful process that should be done every quarter or so. Book a conference room and schedule a meeting with your network engineering staff and IDS analysts. Make sure that the meeting room has a projector. Prior to the meeting, grab a reasonable snapshot of packets off a backbone or segment within your network. You want it to be large enough that it is a good representation of "normal" and small enough that you can filter it efficiently on your system.

During the meeting, hook your laptop up to the projector and open up the packet capture. The group's task is to start with the first packet and figure out what it is and whether or not it belongs on your network. If your team can identify it and you know it belongs, add a display filter to eliminate packets of that type. For example, you see a DNS request heading to your internal name server; reasonably, you would create a filter that eliminates *all* DNS traffic going to and from your internal name server.

If your group runs into something that it cannot identify easily or the purpose of which is unknown, someone at the meeting must take an action item to go and find out what it is and why it's there. If it's needed, get it documented. If it's not needed, make it go away. In the meantime, once it's assigned, move past it in the capture.

In the end, the only packets displayed should be the ones assigned as action items. Save these packets into a file; you will start the next meeting of this type with those packets to make sure they have been dealt with.

If you did this quarterly, how well would your team get to "know your network?"

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Indication of an Issue

Possible ways to learn of an issue:

- IDS/IPS alert
- Call from the help desk
- Firewall/server/syslog logs
- SIM correlation alert
- Antivirus host issue
- Receive indication from another network that your network is attacking them
- Agency/corporate bulletin—that is, warning of a phishing attack
- Twitter/internet exposure
- Unusually high/low network throughput
- Visualization anomaly like a spike or dip of normal behavior

The first phase of any network forensics analysis is learning of some kind of issue. There are many ways for this to occur. You may learn of a possible incident via an IDS/IPS alert. Firewall or server logs can supply indications of suspicious traffic or even content for something more specific, such as a web server log. If you've implemented a syslog server that collects data from different hosts, it may expose details of one or more host-based events.

Security information/event management software and devices are meant to correlate input from multiple different sources to determine if there are events of interest on the network. And although not part of network forensics, antivirus can indicate that a particular host has been infected. This may warrant further scrutiny of network traffic to and from the host to see if there are concurrent issues with the host, possibly exfiltration of data.

You can also learn of issues with your network from another network that yours may be attacking. In addition, perhaps your entire company or agency is a target of some kind of attack, such as a phishing campaign. You may learn about this because affiliate branches or networks of your agency have received a similar attack.

You may learn of security issues in a roundabout way if they are somehow associated with network performance or flow. If there is a known baseline of performance and known baseline of average flow, it is much easier to determine when there is some kind of anomaly. Poor network throughput could be a result of some security incident that warrants investigation. Finally, if you use some kind of visualization tool, an anomaly of some sort, such as a spike or dip of normal behavior patterns, may provide an empirical indication of an issue.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Forensic Analysis

What resources do you have available to investigate an issue?

- Full-packet capture
- Network flow
- A combination of both
- Neither
- Log files from hosts/network traffic

When performing network forensics, you are typically looking at something that has already occurred and requires some kind of retrospective analysis. This means that captured data must be at your disposal or there can be no investigation.

The best type of captured data contains the entire packet, including the payload. However, many sites cannot save this data or retain it for long periods of time just because it is so voluminous. Another type of capture is something such as SiLK or NetFlow data that retains metadata about traffic flows, with pertinent data such as IP addresses, port numbers, and time, and may include byte or packet transfer numbers. Although not as comprehensive as full-packet captures, it can supplement full-packet capture to provide indications of traffic flow, acting like an index or starting point into a more defined investigation. In addition, because there is far less data, it can often be retained for a longer period of time.

Ideally, both full-packet and flow data are available. In this situation, the flow data can validate that traffic from an indicator did occur and give more precise times and communicating IP addresses/ports. This may be used to examine full-packet capture for those exchanges.

And don't forget about the value of log files from the host or from network devices such as firewalls. They can provide additional data and corroborate existing data from other sources.

If you do not retain any network data, you are at the mercy of the alert-driven sensor, such as Snort, to warn you of possible malicious activity. However, you have no way to retrospectively analyze what transpired.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Data Collection Challenges

Even if you collect traffic, challenges may be present:

- Short data retention period
- Full-packet capture of event of interest?
- Data collection on network of interest?
- Encrypted traffic
- Nonstandard port usage/tunneling
- NAT/DHCP attribution issue
- Risk introduced by storing data

Even if you have been diligent about using a variety of different sensors, both alert and data-driven on your network, there may still be challenges that either hamper or preclude network forensic analysis of a particular incident. There may be a short retention period of data, especially for full-capture data. The data associated with the incident may be long gone. Also, some of the investigations, such as a phishing attack, may require full-packet capture and you may not collect it.

It's also possible that your sensors may not be in a location that captures the traffic you need. Many large sites have sensors closer to the perimeter and may miss intranet exchanges. Also, if you need to look at payload content and the network transfers are encrypted, you cannot see the content. And sometimes, an alert-driven sensor such as Snort may be configured to look for content on a standard port. So, if a nonstandard one is used or some kind of tunneling is used, it is possible that you may not see alerts that may be present in the traffic.

Even though you may have some kind of indicator that specifies an IP address of interest, you may be looking at a NAT translation by some router. Or your network may use DHCP, and the IP address is assigned to a different host. These are attribution issues; you found the traffic—you just may not know its true origin.

It is ironic to think that the same data that may indirectly be helpful in protecting your network may also create more risk for the site that stores it. You must protect the stored data against attackers, both inside and outside threats. Also, consider that the possession of such data may be subject to subpoena if law enforcement sees value in it for investigating a breach and subsequently finds your site liable for faulty or noncompliant practices. The point is that you need to be aware of the benefits and disadvantages of storing data.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Investigations Using Network Forensics

Exact methodology of network forensics analysis is based on:

- The types of data available
- The incident type
- The incident details

- Iterative process
  - No precise formula for methodology
  - Exposed indicators may direct your analysis.

There is no one-size-fits-all standard methodology for network forensics analysis. You have to approach the analysis depending on the type of data you have available. Do you have full-packet capture, or do you have network flow? Is the data you need still available?

Also, the method you select to investigate the issue is going to be different if it concerns looking for malicious content, such as an infected host, versus an indicator of throughput problems between certain times.

A final influence on your investigation includes the details available about the incident from the original indicator. The more details you have, the more likely you are to have a focused investigation. If the indicators are vague, you may need to be resourceful, and the investigation may progress in phases in which you become more focused as you learn more from all your sources.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Perform Scenario-Based Network Forensics Analysis

You will be performing a forensic analysis momentarily.

You will have pcaps, logs, and tools available for analysis.

The scenario will be approached in terms of phases of exploitation.

- These map directly to the Cyber Kill Chain.
- There are extra scenarios at the very end of the labs if you would like to test yourself further.
  - We strongly recommend that you complete these labs during your certification exam preparation!
  - You might start them during the bootcamp if you are in a live class.

Let's set the stage for the agenda. Unlike many of the previous sections, this section will contain less instructor teaching and more student analysis. The intention is for you to learn by doing analysis rather than listening to an instructor talk about how to do analysis. You will examine data associated with three different incident attack scenarios.

The idea is to simulate what you might encounter in your work as an analyst. There is an incident of some sort, and you are asked to investigate what happened. At your disposal are packets in a capture pcap, a variety of logs, and the tools you've used thus far, as well as some new ones.

We will approach each scenario in terms of phases of exploitation, much like how incident handlers approach their investigation.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Sections 1 & 2: Packets/Protocols as a Second Language	Section 3: Signature-Based Detection	Section 4: Anomalies and Behaviors	Section 5: Putting it all together & dealing with large scale data	Section 6: "Ride Along" Analysis
• An immersion class in speaking packets, protocols, and hex • Ethernet • IPv4 & IPv6 • TCP, UDP, & ICMP • Wireshark & tcpdump fundamentals, filtering, & investigation	• Architecting for massive packet capture & analysis • Scapy for building tools and simulating signatures & behaviors • Researching common application protocols • Snort/FirePOWER & Suricata	• Overcoming the limitations of signature-based tools • Basic Zeek usage and logs • Threat modelling • Advanced correlation using Zeek	• Guided scenarios to reinforce all skills • Elaboration on effective deployment, collection, and analysis at large scale • Moving beyond alerts to data driven analysis & detection	• Challenge questions guide/follow expert analysts' progression through real-world data

Bootcamp (Sections 1-5): Real-World Application of Theory

- <u>Network Forensics</u>
- Practical NetFlow Applications
- Modern and Future Monitoring

Modern and Future Monitoring

Network Traffic Forensics

Cyber Kill Chain and You

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

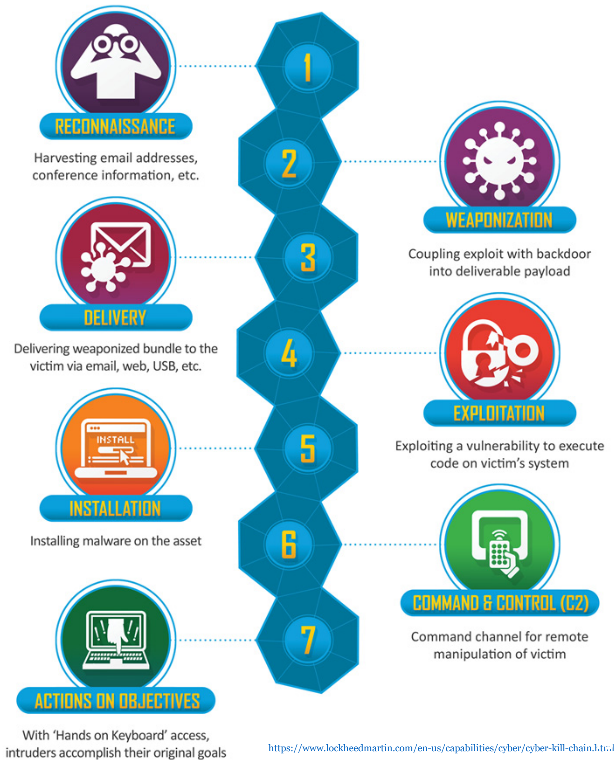
Lockheed Martin Cyber Kill Chain®

A thoughtful time-oriented description of the phases that attackers must progress through

- Not all attackers are that thoughtful…
  - For example, reconnaissance could effectively be a "drive by."
  - Not all attackers have well-defined goals.

This course is somewhat unique…

- We touch every aspect of this Cyber Kill Chain that involves observable activities.



https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

SANS

The Lockheed Martin Cyber Kill Chain is not unique in its depiction of attacker methodologies, nor is it particularly new. It is just one of the more common and current ways that attacker activities are being described. If you meditate on the various aspects of the kill chain as presented and how this course maps to that kill chain, you will come to the realization that the techniques and skills that you have learned in this class cover every observable aspect of the kill chain.

For example, reconnaissance that is in any way connected to packets entering our networks or monitored systems would have been collected if we were capturing all packets. Even if these did not trigger an alert, we might be able to correlate them with activities occurring later in the chain. Weaponization is entirely out of our view since that happens at the attacker's system, but attempts at delivery, some of which might fail, would also leave packet evidence. When the delivery turns into successful exploitation, this too will leave digital footprints on our network. As the attacker operates remotely to install additional stages of malware or other tools onto hosts, all this activity will leave network traces, even if they are encrypted. When the attacker begins to operate a C2 channel, this will almost certainly leave network fingerprints unless it is entirely out-of-band (not on the network at all). When that C2 channel is used to perform actions on objectives, all of this has the very real potential to leave tracks in our network data.

The key, then, is the ability to reconstruct what has occurred based on the network and system traces that have been left behind. The beautiful thing about using packets for this is that even diskless malware must at some point pass over the network, leaving behind packets!

©SANS Institute 2021

Phases of Exploitation

Pre-exploitation
- Reconnaissance

Exploitation
- Successful penetration

Post-exploitation
- Maintaining access
- Privilege elevation
- Internal network reconnaissance
- Lateral movement
- Goal achieved

Exploitation can be divided into three separate phases, including pre-exploitation, which occurs for the purpose of gathering data about the target site or host; the exploitation itself; and the activity that occurs after exploitation that is meant to achieve the attacker's goal.

Pre-exploitation involves reconnaissance to gather data about the site, such as hosts in the network and listening ports. Additionally, an attacker may connect to the target site's web server(s) to gather employee names, positions, and email addresses. This traffic may be detected by the monitored site's analysis (depending on the volume of it), packet capture, and sensor placement. Reconnaissance may be sought from social networking sites such as LinkedIn for employee positions, user forums for software questions by the target sites' employees to gather data on supported products, or search engine queries.

Exploitation occurs when an attack is successful, giving the attacker access on a host on the target network. There are many ways for this to occur, including social engineering attacks such as phishing and opening malicious documents or attacking vulnerabilities that have not been patched. One of the most useful tools for penetration is Metasploit, which we will encounter in some of the attack scenarios in this section’s exercises. It is easy enough for a novice or inexperienced attacker to use successfully.

The attacker must maintain some kind of access to achieve the intended goal and/or to maintain a long-term presence. One means of accomplishing this is by creating connections that originate from inside the firewall to bypass firewall restrictions. Also, a backdoor can be installed for future access, or covert communication can be established using command and control channels.

Ultimately, an attacker would like privileged access by becoming the administrator for Windows hosts or domains and a superuser for UNIX-like hosts. Depending upon the method of attack and the privilege level of the attacked user or process, the attacker may or may not have the highest access desired. Metasploit has the capability of performing a series of attempts to gain administrator access via the use of its Meterpreter functionality with the express purpose of facilitating post-exploitation activity.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

From Packets to Timeline

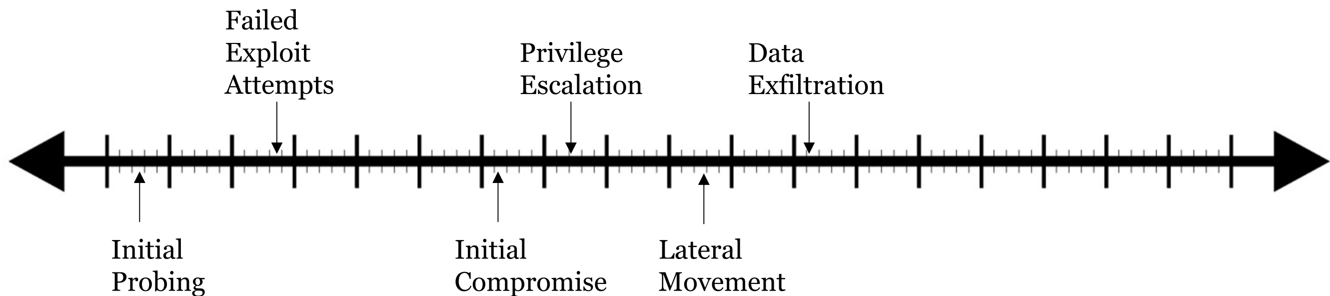


Figure out what you know about when things happened and what else you need to know.

Very early in the class, we showed you a similar timeline. Thinking of things in terms of a timeline or even building a timeline of the incident that you are analyzing is *incredibly* helpful. As you analyze packets and reconstruct events, try to figure out where on the timeline you are. Based on that, develop theories of what things should have come before or might have come afterward. This can help you to know what it is that you are looking for as you sift through the packets!

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Section 5: Putting it all together & dealing with large scale data
- Guided scenarios to reinforce all skills
- Elaboration on effective deployment, collection, and analysis at large scale
- Moving beyond alerts to data driven analysis & detection

- Network Forensics
- Practical NetFlow Applications
- Modern and Future Monitoring

Modern and Future Monitoring

What Is NetFlow?

Using SiLK

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Network Flows

- NetFlow/IPFIX
  - Built into enterprise switches/routers/firewalls – *you should have this configured!*
  - NetFlow was originally a Cisco product that has emerged as an IETF standard.
  - NetFlow versions 5, 7, and 9 are the most common.
  - IPFIX is sometimes viewed as NetFlow v10.
    - Intentional step away from being Cisco driven
- sFlow/jFlow
  - www.sflow.org – Industry standard for sampling flows
    - Statistically, samples flow data periodically; much less granularity
  - jFlow is Juniper's implementation of sFlow

Thus far in this course, we have been very packet focused. We did step away from this a bit when we worked our way through Zeek. With Zeek, we tended to be far more session and event oriented. Zeek logs, while full of wonderful information, present far less data than is actually present in the network communication.

Network flow analysis moves back even further. Network flow data, which is most commonly recorded using NetFlow or IPFIX formats, records absolutely no packet content. Instead, it generates metadata records that describe which host connected to which other host using which protocol, which ports, keeping track of how many bytes went by, how many packets there were, what the aggregate TCP flags were, and more.

When we talk about setting up full packet captures, students will often balk. Even though you can readily appreciate the tremendous value of having all of the packets, there is clearly a cost associated with capturing them all. This is why we suggest starting by collecting everything at your egress/ingress point first to see how much value it adds for you and to allow for better investigations. When it comes to NetFlow, however, things are very different. ***This is the one single thing that every enterprise network should have configured***. In fact, there's no reason not to. Your devices, almost without exception, *already support it*. It's simply a matter of building a collector and turning it on.

NetFlow was originally part of the Cisco line of products, but some years ago it became an industry standard, and it is now under the control of an IETF working group. While NetFlow versions 5 and 7 are very common to find, since our switching and routing hardware often stays in use for many years, version 9 is the most current format. This version has support for logging things like MAC addresses, VLANs, and other fields, which may or may not be populated, depending on how you've configured your receiver. IPFIX could be viewed as more of a "fork" of NetFlow than as a new protocol; in fact, some in the industry refer to it as NetFlow v10. It adds additional field types and more flexible field definitions.

An entirely different approach is to collect only statistical information about flows that are passing by. This is what sFlow is all about. Obviously, this will be far less granular since it is not making an effort to record information about every flow, but only aggregate statistics about flows. jFlow is Juniper's implementation of sFlow.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

NetFlow?

An untapped source of intelligence you've already paid for!

- You must configure it and store it somewhere… Has that even been done?
- If you are, it seems almost no one is looking at it for anything beyond statistics.
  - There are great (expensive) visualization tools for NetFlow. Most use it for health monitoring.
- Switch/router firmware can be buggy and inconsistent in reporting flows.

Is there a way to "normalize" all of this?

We believe that NetFlow data is incredibly valuable. We've also come to the realization that the network engineering community at large seems apathetic about NetFlow. This is very unfortunate since it is one of the most valuable sources of rich, long-term data about your network… and you've already paid for it! Every enterprise class network solution (routers, switches, firewalls, and others) has NetFlow support built in (though, for some, we may need to pay some extra licensing fees or even add management cards to a Cisco). All that we must do is create a repository to store the data in and configure our devices to send the data there!

Our experience tells us, however, that the NetFlow implementations in various switches, routers, and firewalls can be somewhat buggy. For example, we have seen Cisco devices that fail to log critical NetFlow fields, even when they are configured to do so. We have Avaya devices that will randomly generate flows that appear to be in the far future or in the distant past. We have seen other vendors' devices generate flows that claim to transfer terabytes of data in seconds over gigabit links. All these things can call into question the overall accuracy of your NetFlow data.

Is there a way to leverage NetFlow and correct for these issues? Yes!

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

SiLK – System for Internet Level Knowledge

Free NetFlow implementation from CMU (cert.org)

- Provides a repository that can accept any version of NetFlow or IPFIX
- Can accept NetFlow data from any vendor
- Can generate NetFlow records from network data directly!

Provides tools to manipulate, search, and output flows

- Facilitates iterative and progressive analysis of data by piping results between a suite of commands
- Perfect for subnets where we need intra-host communication data
- Provides forensics for pre-/post-incident activity
- Supplements signature-based detection

SiLK is free tool from Carnegie-Mellon University's CERT that can be used as a repository for NetFlow data from any source that you have in your network, a NetFlow sensor that can convert network packets into NetFlow data in real time, and a suite of analysis tools for manipulating and reporting on the data in the repository. SiLK allows you to store massive amounts of data in a repository using an efficient packed binary format. The repository location is selected by the administrator upon installation and represents a directory and subdirectories that store the collected NetFlow data.

We can use SiLK to perform network behavioral analysis. Intrusion detection and prevention systems are not particularly adept, nor should they be, at recognizing or analyzing network flow anomalies. They mostly concentrate on finding attacks by assessing the payloads sent. Although some have the capability to look for network anomalies such as SYN floods or port scans, these are processes best left for network behavioral analysis, such as finding the top 10 talkers per hour by packets, bytes, and flows sent or received. Network behavioral analysis relies on summary data and not a particular individual packet or payload.

SiLK (or NetFlow, generally) is not limited to performing behavioral analysis. NetFlow data is the perfect solution for networks where we are unable to collect all of the packets. We suggested in Book 4 that we might not have visibility of intra-subnet communication (i.e., communication between hosts connected to the same switch) since our tap would typically be connected between the upstream port and the router. What if a host connected to that switch becomes compromised? While we can use our packet data to determine if that host communicated across the router after the compromise, how can I easily determine if there was lateral movement to other systems on that same subnet? While I won't have the packets, the NetFlow data can tell me almost instantly if that host communicated with any other hosts on the subnet within any given time period.

All of this can be accomplished using a suite of command-line tools that make up the SiLK suite of tools. We will explore some of these tools in this book and in the accompanying exercises.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

What Is a Flow?

- Summary of unidirectional traffic that shares a 5-tuple of:
  - Source IP
  - Destination IP
  - Source port
  - Destination port
  - Protocol
- Data associated with a flow includes:
  - Above 5-tuple
  - TCP flags
  - Total bytes and packets
  - Start time, end time, duration, acquiring sensor identification

The recording unit for SiLK is known as a *flow record*. The SiLK flow record is based on NetFlow but may record additional items as well, supporting all versions of NetFlow and IPFIX, including custom record types.

A flow record summarizes the traffic between given source and destination IPs, ports, and the protocol. The notion of ports may vanish if not supported by the protocol. For example, if we are looking at ICMP flows, the source port field will contain the ICMP type, and the destination port field will contain the ICMP code.

An important concept to grasp with NetFlow when compared to the packet data we are accustomed to is that a flow record is *unidirectional*. Consider a TCP session, for instance. Two flows are created for this session: one from client to server and another from server to client.

A flow record is stored with a 5-tuple used to distinguish the flow—source/destination IPs, the ports, and the protocol—along with all TCP flags, if any, used in the flow, the aggregated number of bytes and packets for the flow, a start and end time, and the duration. Additional support is available for things like application identification, geographic localization, and others. There is some metadata in each record that identifies the sensor name that collected it.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

NetFlow and SiLK Peculiarities

- A refresh interval must be configured.
  - 10 minutes seems to be common, but there is no standard.
  - Every interval, sensors flush data about *every* connection to the repository
    - Connections crossing interval boundaries will appear in multiple intervals.
    - We must stitch these together to determine aggregate data using 5-tuple.
- The repository has to live somewhere.
  - Hierarchical directory structure
    - Used for “partitioning” criteria
      - Sensor, type of traffic (in, out, etc.), time interval

SANS SEC503 | Intrusion Detection: In Depth 21

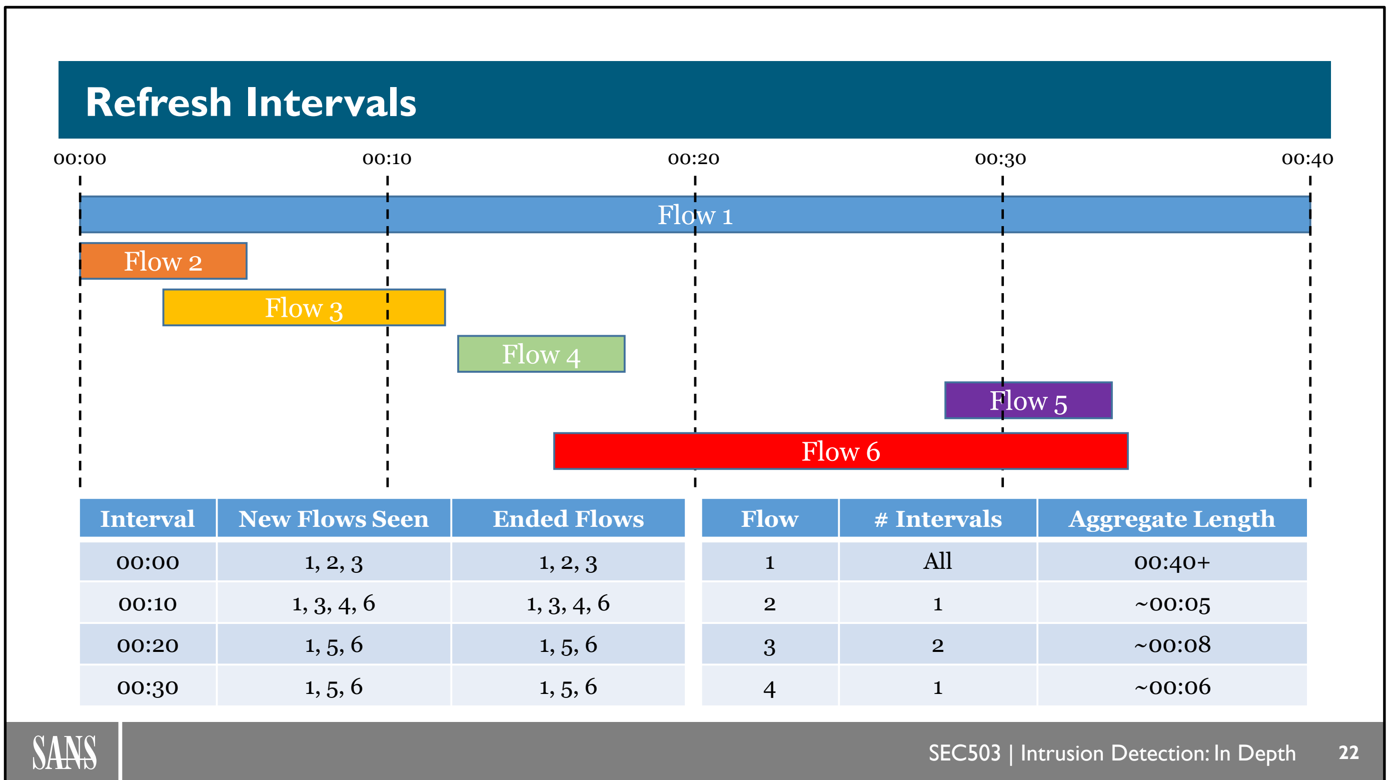
NetFlow and SiLK have a peculiarity that you need to be aware of to properly interpret the data. A refresh interval must be configured on the sensor or probe that is generating the NetFlow data sent to the repository. There is no standard for this time interval, and each sensor can have a different interval configured. Ten minutes seems common, though we have seen intervals set as high as 30 minutes. An interval this high can create some spiky loads on busy networks since all the metadata about every connection that occurred in the interval must be streamed to the server at the end of the interval. The longer the interval, the more data that is streamed.

Regardless of the interval used, whenever the interval elapses, all the data about all current (and ended) flows from that interval are sent to the repository as an update. Following the update, the sensor will flush all this data out of its memory. This brings us to something that might seem peculiar. From NetFlow’s point of view, all sessions end when the interval boundary is crossed. This isn’t how network protocols work, however. Therefore, the very next time that a packet from an existing sensor passes, the sensor will note it as a new connection! How can we stitch these together? Primarily through the 5-tuple that is used to identify each flow. Since some protocols could reuse port pairs, we might need to look at things temporarily to disambiguate connections, or, using a protocol such as TCP, we can look at the flags that were seen in the interval.

SiLK is an excellent choice as a repository solution. Not only is it free, but it supports all versions of IPFIX and NetFlow that are currently in use today.

https://t.me/learningnets

©SANS Institute 2021



The notion of how flows cross refresh intervals often confuses people at first. We are used to thinking about network connections in terms of when they start and end based on when the communication first begins and when that communication ends. NetFlow views things differently. Instead, NetFlow is effectively taking snapshots of the network connections over time and reporting that data at the end of a snapshot (or refresh) interval.

Think of it this way. We are setting up a camera with a timed exposure. Our camera will open its shutter for ten minutes at a time. Everything that passes in front of that lens in that time will be captured, then the shutter closes. Our camera, however, is set up to automatically take a series of ten-minute exposures.

Using the camera analogy, and imagining that we can make sense of the image created, looking at one of the snapshots we might know that a dog was present during the interval. Was the dog already here when the shutter opened? We can't tell. Did the dog wander off immediately after this shutter closed? We have no idea. The only way to answer those questions would be to look at the refresh intervals to the left and right of the current interval.

This is precisely how NetFlow data is captured. When a refresh interval starts, the device's memory of NetFlow data is flushed, so we are starting fresh during every interval. When the very first packet is seen for a connection during that refresh interval, it will be recorded as a brand-new flow. For the duration of that interval, related information about packets in the same direction will be aggregated to that flow. At the end of the interval, everything that the sensor has collected will be sent to the repository, and the memory will be flushed again.

This means that if we wish to determine how long a connection existed, we need to find all of the flows related to that connection, even if they occurred across multiple time intervals.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Sections 1 & 2: Packets/Protocols as a Second Language	Section 3: Signature-Based Detection	Section 4: Anomalies and Behaviors	Section 5: Putting it all together & dealing with large scale data	Section 6: "Ride Along" Analysis
• An immersion class in speaking packets, protocols, and hex • Ethernet • IPv4 & IPv6 • TCP, UDP, & ICMP • Wireshark & tcpdump fundamentals, filtering, & investigation	• Architecting for massive packet capture & analysis • Scapy for building tools and simulating signatures & behaviors • Researching common application protocols • Snort/FirePOWER & Suricata	• Overcoming the limitations of signature-based tools • Basic Zeek usage and logs • Threat modelling • Advanced correlation using Zeek	• Guided scenarios to reinforce all skills • Elaboration on effective deployment, collection, and analysis at large scale • Moving beyond alerts to data driven analysis & detection	• Challenge questions guide/follow expert analysts' progression through real-world data

Bootcamp (Sections 1-5): Real-World Application of Theory

- Network Forensics
- Practical NetFlow Applications
- Modern and Future Monitoring

Modern and Future Monitoring

What Is NetFlow?

Using SiLK

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

rwfilter

- Workhorse of SiLK tool suite
- Input consists of binary efficiently packed flow records
- Default output is binary format
- Processes flow records for specified characteristics
- SiLK tools maintain the data as binary records in memory to allow for easy and efficient piping between tools.

We are going to spend some time reviewing the **rwfilter** command and the associated syntax and processing because it's the workhorse of the SiLK tool suite. **Rwfilter** is typically the tool you can use to extract the flow records of interest to you. You can think of it as a means of filtering, kind of like BPF only more comprehensive and easier to use. Most of the other tools in the suite take their input from **rwfilter** output for further processing.

The default input to **rwfilter** is flow data from the repository. You can specify your own flow data, or **rwfilter** can accept binary flow output from other **rwfilter** commands as input data. The default is to maintain this same binary flow output.

Because SiLK was written to store massive amounts of data efficiently, it tries to leave the data in binary format and in memory as long as possible to maintain that efficiency. Therefore, the **rwfilter** command has a default mode of preserving the binary format unless you explicitly pipe the output to another command that processes it and formats it as ASCII.

Rwfilter uses many different parameters to specify the type of traffic you'd like to process in terms of direction into or out of the network, a particular sensor from which you'd like to extract the data, and many other different selection criteria, including but not limited to source and destination IPs and ports, time of capture, duration, number of packets/bytes, and protocol.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Wrapping Your Head Around NetFlow

NetFlow processing is extremely different from packets!

- Records are unidirectional.
- There are no packets.
- There is no payload.
- There is a summary of sessions/conversations only.
- Determining what happened may require us to stitch together records from multiple refresh intervals.

If you try to use SiLK tools (especially the workhorse command **rwfilter**) like tcpdump, you'll get frustrated. Not only is the data very different from that to which we have become accustomed, but there are inconsistencies within the tool suite and command-line options. This means that, more than any tool we've examined in this course, these tools will frequently require you to consult the manual pages to figure out how to get the output you need.

There are a few important things to keep in your mind as you review NetFlow data:

- Always try to remember that each flow record is unidirectional. This means that a ping and the corresponding response will be recorded in two separate flows. The same is true of a web request; there will be a flow from the client to the server and a separate flow from the server to the client. This is the thing that most tends to confuse people at first.
- There are no packets. If we had the packets, we might use them directly, but we are using NetFlow, perhaps because there are no packets to look at. (More on this later… there's a reason we might use NetFlow over packets even when we have the packets.)
- Since there are no packets, there is also no payload available. This means that we must use our expertise in how network and application protocols function to make an educated guess as to what is happening in a flow.
- What we are viewing in the NetFlow data is effectively an aggregated summary.
- Since flows are distinct from the connections, we have to remember that the connection may exist across multiple refresh intervals. This will require us to find all the related flows and view them together to understand the overall conversation.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Using rwfilter

Most pipelines will begin with this command.

- First, decide where the data is coming from.
  - The repository is the default unless you tell it otherwise.
  - You can also specify a SiLK format file containing NetFlow records.
  - You can even chain together rwfilter commands, so the pipeline could be the source.
- Specify selection criteria.
  - Which sensor? What type of flow? What time frame?
  - Effectively, this translates into *which files in the repository?*
- Specify a partitioning criteria.
  - Which hosts, protocols, ports, etc.
  - If you want everything, `--proto=0-255` is a quick and easy shortcut.

The primary tool that we use to interact with SiLK is **rwfilter**. When using this tool, we must decide where our data is coming from first. The default source of data will be the repository. You do not need to do anything special to retrieve data from the repository; rwfilter knows how to do this automatically. You can, however, also specify a SiLK-formatted NetFlow data file as the source. We'll look at doing this in a couple of pages. You might even chain together several filtering commands, so rwfilter will allow you to use the pipeline as an input data source.

We must also specify what SiLK terms selection criteria. This ends up translating to the set of files in the repository that SiLK will consult. The files are organized by the type or directionality of the communication (next slide), the sensor that collected them, and the time periods or refresh intervals that they cover.

Finally, we must also specify a "partitioning" or filtering criteria. This defines the hosts, protocols, networks, ports, flags, or any other piece of data that is stored within a NetFlow record as the search criteria for reporting that flow in the output. When we are first looking at our data, it is common to want to see everything and to then narrow that view down. An easy way to see all the flows is to specify `--proto=0-255`, which asks for all IP protocols.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Which Data?

SiLK automatically partitions data based on direction.

Direction Name	Meaning	Direction Name	Meaning
`in`	Inbound	`int2int`	Internal to Internal
`inweb`	Inbound HTTP	`out`	Outbound
`outweb`	Outbound HTTP	`ext2ext`	External to External
`inicmp`	Inound ICMP	`outicmp`	Outbound ICMP
`other`	Other	`innull/outnull`	Blocked In/Outbound

Defining other types requires source code modifications.

One of the partitioning criteria that we can define is the type of traffic, or the directionality from SiLK's point of view. There are ten predefined classes that are used by default, as defined in the slide.

How does SiLK know if something is inbound or outbound? Part of the configuration of SiLK is to define which network address ranges are considered internal and which are considered external. This is configured in `/data/sensors.conf`. SiLK uses the configuration information there to decide whether something is inbound or outbound.

Some of these types are special, however. For example, `innull` and `outnull` are not values that SiLK will usually assign if we are using the accompanying flowmeter software to generate NetFlow data. These types are usually limited to firewalls and routers being used as NetFlow data sources. They indicate that a flow was seen attempting to come inbound, for example, but the packets were denied or dropped.

It is possible to define additional types, and we can easily do so within the main SiLK configuration file. However, for those types to be used, we would have to modify the SiLK source code.

We find that we usually want to look at all the SiLK data, regardless of direction, in our general queries. An easy way to do this is to define `--type=all`.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Using a Repository: The Most Typical Use

```
sans@sec503:~$ rwfilter --type=all --proto=17 --start-date=2018/10/01 --end-date=2018/11/01 --
pass=stdout | rwcut | more
                                    sIP|                                    dIP|sPort|dPort|pro
|   packets|     bytes|
 flags|                 sTime| duration|                 eTime|   sensor|
                            192.168.2.35|                           74.117.214.3|  123|  123|
17|         1|        94|
      |2018/10/01T01:01:30.486|2162833.860|2018/10/26T01:48:44.346|      ERS|
                            192.168.2.35|                           74.117.214.3|  123|  123|
17|         1|        94|
      |2018/10/01T01:01:32.486|2192465.290|2018/10/26T10:02:37.776|      ERS|
                            192.168.2.35|                           74.117.214.3|  123|  123|
17|         1|        94|
      |2018/10/01T01:01:32.486|2244581.300|2018/10/27T00:31:13.786|      ERS|
                            192.168.2.35|                           74.117.214.3|  123|  123|
17|         1|        94|
      |2018/10/01T01:01:33.486|2285626.520|2018/10/27T11:55:20.006|      ERS|
```


In practice, NetFlow data should be stored in a repository, and the repository should be receiving NetFlow data from the sensors (your routers, switches, firewalls, etc.) throughout the network continually. In this way, you can use the **rwfilter** command to query records directly out of the repository with no fumbling around to find files.

In the slide, you can see that we have told SiLK that we are interested in **--type=all** traffic, or all traffic, regardless of type or directionality. We then used a selection criteria using **--proto=17**, which instructs SiLK to select only UDP protocol packets. We also asked SiLK to show us flows from between October and November of 2018. Lastly, we tell the **rwfilter** tool to **--pass=stdout**, or send all flows matching this criteria to standard output.

In our case, we requested data from a specific time frame. How would we know which time frame to use? That would be driven by whatever has motivated you to look at your NetFlow data. Was there some indication or other warning that brought you here? Or are you here because you are doing some threat hunting, just looking for interesting or unusual patterns?

If you use **rwfilter** without specifying a start or end time, SiLK will automatically show you data from the last hour from your repository. If you specify a start time, it will show you one hour from that starting date or time. If you specify an ending date, it will show you the last hour of that date.

The **rwcut** tool is one of the tools we can use to process flow data. This tool will print the flow data in human readable format for you. With no options, it will print the default set of fields using tabs and vertical pipe signs as separators.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Making Things More Readable

```
sans@sec503:~$ rwfilter --type=all --proto=17 --start-date=2018/10/01 --end-date=2018/11/01 --
pass=stdout | rwcut --fields sip,sport,dip,dport,bytes | more
                                     sIP|sPort|                                    dIP|dPort|     bytes|
                            192.168.2.35|  123|                            74.117.214.3|  123|        94|
                            192.168.2.35|  123|                            74.117.214.3|  123|        94|
                            192.168.2.35|  123|                            74.117.214.3|  123|        94|
                            192.168.2.35|  123|                            74.117.214.3|  123|        94|
                            192.168.2.35|  123|                            74.117.214.3|  123|        94|
                            192.168.2.35|  123|                            74.117.214.3|  123|        94|
                            192.168.2.35|  123|                            74.117.214.3|  123|        94|
```


With the **rwcut** tool, we can also specify the fields that we wish to see. In this example, we have limited the output to only the source IP address, the source port, the destination IP, the destination port number, and the total number of bytes in the flow.

While this happens to be the order in which the fields appear within the flow record by default, we are not restricted to this order. The various fields will be displayed in whichever order to specify the field names that you pass to **rwcut**.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Statistics

SiLK provides some very useful statistics functions!

- `rwstats` allows you to "bin" the data.
- `rwcount` allows you to "bin" the data by time.
- `rwuniq` allows you to find uniq values.

rwuniq	rwstats	rwcount
pro| Records| 2| 25782| 6|12614033| 58| 21434| 132| 1| 41| 1| 17| 4439098| 1| 20077|	INPUT: 12614033 Records for 65520 Bins and 2316423618461 Total Bytes OUTPUT: Top 5 Bins by Bytes dPort| Bytes| %Bytes| cumul_%| 443|203730268095| 8.795035| 8.795035| 55879| 15110595545| 0.652324| 9.447359| 25012| 15094894062| 0.651647|10.099006| 6110| 14442214876| 0.623470|10.722476| 58337| 14440801761| 0.623409|11.345886|	Date,Records,Bytes,Packets, 1539594000,2310.31,8552467.29,15518, 1539594900,1663.78,7790346,17126.42, 1539595800,2749.7,16295474,31483.02, 1539596700,1664,6193512.22,15513.00,

One of the places that NetFlow and SiLK really shine is in the area of the statistics that can be generated. Consider the examples on the slide. In the first column, we have asked SiLK to tell us how many and which of the unique IP protocols appear within our network within a given time period. Think about the power of this. The data generated for the slide is looking at just over a one-month period, but we can easily look at a year or more. Finding the unique protocols used over the past year will take a SiLK repository a few seconds to process. How long would it take you to answer that same question if you had a year's worth of packets?

The middle column is showing which destination port numbers have the greatest number of bytes sent to them, regardless of which connection those bytes are seen in. Again, the results are available very quickly.

The final column changes the output format a bit. Rather than using the standard SiLK columnar output, we have modified the options so that it produces comma-separated values. These rows are showing us how many flows, bytes, and packets were seen during each 15-minute time frame. That data is perfect for passing into a visualization tool!

If you would like to try these commands yourself, these are the commands that were run on the class VM to produce the output above:

Rwuniq:
rwfilter --type=all --start-date=2018/10/01 --end-date=2018/11/15 --proto=0-255 --pass=stdout | rwuniq --fields proto

Rwstats:
rwfilter --type=all --start-date=2018/10/01 --end-date=2018/11/15 --proto=6 --pass=stdout | rwstats --fields dport --bytes --count 5

Rwcount:
rwfilter --type=all --start-date=2018/10/15T09 --end-date=2018/10/15T09 --proto=0-255 --pass=stdout | rwcount --bin-size=900 --timestamp-format=epoch --column-separator=, --no-columns

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Quick Research: Weird Protocol

You may have noticed protocol 41 on the last slide.

- What is it? When did that happen? Who was involved?
- SiLK answers the question very quickly!

```
sans@sec503:~$ rwfilter --type=all -start-date=2018/10/01 --end-
date=2018/11/15 --proto=41 --pass=stdout | rwcut --fields
sip,sport,dip,dport,proto,type
      sIP|sPort|                         dIP|dPort|pro|   type|
 131.193.34.220|    0|          74.89.64.229|    0| 41|ext2ext|
```

- Since this is ext2ext traffic, it appears to be misdelivered from our ISP.

Of course, we can find out who was involved! First, generating the statistical information on the previous slide took only a few seconds. In our experience, with modestly sized hardware, you can produce reports such as this for multiple years' worth of NetFlow data in minutes… not days or hours!

To find out who is involved, we know the time frame and we know the protocol, so we now search our repository for protocol 41 within that time frame and send the output to **rwcut**. As you can see, in under two seconds, we are able to identify the hosts involved. Since we can see that this was seen in the **Perimeter** sensor and we know (*we* know… we don't expect you to know!) that the external IP address for the network being defended is 74.89.64.229, we can rest easy, knowing that this weird protocol packet did not originate on our network; instead, it tried to get into our network.

Was it successful? Absolutely not. How can we be so sure? Think about it for a moment. Remember that we are specifying **--type=all** in our **rwfilter** command? This will cause it to check the flows for *all* our sensors. Therefore, since this flow was only seen in the perimeter, we can be confident that the packet did not traverse the firewall!

We just went from very high-level information to deciding if something was a threat in *seconds*… and we didn't use any signatures to do it!

https://t.me/learningnets

©SANS Institute 2021

Many Fields

- The version of SiLK on the VM supports 29 fields.
  - Some are not available because we did not populate geographic data.

```
sans@sec503:~$ rwfilter --type=all --proto=0-255 --pass=stdout | rwcut --fields 1-15,20-29 | head -2

              sIP|            dIP|sPort|dPort|pro|   packets|     bytes|    flags|                  sTime|
duration|                    eTime|    sensor|    in|   out|                                     nhIP|cla|    ty
pe|              sTime+msec|             eTime+msec| dur+msec|iTy|iCo|initialF|sessionF|attribut|appli|
```

- Note that many of these are not in the default set!
  - Of particular note: **initialFlags**

NetFlow (and SiLK) supports many different fields. You can refer to these fields by name or by number. Especially when working with a new NetFlow repository, we will usually ask the system to dump out fields 1–60 just to see what happens. If you do that with the repository on the VM, you will find that there are a few fields (16, 17, 18, and 19) that produce errors; this is because we have not provided IP-to-geographic-location mapping data. However, if we skip over these fields, we find that this installation has fields up through 29.

Realize that SiLK will only display the fields that are set as the default in its configuration. We are bringing these extra fields to your attention because at least one of them is extremely useful. Let's talk about **initialFlags**.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Flags vs. initialFlags

Flags is the aggregate from all packets seen in the flow:

- `initialFlags` is only what appeared in the first packet seen
- Format: `--flags-initial=<flags you want>/<flags mask>`
- Allows you to identify easily who started the flow
- Allows you to determine if the flow started in this hour

```
sans@sec503:~$ rwfilter --type=all --start-date=2018/10/01 --end-date=2018/11/01 --flags-initial=S/SA --
pass=stdout | rwcut --fields sip,dip,dport,flags,initialflags | more
                                    sIP|                                    dIP|dPort|   flags|initialF|
                           192.168.2.86|                           172.20.5.239|45265| S      | S      |
                           192.168.2.86|                           172.20.5.239|45265| S      | S      |
                           192.168.2.86|                           172.20.5.239|45265| S      | S      |
                          192.168.2.161|                          34.239.109.14| 9543|FS PA   | S      |
```


The **Flags** field in NetFlow represents the aggregate flags from a session. In other words, if the flow started with a SYN, then an ACK went by, then a PUSH ACK, then a whole bunch more ACKs and PUSH ACKs, finally ending with a FIN ACK, the aggregate flags would be **FSPA**. This shows the aggregate (or all) of the unique flags that appeared.

Which flags were in the first packet seen? Well, we know the answer because we just said that it was a SYN. What if you didn't have this inside information?

It turns out that NetFlow will store the flags from the first packet separately in **initialFlags**. This allows you to find things like connection initiations and listening ports! To do so with SiLK, we use the **--flags-initial** option. Yes, we know it's backward. ☺

To use this field, our bit-masking knowledge comes into play. Along with **flags-initial**, you must specify two additional values. The first is the list of flags that you *want* to find or which are required for there to be a match. The second is the set of flags that is *examined*. When we say, "the flags that are examined," you should think ”bit mask”!

In other words, using BPF, we could say something like **tcp[13]&0x3f=0x02** to find initial SYNs. To do this in SiLK, we would write **–flags-initial=S/FSPARU**. So we say which flag(s) we require and list the flags that should be masked *on*, just like we did with BPF!

https://t.me/learningnets

©SANS Institute 2021

Flow Continuation

- If the flow started in a previous hour, the **initialFlags** must not be **S** or **SA**.

```
sans@sec503:~$ rwfilter --type=all --start-date=2018/10/01 --end-date=2018/11/01 --flags-initial=S/S --
fail=stdout | rwfilter --input-pipe=stdin --flags-initial=A/A --pass=stdout |  rwcut --fields
sip,dip,dport,flags,initialflags | more
                                   sIP|                                    dIP|dPort|   flags|initialF|
                        209.85.232.109|                           192.168.2.40|65464|F  PA   |   PA   |
                           17.36.205.4|                           192.168.2.40|65463|F  PA   |   PA   |
                           17.36.205.4|                          192.168.2.146|49687|F  PA   |F  PA   |
                         17.249.172.28|                           192.168.2.43|49790|   PA   |   PA   |
```

- Allows you to determine the actual duration of a connection

This leads to something that might at first seem odd. NetFlow data is logged at two points. The first is when the connection *ends*. The second is after a period of *time has elapsed*. While the time period is configurable, usually at your sensors, SiLK uses a default of 30 minutes. This means that, if a connection starts up and continues for an extended period of time, you will never see it until the end of the current time period.

However, this also means that in the next time period, the connection is now viewed as existing. So what happens to the **initialFlags** in this case? They will report the first flags seen in that session! You can see some examples in the slide. We have filtered the data so that we are only looking at flows that are continuations from the previous period. It is purely coincidental that, for the first pair of these, the **flags** and **initialFlags** match. Consider the first flow listed. This indicates that the very first packet seen had PUSH and ACK set, but that at some point after that, the FIN bit was also set, implying that this flow ended during this period. In other words, we know that this flow started during some prior refresh interval and that it ended during this refresh interval!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Converting from Packets

It can be very valuable to convert from packets to NetFlow.

- We have a packet capture or repository with more than a gigabyte of packets.
- We are running more than one or two commands against the entire capture.
- Why not preprocess it all to NetFlow first, then answer questions?
- This can be a huge time saver.

```
sans@sec503:/sec503$ analyze backbone | rwp2yaf2silk --in=- --out=backbone.silk
sans@sec503:/sec503$ rwuniq --fields proto backbone.silk
pro|   Records|
  6|    778967|
 58|       344|
 17|    100258|
  1|     50008|
```


When we find ourselves using a packet capture or repository that is larger than a gigabyte or two, and we find ourselves running multiple commands against that entire capture or repository, it's time for us to think about converting that data to NetFlow. For example, in this course we have queried the backbone data repeatedly. How many ports are listening? How many ports are probed? How many hosts are there? These and many other questions have been tackled in our bootcamp work. Some of these commands can take minutes to run.

What if we converted the entire repository into NetFlow data first? We can answer all those questions in a fraction of the time because we do the up-front processing of the packets first. All of the queries then make use of the much smaller, very efficiently indexed, NetFlow data!

The command **rwp2yaf2silk** converts a standard libpcap file to flow format. You have to install both the SiLK tool suite along with another open-source package known as *Yet Another Flowmeter (YAF)* and all the prerequisites that YAF requires in order to install and run it.

The **rwp2yaf2silk** command has two required command-line switches: the **--in** switch and the **--out** switch, which specify the name of the input libpcap file and the name of the created file to store the output flow records, respectively. The above command uses the data in the backbone repository to generate a set of NetFlow records that we can use with SiLK.

Just to make sure we successfully converted to flow records, we run the **rwuniq** command on the converted file. We ask for the list of unique protocols that were seen in the data, passing the **backbone.silk** file as the data source. Our results are delivered almost instantly!

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

rwfilter Statistics

We can use the `--print-statistics` option to see what's involved in a query.

- Rather than running an expensive query against the entire repository, we can gauge just how expensive it will be, *first*.

```
rwfilter --type=all --start-date=2018/01/01 --end-date=2019/12/30 --
proto=0-255 --print-statistics


Files 11510.  Read   62059218.  Pass   62059218. Fail          0.
```

- This query examines 11,510 files, reads 62,059,218 flows, all of them match, and none of them fail

SANS SEC503 | Intrusion Detection: In Depth 36

A useful option that **rwfilter** includes is the **--print-statistics** option. This option allows us to evaluate just how expensive a query against a repository is going to be before we perform the full query. It's sort of like profiling the query.

For example, in the slide we ask for all of the flows between January 1, 2018, and December 30, 2019, asking for all protocols. The output reports that answering this question will require 11,510 files to be examined. Within those files, there are more than 62 million flows, all of which match. If we were to send that through **rwcut**, we would have hundreds of thousands of pages of output, which is probably undesirable.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Output

- What to do with extracted records' required parameter:
  - --pass=stdout
  - --fail=stdout
  - --print-stat: Print count of records passing/failing to screen
  - --print-vol: Print counts of flows/bytes/packet
  - --max-pass: Maximum number of flows to pass
  - --max-fail: Maximum number of flows to fail

An *output parameter* is required to designate the type of output and whether or not the filtered flows should pass or fail further processing. The **--pass=stdout** option passes the selected flows for further processing by piping them to another command such as **rwcut**. Conversely, the **--fail=stdout** option passes flows for further processing that failed to meet the selection criteria. As an alternative, the **--pass** and **--fail** options can designate a filename to store the binary output rather than passing it to another command for processing.

Some summary options print the number of records that pass or fail the selection or print the flows, bytes, and packets to the screen. For instance, you may want to take a sampling of traffic and limit the number of flows for further processing. This can be accomplished using the **--max-pass** and **--max-fail** to designate the number of flows to process.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

SiLK Wrap-Up

- Handy tool for examining network flows
- Extremely different from tcpdump/Wireshark
- Many different commands to examine different aspects of traffic summary
- Best used with tools that capture packet data

The many SiLK commands give a different perspective on traffic analysis. You typically use the **rwfilter** command to begin your analysis process and examine flows of interest by using different parameters. SiLK is concerned about network flows—a summary of traffic in a single direction.

The SiLK tool suite analyzes traffic in an entirely different manner from the other tools we've examined—tcpdump and Wireshark, which are more concerned with sessions of data between hosts, packet headers, and data. The SiLK tools are best used with full packet capture tools that contain data because the abbreviated output of SiLK often provides a good starting point for further investigation.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

SiLK Exercises

Workbook exercises "SiLK and NetFlow"

This page intentionally left blank.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

- Network Forensics
- Practical NetFlow Applications
- Modern and Future Monitoring

Modern and Future Monitoring

Being Data Driven

What Is Machine Learning?

Experiments with Analytics and Machine Learning

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Alert-Driven versus Data-Driven Sensor

Alert-driven sensor:

- Generates message/alert on traffic
- Provides minimal contextual data with alert
- Generally, does not capture other packets associated with the same session

Data-driven sensor:

- Collects traffic, perhaps payload too
- Stores, perhaps in some efficiently searchable format
- Analyst or canned scripts responsible for finding/searching for noteworthy events

SANS SEC503 | Intrusion Detection: In Depth 41

Another major thing that we would like to use as the premise of this entire section is the idea of moving from being *alert* driven to being *data* driven.

We've covered monitoring and alerting systems such as Snort and Suricata in this class. Other technologies work in the same fundamental way. For instance, if you have a SIEM, the most common configuration is to aggregate logs and generate alerts based on signatures within those logs. These are all *alert*-driven systems, and they make our workflow alert driven.

While Zeek can be used as an alerting system, for us it is more of a correlation engine. By the end of that material, we had learned how to implement correlation systems based on behaviors within our network data. In other words, we aren't waiting for some particular signature to match; we are instead reacting to patterns and behaviors within our data. This is a start at *data*-driven analysis. SiLK and other NetFlow-like tools are often used to simply log information about connections on our network, but when used properly, these represent tremendous sources of data that can be marshalled to perform anomaly detection or even predictive activities.

A data-driven sensor just blindly collects traffic. It does not try to interpret or associate any meaning to the collected packets. Oftentimes a data-driven sensor stores the data in an efficient format for optimized searching through a large amount of data (NetFlow does this). The analyst is responsible for finding or searching for noteworthy events or creating tools and systems that can identify interesting patterns, behaviors, and events.

We like to refer to the process of purposeful data-driven search and analysis as ”looking for trouble."

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Become Data Driven

If we only ever wait around for alerts, we will always be in a world where enterprises are compromised for six or more months before detection!

How can we change this dynamic?

This is the land where Threat Hunting lives!

We want to encourage you to take the things you have learned in this class, and that you are learning in the exercises, and to use them to start looking at your network in new ways. If you think about it, there's really something wrong with six or more months being the average amount of time that an enterprise network is compromised before it's detected (according to the Verizon breach reports). What can we do to change this?

There must be something. It feels as though what we are doing now (i.e., waiting around for our systems to tell us there's something wrong and for the vendors to send us new signatures) must not be sufficient. If it were, the average would be shorter than six months, wouldn't it? Think about some of the major compromises that have occurred; often the intrusion is not detected by a security tool! Instead, these compromises are often detected because of secondary effects. This could be something like the attacker inadvertently filling a hard drive, taking a system down; use of compromised credentials or PII on other sites, leading to fraud detection systems at banks potentially identifying the common source of the data; or because the attackers eventually get bored and begin to use the compromised network to attack others (or conduct false flag operations).

Clearly, we need to change from being reactive, waiting for alerts to happen, to being proactive, hunting for unknown threats through advanced analysis techniques.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Think About This

Think about application protocols and how they influence the payload.

- Won't an application protocol typically have a header of some kind?
- Could we identify unknown protocols by finding common initial payloads?

If you graphed the frequency of each of the following, what would you predict that it would look like?

- TCP sequence numbers
- TCP port numbers
- IP ID numbers
- IP checksums

SANS SEC503 | Intrusion Detection: In Depth 43

To start using this on *your* network, start thinking about things in new ways. Stop focusing on signatures and alerts so much and start thinking about anomalies and behaviors. Signatures and alerts still have their place, but we know that there is far more happening than signatures will find!

For example, you know a great deal about network protocols now. Think about application layer protocols. While every protocol is different from every other, are there still commonalities? For example, if you were going to implement some kind of file transfer protocol, wouldn't you need some type of application header that is used to initiate transfers, define filenames, indicate transfers have completed, resume transfers, and so on? Wouldn't this require some kind of application layer header? If an application uses a header, doesn't it make sense that the format of it and perhaps even some of the bytes within it will be somewhat static if we were to look at a large number of them?

Take a few moments and see if you can produce some quick sketches of what you would expect a histogram (frequency analysis) of TCP sequence numbers to look like for large numbers of packets, regardless of the session. What about TCP source port numbers? IP IDs? How about checksums?

Once you have some ideas jotted down, continue through this section to see what we find when we graph this data out!

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

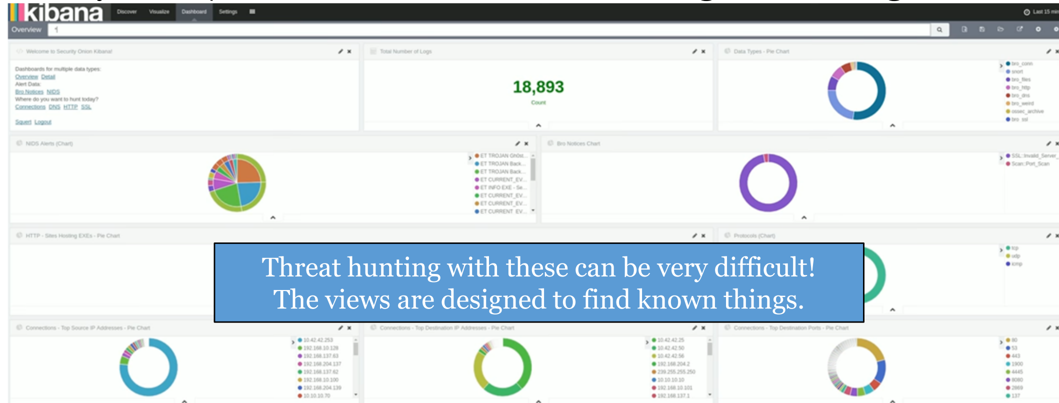
Other Visualization Options

Visualizations are everywhere!

- SecurityOnion
- Every SIEM/SEM

There are limitations.

- Weird countries are great.
- Large flows are great.



Before we start looking at some custom visualizations, it is important to acknowledge that there are many free and commercial options available for visualizing your data. For example, the venerable Security Onion offering by Doug Burks includes some wonderful visualizations using Kibana that are generated from your Zeek and Snort logs out of the box. On the commercial side, we have every single SIEM and SEM on the market today, in addition to some very nice visualization tools from companies like SolarWinds and Arbor Networks.

As wonderful as these are, especially as a dashboard view of the network over time, we find them to be of somewhat limited value when it comes to threat hunting. Certainly, having a visualization that shows me a heat map of connections to other parts of the world is handy. That might be a strong indicator that something is wrong. Having a diagram that shows me hosts involved in large flows is also very nice, as is a chart showing me which alerts are happening most frequently in my organization.

The trouble is that these are only useful at looking at the data in ways that the creator envisioned. For example, it is quite challenging to convince Security Onion to allow you to look at more than a 30-minute swath of data. Additionally, there is no way to ask any of these tools to drill into arbitrary bytes or fields within packets and represent that data in any kind of visualization.

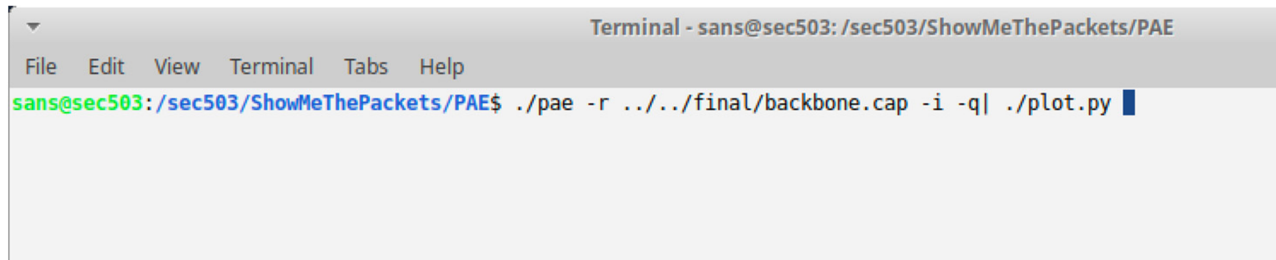
For these reasons, we're going to explore some "homebrew" visualizations. These are largely made possible, not because of our ability to write code, but because of our deep knowledge of networks and communications. This knowledge allows us (which includes you at this point!) to ask interesting questions that these tools can't answer.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Handy Tool: PAE

- PAE: Packet Analysis Engine
  - Consumes raw pcap files
  - Very fast
  - Generates frequency analysis (and other) as output
    - Has a handy Python script that will turn that into a graph!



Extracting the first 32 bytes of application layer data to look for headers and graphing out arbitrary fields in packets may sound like a difficult things to do. If you're thinking about taking the ASCII tcpdump output and jamming it into Excel, please don't! If you don't crash Excel, you're not looking at enough data for it to be useful, and there is a better approach.

A much better alternative is a tool called PAE (Packet Analysis Engine). This is a very simple tool written in C that can take massive packet captures, process them very quickly using hash tables, and spit out frequency analysis information. The output is just raw data, so we need something else to make it look a bit better. For this purpose, there is a Python script that comes with it that will turn the data into a pretty graph. (While PAE is very fast, be aware that Python can be very slow, especially when trying to graph sequence numbers, which are in a 32-bit space!)

Let's look at some different ways to graph out data that we find useful.

https://t.me/learningnets

©SANS Institute 2021

Handy Feature for Payload Similarities

Finds very uncommon and very common "app headers"

```
Terminal - sans@sec503: /sec503/ShowMeThePackets/PAE
File   Edit   View   Terminal   Tabs   Help
sans@sec503:/sec503/ShowMeThePackets/PAE$ ./pae -r ../../final/backbone.cap -q | more
+-------------------------------------------------------------------------------------+
02 02 00 00 00 02 00 00 c0 a8 3c 00 ff ff ff 00 00 00 00 00 00 00 00 01 00 02 00 00 c0 a8 3d 00
.  .  .  .  .  .  .  .  .  .  <  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  =  .    -#: 6101
+-------------------------------------------------------------------------------------+
d2 4e 01 00 00 01 00 00 00 00 00 03 76 31 30 0a 76 6f 72 74 65 78 2d 77 69 6e 04 64 61 74 61
.  N  .  .  .  .  .  .  .  .  .  .  v  1  0  .  v  o  r  t  e  x  -  w  i  n  .  d  a  t  a    -#: 5
+-------------------------------------------------------------------------------------+
9c 4e 01 00 00 01 00 00 00 00 00 03 76 31 30 0a 76 6f 72 74 65 78 2d 77 69 6e 04 64 61 74 61
.  N  .  .  .  .  .  .  .  .  .  .  v  1  0  .  v  o  r  t  e  x  -  w  i  n  .  d  a  t  a    -#: 5
+-------------------------------------------------------------------------------------+
48 59 81 82 00 01 00 00 00 00 00 01 73 05 61 6d 69 67 6f 04 6d 61 69 6c 02 72 75 00 00 01 00
H  Y  .  .  .  .  .  .  .  .  .  .  s  .  a  m  i  g  o  .  m  a  i  l  .  r  u  .  .  .  .    -#: 4
```


One of the things that we find PAE especially handy for is identifying very rare and very common payload data. This is called *Long Tail Analysis.* This term comes from a histogram view of the data. When we graph the frequency of arbitrary data (like packet payload bytes), the majority of the data will tend to cluster around the average or *mean.* The *standard deviation* is the average variance from the mean. The further from the mean, the fewer items you will find in this *bell curve*. In this bell-shaped curve, the very frequent and very infrequent items appear in the “tails” to either side.

There is an option in PAE to set the "Anomalosity" value for data. This number implements a *notch filter.* A notch filter keeps things that are outside of the notch (the "tails") and eliminates the center (the mean). If you leave this number alone, you will only see data that appears several standard deviations from the mean. The mean is the average frequency with which any given payload appears. PAE analyzes 32 bytes by default, but an undocumented option, **-b**, allows you to specify the number of bytes to analyze.

How is this helpful? Take the output in the slide. The very first payload occurs more than 6,000 times in just a few days. It appears to be binary data. There's nothing that looks especially recognizable, but since it appears so frequently, it is almost certainly some kind of application header. Could we use some of this data to extract the related packets and then begin to research? Of course!

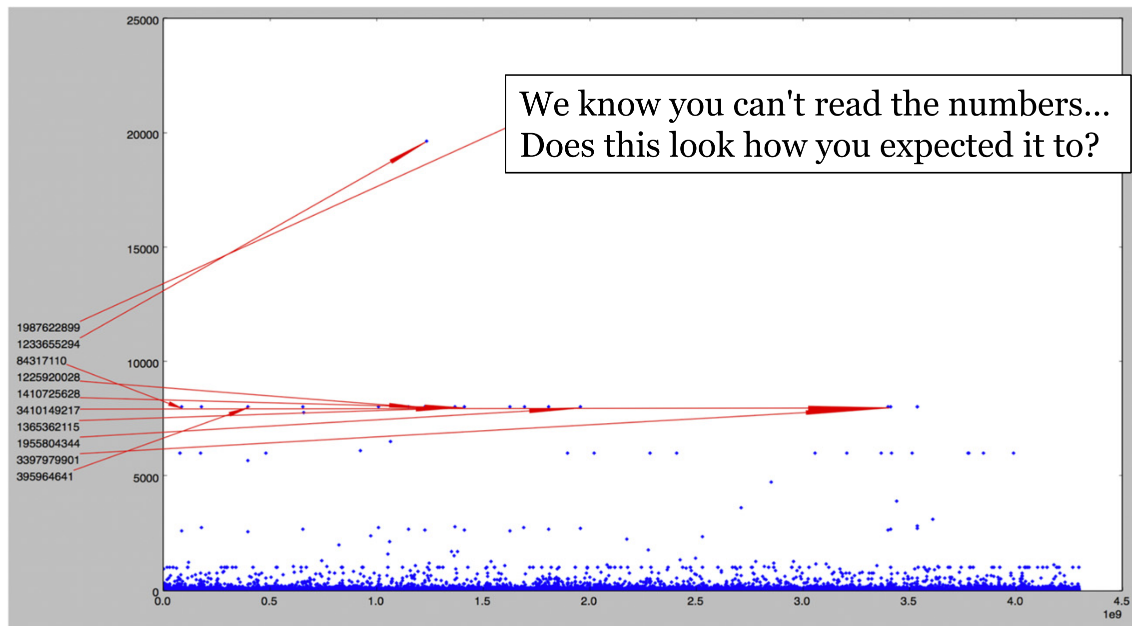
The two payloads following are also very clearly application headers. Looking at the bytes, we can see that they appear to follow the same structure. Is the fourth related? We can find unknown protocols this way.

Think about command and control beacons. Usually there will be a very standard structure to the beacons. This provides another way for us to find them!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Sequence Numbers



15

Here we can see an example of the output from PAE. First, we fully understand that the numbers on the graph are likely too small to read from a distance on a projector screen. We're sorry about that, but the actual numbers don't *matter*. The big question: Does this graph look anything like what you predicted a frequency analysis of sequence numbers would look like? Probably not!

Think about what some of this means. For example, we see that there are two sequence numbers that appear thousands of times more frequently than any others. What would cause a sequence number to repeat over and over and over? Could this just be an outlier? Probably not. It's just too many repeated instances.

Instead, think about how TCP behaves. The sequence number tracks the number of bytes sent by a host during the connection. Additionally, every time someone sends you data, regardless of how much, you must ACK that data. What if there were a large file transfer? While the sequence numbers for one side of the connection would continually increase, the sequence numbers *in the other direction* would remain static!

OK, so there were two large transfers, downloads, videos, or something… But there's more that we can ascertain. What if we looked more closely and found that, say, 15 sequence numbers all appeared way more frequently than the average, and they all appeared *exactly* the same number of times? Would a hypothesis that this indicates that multiple people downloaded the *same* file or watched the *same* cat video make sense? Could we test that? Do we have tools that would allow us to extract packets based on the sequence number?

https://t.me/learningnets

©SANS Institute 2021

The Advantage of Developing Analysis Methods

When you think of an interesting analysis, try to generalize

- Perhaps you cutting and grepping as a first cut
- It won't be pretty, but it works, and you can test your theory.

If it turns out to be useful, generalize and automate!

Andy and the "Sequence Number Challenge" from server:

- `00:03:10.93` tcpdump with sed (the answer on the server)
- `00:18:45.18` tshark with some parsing
- `00:43:0.19` Python/Scapy
- `00:00:30.08` PAE

If you think up an analysis that sounds useful, test it out! It doesn't matter how long it takes to run, and it doesn't matter how "pretty" your approach is; it only matters if the result is useful to you. If you come up with something that's really useful, though, you should definitely take the time to not only automate it, but to generalize it!

What this means is that you should take some time to figure out how to write a Zeek script, for example, or to write a simple Scapy script. Possibly, you should take a complex set of parsing instructions and save them into a shell script so that you can use them again and again. Here we take this even further. Since we find it useful to perform some statistical analysis and visualization on arbitrary bytes in very large numbers of packets, we've written some code to do it!

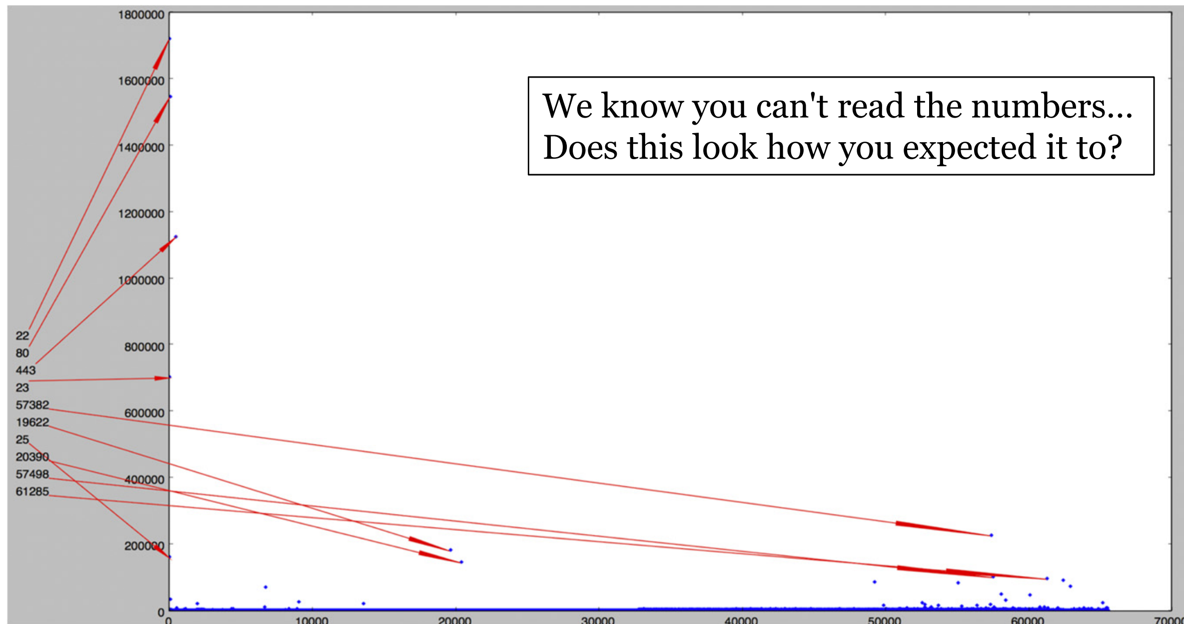
Andy Laman, a major contributor to this course, decided to solve one of the challenges in the score server four different ways. The question asks you to identify the sequence number that appears most frequently in the internal network. The solution provided in the score server will take about three minutes to run. Solving the problem with tshark will take nearly 19 minutes to run. Running a Scapy script takes nearly 45 minutes. Using PAE, we can answer this question in 30 seconds!

What's the lesson? It's what we said at the outset of this page. If you come up with something that seems useful, test it out. If it *is* useful, take the time not only to automate it, but to generalize it so that you can apply it to many problems rather than just one!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

TCP Ports



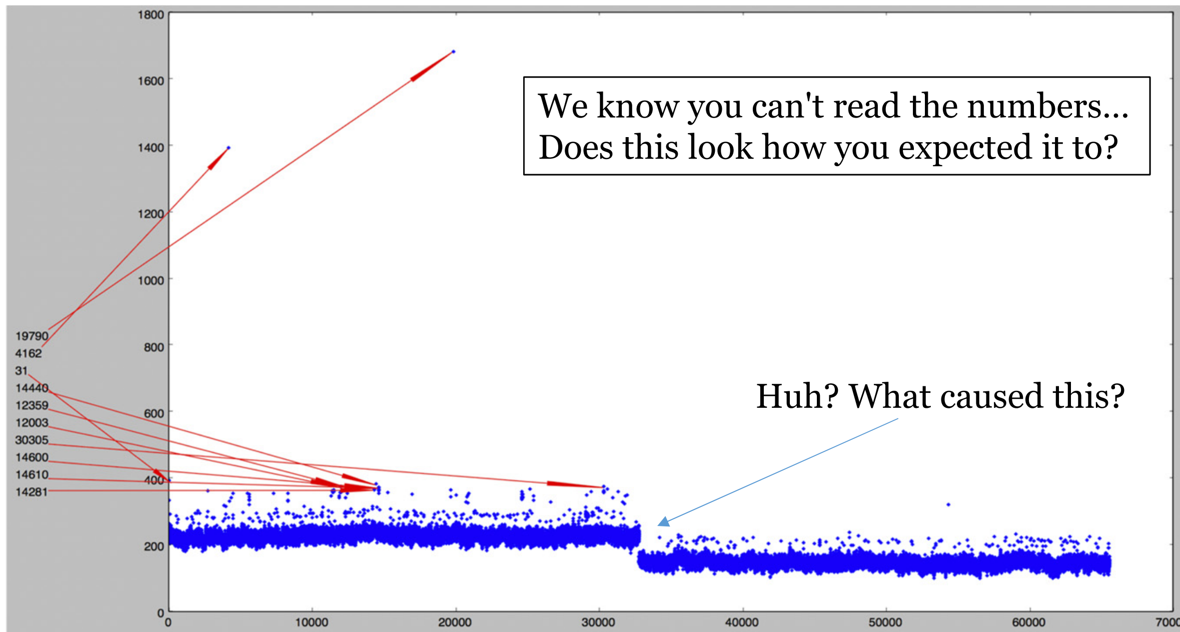
We can do the same type of analysis using the port numbers. Here we find some ports that are not very surprising: ports such as 22, 80, and 443. But the fifth most common port is 57382. What is that? Was that being used as a client port? Is it a service port? Can we find out?

Are you already looking at the data in an entirely different way? Are you finding things for which *there was likely no alert!?* This is entirely the point. We are now finding things to research and investigate based on the *data* rather than an *alert.*

https://t.me/learningnets

©SANS Institute 2021

IP IDs



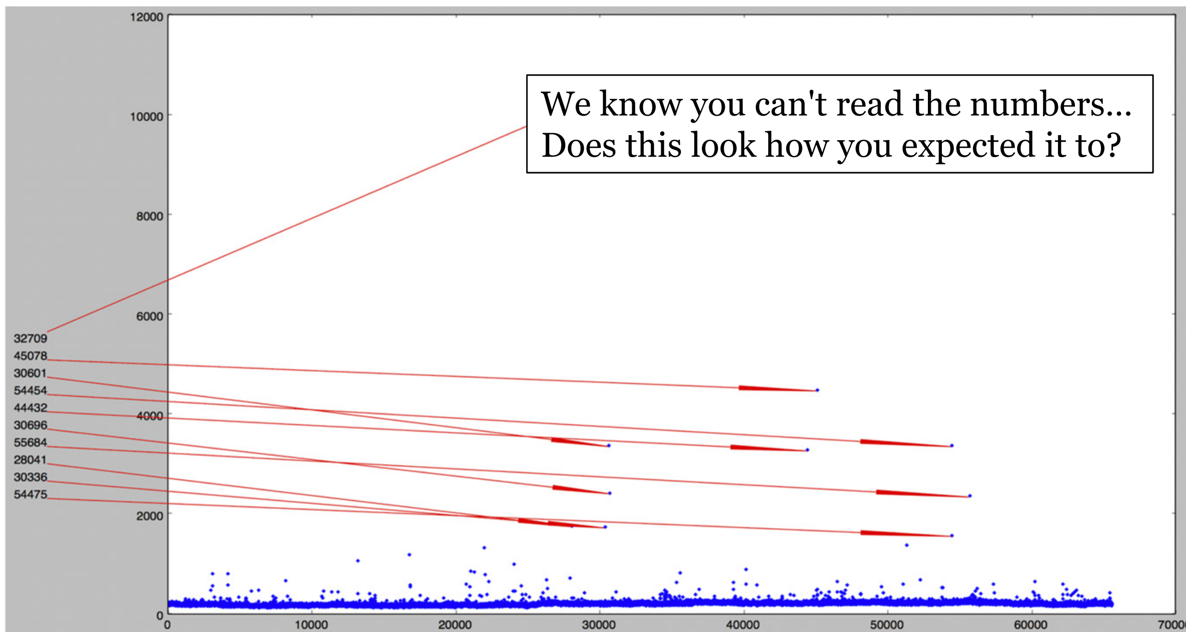
We can apply the same process to the IP ID numbers. While there are a couple of interesting outliers that we should research, there's another behavior that's super interesting. Notice that at around the middle, there is a sharp drop in frequency. What could cause this?

We're not going to answer this question here. Think about it. Research it, if you can. You might be surprised to discover that there is almost nothing out there that describes this behavior. If you think you've figured it out and want to chat about it, reach out to us in the class Slack channel, and we'll see if you've figured it out!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

IP Checksums



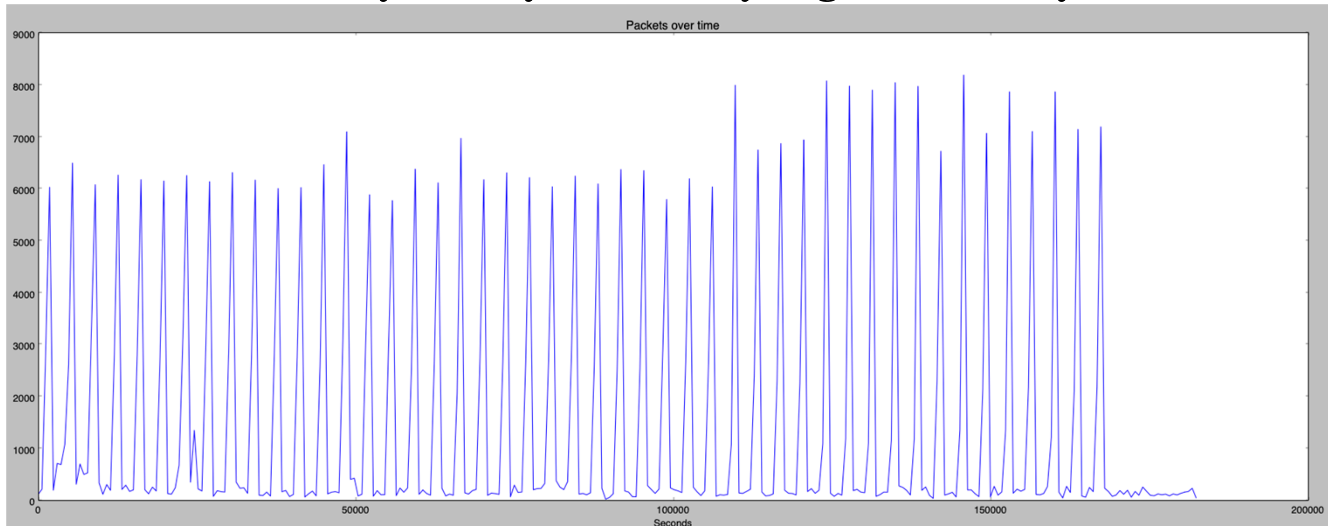
Lastly, we've included the checksums. The checksums turned out to be an interesting item for those of us who developed and maintain this class. The research is still in progress, but our initial theory was that no checksum would be any more likely than any other…. However, after seeing this, we've thought about it a great deal more, and we suspect that the checksum is far less "random" than you might think!

While we haven't put a great deal of energy into this particular analysis (because we suspect that the value is quite low), it does turn out that if you run this same test on large numbers of packets from different networks, you will see roughly the same shape, but it will shift to the left or the right. Our suspicion is that this is correlated with the change in network addressing within different networks.

©SANS Institute 2021

Graphing SYNs Over Time

There is very clearly some very regular activity here!



SANS | SEC503 | Intrusion Detection: In Depth 52

The diagrams that we have looked at so far just graph the frequency of the data from some large chunk of time. Don't limit yourself to this view! When we think about networks and communication, time-ordered sequential behavior is how it all works.

What if we selected some feature of communication and graphed the frequency of that feature over time rather than simply determining how common that feature is overall? For example, what if we asked, "If we were to look at our network as ten-minute slices of time over the course of 48 hours, how frequently do we see SYN packets originating from within our network within those time slices?"

That still might be confusing to read… but it's much easier to look at! What we've done in the slide is produce a graph based on the following tcpdump command line:

```
mergecap -w - packets/backbone/dailylogs/2019-05-0[23]/* | \
tcpdump -tt -r - -n 'tcp[13]&0x12=0x02' | cut -f 1,3 -d ' '
```

The output of that command will be a UNIX timestamp followed by the source IP address from the network monitored. We've taken those lines and pushed them through a Python script named "freq.py" to look at this data in ten-minute time slices. You will use this script in the next exercise.

What does this do? Look at the slide! It is very clear that there is some kind of very regular behavior occurring within the network. We might not know what it is from this graph, but this gives us yet another starting point to investigate activities in the network!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Apply Data Science, Statistics, and Machine Learning

Very powerful analyses are often easy to apply

- Applying Fourier analysis to the data on the previous slide makes the periodic SYN scan on this network appear almost instantly.
- Applying a DBSCAN analysis with some custom transformations to the dns.log data (from Zeek) for this network instantly reveals an Iodine C2 channel. (See the next slide.)
- Machine learning can allow you to create smart anomaly detection to find almost anything. We'll talk through and demonstrate this in this book.

Where can I learn how to do this stuff?

- SEC595: Applied Data Science & Machine Learning for Cybersecurity Professionals

Based on what we've covered in this class, we have a lot of different things we know about network, transport, and application protocols. We can use our domain-specific knowledge to create wonderful analyses and to identify anomalous behaviors when we visualize network activities. How can we go further?

The industry is benefiting more and more from the application of data science approaches. One obvious piece of evidence of this is the huge investment in marketing machine learning and AI solutions, or at least so-called AI solutions, in the network monitoring space. We're going to investigate how this works and try to come up with a usable real-time protocol anomaly detector in the second half of this book. If you really want to dig into this, though, we'd recommend you seriously consider the SANS SEC595 course: https://www.sans.org/cyber-security-courses/applied-data-science-machine-learning/

Just two of the things that are covered in that course include Fourier analyses and unsupervised clustering techniques such as DBSCAN. Carefully applying a Fourier analysis to the data on the previous slide makes it immediately apparent that there is something happening every hour like clockwork. Is it possible to detect this kind of thing automatically? Absolutely! We'll show you how.

One of the other minor topics is the use of unsupervised clustering techniques. One of the techniques covered in the class is DBSCAN. Let's see how that helps us with some DNS analysis on the next two slides.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

DBSCAN Applied to Zeek Logs

Transformations to turn DNS names into numeric values

- Applied an unsupervised clustering technique, DBSCAN
- These immediately become visible as an unusual cluster.

	query	qtype	rcode
54215	zdzqaabbccddeeffgghhiijjkkllmmnnooppqqrrssttuu…	10.0	0
55000	0mkb8m\xe2\xd933jli\xc5\xc0\xd9\xc9\xe2\xf7\xf…	10.0	0
55202	0ejb2\xf4\xd1\xdegs\xc9q\xf473p\xccb\xc6mww\xd…	10.0	0
55208	0uob7x9\xc4bet\xbd\xf2\xd73\xbe9\xe61n\xc9\xe2…	10.0	0
55779	0iwb6h\xc9\xc43\xe2m\xd0zqzy\xf6\xe6v\xc5\xc3\…	10.0	0
56740	0mobfebr\xf9\xc6y\xda3\xe6\xcei\xf1\xde\xc80\x…	10.0	0
56742	0qpbhy\xd13l\xcd\xc7\xf4\xf9uf9koy\xc3wj\xd1rn…	10.0	0

We're not going to dive into data science or machine learning at this point. That's well outside of the focus of this course. For now, however, we will simply state that we need to transform everything that we want to analyze into some kind of numeric representation. You'll see some of this as we walk through a high-level intuitive discussion and explanation of machine learning and its application to real-time protocol discovery in the second half of this book.

For now, just trust that we worked out a method to transform all of the data concerning hostnames in the dns.log generated by Zeek. After transforming the data into a number of different numeric features, we applied a principal component analysis to the data to do some dimensionality reduction, then applied the DBSCAN clustering algorithm.

The goal of any clustering algorithm is to attempt to identify and group things that are somehow related. DBSCAN is, we find, especially useful when working with network data. In the slide, we are showing you an abbreviated view of some of the data returned. The data seems unintelligible, but you actually know what that is. The long strings of characters are the DNS hostnames and domain names that were resolved. Normally, these are all in the log in the order that they appear. Finding something unusual can be difficult, unless you already know what it is. By applying DBSCAN, all the DNS queries that look like this end up in a single cluster. What can we do with this?

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

What Are Those?

Lookup was for:

- zdzqaabbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz.micros0ft.com

Could we find this manually?

- Sure! If you already know to look for it
- Applying data science techniques allows us to find this kind of thing automatically.

Even so, could we just look at the names ourselves?

- Sure… but what if your dns.log has > 50,000,000 domains in it?
- What if we told you the approach we took reduced that to ten clusters?
- Can you look at ten things?

Let's look more closely at one of these names. You can see here a string of ASCII characters followed by "micros0ft.com." That is just one of the DNS names resolved that ended up in one of our clusters. You might look at that and say, "Okay, but why do I need data science to find that? That's obviously weird and should be investigated."

Sure, it's weird, and that's easy to see. But what if that DNS query name is buried in more than 50,000,000 DNS queries? How easy will it be to see now? What if we told you that, using the technique we've described, we can take 50,000,000 queries and turn it into 10 or fewer clusters? If you only have ten clusters to look at, it makes it very easy to find something like this.

What is this, by the way? This is evidence of an Iodine covert C2 channel running on the network. Does that seem like something you'd like to find quickly? Again, we'd suggest you think about SEC595 for some great tools and techniques for finding this kind of activity. The course isn't specific to network monitoring. Instead, the course can be applied to almost any set of problems you have as a cybersecurity professional.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Basic Analytics

Workbook exercises “Basic Analytics"

Please turn in your workbook to the Basic Analytics lab so that you can experiment with some of these techniques yourself!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Section 5: Putting it all together & dealing with large scale data
- Guided scenarios to reinforce all skills
- Elaboration on effective deployment, collection, and analysis at large scale
- Moving beyond alerts to data driven analysis & detection

- Network Forensics
- Practical NetFlow Applications
- Modern and Future Monitoring

Modern and Future Monitoring

Being Data Driven

What Is Machine Learning?

Experiments with Analytics and Machine Learning

SANS | SEC503 | Intrusion Detection: In Depth 57

The material so far should give you ideas of visualizations that you might want to create that would be particularly useful within your environment. Let's take this a step further and push into the current frontier in information security: machine learning and artificial intelligence.

https://t.me/learningnets

©SANS Institute 2021

Our Approach

Our approach in this section is *very* different:

- High-level, intuitive approach to machine learning
- Some things are greatly simplified… don't fret over the deep details.

Throughout this class, we have taken a bottom-up approach. If you recall, this means that we would start from the most elemental components and progressively build our understanding of how things work by assembling those fundamental elements into more and more complex understandings of packets, protocols, and communications. This section is *completely the opposite.*

We, along with Doctor StrangeBob, are going to try to develop an intuitive understanding of the principles that stand behind modern machine learning approaches. Along the way, we hope you learn some important lessons about how machine learning works and, more specifically, what it doesn't do. It is our hope that, based on the following material, you will both be able to communicate with some authority about the current state of machine learning and AI relative to security products and possibly even to be able to apply some of these techniques within your enterprise to devise interesting machine learning applications.

Please bear in mind that, while we have made diligent efforts to be accurate in this section, we are by no means going to attempt to be thorough. This means that some of how things are explained will be gross oversimplifications. We have made an effort to ensure that nothing that we say is *wrong*, but there is definitely much more to the story than what we are saying here. Still, we all use technologies every day that we could not personally re-create and likely do not deeply understand.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Intuitive Understanding: Linear Regression

Might not seem like machine learning... but you might be surprised how closely related it turns out to be

- Take a graph of a bunch of data points.
  - In our case, bytes per second on a network link
- Analyze the graph, looking for a line that best fits the data.

$$y = mx + b$$
$$f(x) = mx + b$$

Machine learning, as it stands today, is largely a field of applied mathematics. Those words might cause you some dismay, but we are going to work hard to demystify many of the terms you will hear bandied about in machine learning books, articles, and marketing slicks.

To get started, we're going to talk about *linear regression,* which we can think of as one of the most basic examples of machine learning as it is performed today*: linear*, involving straight lines, and *regression,* which is used in this field as a measure of relationships between variables. For example, if we were to look at the cost of housing, it seems reasonable that the cost will go up relative to the quality and availability of public services. This is one of the reasons that housing in the New York metropolitan area is so much higher than it is in Saskatoon in Canada. There appears to be a relationship between these two variables. Measuring that average difference is called *regression*.

When we talk about a linear regression, we are trying to come up with the formula for a line that best captures the relationship between these values. For our purposes, perhaps we want to measure the number of bytes per second passing across a network link. A linear regression could potentially help us to predict how many bytes per second our network is likely to carry a year from now, allowing us to plan accordingly.

When we talk about lines, we can call back to the basic algebra and geometry that you learned in secondary school: $y=mx+b$, where x is the x coordinate (in our case, this might be time), y is the y coordinate (in our case, the number of bytes at that time), m is the *slope* of the line (the change in y relative to the change in x), and b is the "Y intercept," or the point along the Y axis where this line crosses. This same formula can also be written using $f(x)$ to replace y. There's really no difference; this just gives us a handy way to say that $f(x)$ represents the set of operations that we perform on x to produce y.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Our Laboratory

We will do all our demonstrations and experiments with Jupyter.

- Web-based interactive Python "notebook"

What you have:

- All the demos in the book are on your VM.
- All the experiments in the labs are on your VM.

What you are using:

- Python 3.9, Keras, NumPy, Scipy, Sklearn, TensorFlow, Matplotlib

The exam can only cover material *printed in books!*

Since we're taking an intuitive and exploratory approach to this topic, we're going to invite you to either follow the directions in your workbook (if you are taking this class OnDemand or through some other self-guided modality) or the directions that your instructor provides. All the things that are shown in the slides and all the experiments that you will conduct will make use of a very handy tool called Jupyter.

Jupyter provides a web-based interface to a Python notebook-style tool. These notebooks allow you to create chunks of code (and to take notes because it's a notebook!) and execute them… and then edit them and execute them again, iteratively tinkering with your code as you experiment.

In the long term, after working through something using Jupyter (which we absolutely use in the real world to develop machine learning applications within our business), we can take the code that we come up with and plug it into a Python script to use in production.

Since we have placed all the exercises and experiments into Jupyter notebooks, it is also very easy for you to take those notebooks somewhere else and use them to experiment within your environment. There are a number of dependencies required to make this work, which we have included in this slide as well as documented in the workbook.

We have chosen to build these demos/experiments in this way for two big reasons. First, Jupyter is probably the most commonly used tool by machine learning developers for exactly the kind of experiments we are about to do. Second, GIAC can only test you on things that are printed on paper in a book that SANS gives you. This means that you are responsible for everything in the course books, including the workbook, but you are *not* responsible for the content of the Jupyter notebooks that don't appear in these slides! This seems fair since this isn't a data science or mathematics course. ☺

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Getting Started

We need to do some math, so we import NumPy.
We also need data, so we ask NumPy to load our data set.


```
In [ ]: import numpy as np

In [ ]: # Load Netflow statistics for multiple days
        data = np.genfromtxt("sample_data/flowStats.csv",delimiter=",",skip_header=1)

In [ ]: print(data)
```


Let's get started with our first experiment. Since this is our very first experience with Jupyter, and since this is not a Python class, we'll walk through this example very slowly and explain everything that's going on.

We know that we want to perform "linear regression," which involves mathematics. While Python has built-in support for mathematical methods, there is a well-known Python library called NumPy that is both highly optimized and offers some advanced mathematics capabilities that are not built into Python.

After telling Python to import NumPy and to provide us with a convenient "handle" or "alias" to access the capabilities that NumPy offers ('np'), we will ask NumPy to use a handy function that it has called *genfromtxt.* (It is good practice in Python to ask for libraries to be imported to an alias in this way. While simply importing all of NumPy could be done, eliminating the need to use the 'np.' notation, this would also "pollute" the namespace, overwriting other functions with the same name.)

We've asked NumPy to load the data in that CSV file into a variable named *data*. Just to make sure everything is working so far, we'll ask Python to print out the result.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Examine the Data

Our data is a count of the number of bytes on a network at different points in time.

- The data is “two dimensional” in that each row has:
  - An hour number
  - A byte count

```
data = np.genfromtxt("sample_data/flowStats.csv",delimiter=",",skip_header=1)
```

```
In [3]: print(data)

[[2.10000000e+01 7.77400000e+03]
 [2.20000000e+01 5.88057000e+05]
 [2.30000000e+01 1.52452000e+05]
 [2.40000000e+01 2.66039000e+05]
 [2.50000000e+01 7.69964000e+05]
 [2.60000000e+01 1.24175300e+06]
 [2.70000000e+01 1.80619000e+05]
 [2.80000000e+01 1.03261600e+06]
 [2.90000000e+01 8.10980000e+05]
 [3.00000000e+01 2.42749900e+06]
 [3.10000000e+01 2.35670600e+06]
 [3.20000000e+01 3.59974000e+05]
 [3.30000000e+01 1.13421000e+05]
 [3.40000000e+01 1.14391200e+06]
 [3.50000000e+01 7.32788000e+05]
 [3.60000000e+01 4.89158200e+06]
 [3.70000000e+01 2.63760000e+05]
 [3.80000000e+01 3.85384800e+06]
```


Python prints out the content of *data*, showing us pairs of numbers inside of square brackets. NumPy uses the square brackets to indicate a NumPy array. In fact, you can see that it is showing you that you have an array that is composed of lots of smaller arrays.

Each of the smaller arrays has two values in it. These values were loaded out of the CSV file and represent an hour number (column 1) and a byte count (column 2). This data was extracted using the *rwflowstats* tool (part of SiLK) from a real network. It represents the number of bytes passing the sensor during each hour.

Let’s attach a few more terms to our data. We can say that the data in the CSV file has two *dimensions* because it has two columns for each row. The term *dimensions*, used in mathematics, refers to a number of related values. If there were two or three values, or dimensions, we would frequently think of them as *coordinates*, which can be visualized by graphing them onto coordinate axes (like X and Y for two dimensions, or X, Y, and Z for three dimensions).

Another way to refer to a set of coordinates is as a *vector.* A vector that has only one value in it isn’t typically referred to as a vector but is instead called a *scalar*. We will simply state that a *scalar* value is a single value that can do nothing other than *scale* another value or vector.

To summarize, we have created a NumPy array that has a number of rows. Each of those rows has a two-dimensional array within it, which we can also call a two-dimensional vector.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Manipulate the Data

Manipulate the data:

- We have 79 rows of two values (dimensions) each.
- To test our "learning," we will want to reserve some data as a "test" set.
  - Let's take the first 60 values to do our "learning."
  - We can use the remaining values as "test" values to see how well our resulting approach can predict the outcomes.

```
In [5]: print(f"The data has {data.ndim} dimensions of shape {data.shape}")

        The data has 2 dimensions of shape (79, 2)

In [7]: # Let's grab the first sixty hours of data, selecting only the data in
        # column 1... so, in row[x], we want column[1], which is the byte count
        # for that hour.
        bytes = data[0:60,1]
        # Print the result out so we can see if we got only byte counts.
        print(bytes)

        [7.77400000e+03 5.88057000e+05 1.52452000e+05 2.66039000e+05
         7.69964000e+05 1.24175300e+06 1.80619000e+05 1.03261600e+06
         8.10980000e+05 2.42749900e+06 2.35670600e+06 3.59974000e+05
         1.13421000e+05 1.14391200e+06 7.32788000e+05 4.89158200e+06
         2.63760000e+05 3.85384800e+06 2.85450000e+04 7.62825400e+06
         5.34269300e+06 4.45879400e+06 5.34050500e+06 9.60050700e+06
         9.95534100e+06 1.05269260e+07 1.13672040e+07 1.54931800e+07
         1.49292830e+07 1.09028550e+07 9.60557700e+06 9.00422400e+06
         1.69897980e+07 2.45866000e+06 2.28025180e+07 1.00792870e+07
         1.64182200e+07 2.71983200e+07 2.18313110e+07 1.06091600e+06
         2.30984110e+07 1.15058740e+07 4.48170090e+07 2.56370850e+07
         4.32183830e+07 5.02371500e+06 5.50694740e+07 1.29459440e+07
         1.48257110e+07 6.38544300e+06 5.99259740e+07 2.88231460e+07
         4.20884540e+07 7.41798440e+07 7.61446540e+07 1.79402210e+07
         3.77031580e+07 1.07154139e+08 1.56858600e+07 9.05054810e+07]
```


Let's ask Python to confirm some of what we've said so far. NumPy arrays have some handy functions that aren't available on normal Python arrays. One of those functions is *ndim*, which returns the number of dimensions in an array. The other is the *shape* function, which tells us how the array is structured. When we ask how many *dimensions* our overall array has, Python tells us that it has two dimensions. When we ask it what shape the array has, it tells us that it has 79 rows of two dimensions.

Before we try to create a line that "fits" this data, let's reserve some of the data for testing. This is an important step in machine learning activities. We always want to have some data set aside that we can use to verify how well our *model*, or predictive approach, fits real data. It should be obvious that our model *should* fit the training data very well! If it doesn't, we are likely using the wrong approach to try to model our data.

To set aside some data, we will use a very handy capability that NumPy arrays make available: simple array slicing. Note the line that states the following:

```
bytes = data[0:60,1]
```

What this does is it copies the second column of the rows at offsets 0 through 60 (remember, array columns are referenced by the *offset*, which should feel very natural for you) into a variable called "bytes." We then print out the contents of that variable just to make sure it's what we expect.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Preparing to Plot

More prep:

- Create an array, "hours," that will serve as our X axis.

Magic handwaving:

- Define a function that can plot stuff.
- This isn't a Python class, so we're going to provide you some utility functions.

```
In [ ]: # Create a set of hours that we can map this against.  Honestly,
        # we could have simply used the entire data set as-is, but
        # then we wouldn't have had an excuse to slice an array and
        # create a range
        hours = range(0,bytes.size)


        # Next, we'll graph the data as-is. We need plotlib.
        import os
        import scipy as sp
        from scipy.stats import gamma
        import matplotlib.pyplot as plt
        def plot_data(x, y, models=None, mx=None, ymax=None, fig_idx=None):
            '''
            Plot the data (y) over time (x).

            models, if present, is an array of polynomial models generated
            by polyfit()
            '''
            x = range(0,x[-1])
            plt.figure(figsize=(12,6), dpi=300) # width and height of the plo
            plt.scatter(x, y, s=10)
            plt.title("Bytes per Hour")
            plt.xlabel("Time")
```

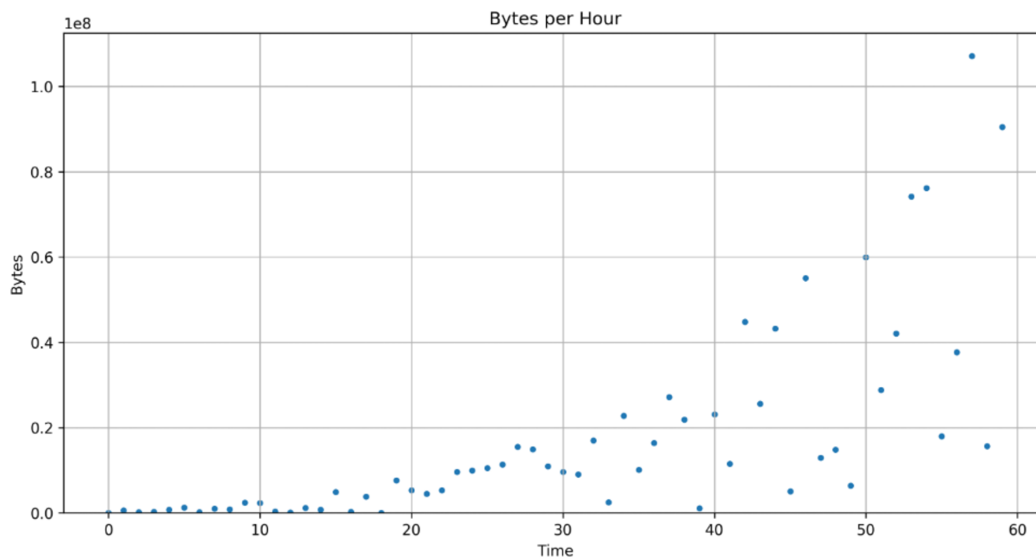

Since we've split the byte data away from the hours that they were associated with, we will need to create something to use as the X axis in any graph that we create that maps bytes against hours. To do this, we ask Python to create a list of numbers that goes from zero to the length of the *bytes* array that we just created. This should give us a list of numbers from 0 through 59. In fact, it really wasn't necessary for us to proceed in this way. We chose to because, otherwise, the hour numbers will be massive timestamps. This approach allows us to use very tame numbers in a very small range. ☺

The part that comes next is some more complex Python code that defines a function that we can use to graph out the values that we have in these arrays. Don't worry about this code at all! Simply view it as a "recipe" that you could copy and paste to experiment with graphing out two-dimensional data by passing it two arrays of the same length. We will see this in action very soon.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Bytes over Time



The next step in our notebook, which isn't actually pictured here, is asking it to plot the *x* and *y* arrays of bytes and hours. The hours are on the X axis and the number of bytes is on the Y axis.

Unsurprisingly, the graph indicates that the number of bytes per hour is increasing over time. An important question is, "How quickly is our volume of data increasing?" Again, one of the reasons that we might care about this is for future planning and forecasting. I'd greatly prefer to be able to predict when we will exceed the current capacity of our network and to plan upgrades ahead of that, rather than wait for the network to become unreliable and sluggish!

Another reason we might want to predict this is to look for indications of threats. Could a massive and unexpected spike in data flow indicate an exfiltration? Could the number of bytes accessed per hour on our file server indicate that ransomware is currently encrypting the file server?

© 2021 David Hoelzer

https://t.me/learningnets

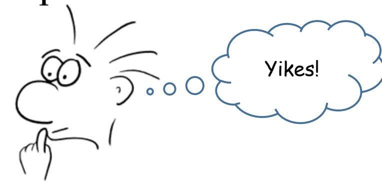
©SANS Institute 2021

Linear Regression

If you had to draw a line that best fit that data, what would it look like?

- Intuitively, you would draw a line that was as close to the middle of all of those points as possible.
- Mathematically, you are trying to limit the error between the line and the points.
- A very common way to do this is called the Mean Squared Error.

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



Intuitively, we could simply try to draw a line that represents an "average" number of bytes of increase. While we could visually estimate where that line should go, a much better approach would be for us to apply some mathematics to the problem. The most common way to solve this problem, linear regression, is to determine what the average *error* is between our line that approximates the average and the actual data. When we say *error* here, all that it means is the difference between the approximation and the actual data.

For example, imagine that our estimate is that hour 10 will have $2.4x10^4$ bytes. Yet, the actual measured number of bytes comes in at $3.6x10^4$. We can work out the difference between these two, which represents the amount of *error*. What we'd really like to do is work out the formula for a line that *minimizes* the error *over the length of the entire line* rather than for a specific point.

We cannot simply add up the amount of error for every point. What if one point is off by +1,000,000, but another point is off by -1,000,000? If we simply add those together, we would inaccurately conclude that there was no error at all!

Instead, we calculate something called the *Mean Squared Error*. We already know that *mean* means average. What does *squared error* mean? We subtract the predicted value (typically represented in machine learning as "y hat") from the measured value (y) at each value of *x* and square the difference.

Why square the difference? Well, if we square the difference, it has the effect of making all of the errors positive! When we take the average, it gives us the average square error for the entire line!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Linear Regression with NumPy

Let's get NumPy to do the hard stuff:

- We create an error function.
  - This is just for us... we don't *need* to do this.
- Then we ask NumPy to find values that best fit our data...
  - Hmmm.. Two values...?
- This has an MSE of 1.6.

```
In [9]: # We need an error function that gives us the sum of the squares of the di
        def error(f, x, y):
            return np.sum((f(x)-y) **2)

In [10]: # Next, we try a naive approach of a straight line approximation of our fu
         # The polyfit function performs this... regression analysis?  We can speci
         # is 1 for a line:

         fp1 = np.polyfit(range(0,len(hours)), bytes, 1)
         fp1

Out[10]: array([   962336.04623507, -10774002.66393442])

In [11]: # These means that f(x) = 2.58462016 * x + 996.506..., or y=mx + b
         # To convert that array into a function, we pass it to np.poly1d, where 1c
         f1 = np.poly1d(fp1)
         error(f1, hours, bytes)

Out[11]: 1.6341676893883622e+16
```


Certainly, you could work to analyze this by hand and work out the parameters for the equation of a line that minimizes the error (has the smallest mean square error). That is tedious. Fortunately, NumPy has a function that will work this out for us!

First, let's define a function that calculates the error for us. Those first few lines will calculate the mean square error across the entire set of values of *x* and *y* using a function that we pass to it. Where can we find a function to use to approximate our line?

NumPy provides the *polyfit* function for just this purpose! *Polyfit* allows us to pass in two arrays along with a *degree*, from which it will calculate the function of that *degree* that best fits the data.

What? What's the *degree*? The degree is the value of the highest exponent used in a polynomial. WHAT? This is getting worse! Polynomial??? Relax. A polynomial, which you learned about in secondary school, simply means "many terms." (It literally means "many names.") For example, the *polynomial* equation "$y = x^2 + 3x - 4$" is of degree 2 because 2 is the highest value of an exponent in the equation. If you recall the formula for a line ($y = mx + b$), this is a first-degree equation.

So, *polyfit* determines what the coefficients for each term in the equation should be. We can convert that into an actual formula using the *poly1d* function and then pass that to our error function. This gives us an absolutely enormous error value!

(In case you're worried about some of these details, relax… remember that we are only introducing the mathematics to give you an *intuitive* feeling for what's happening… in the end, the math won't matter.)

https://t.me/learningnets

©SANS Institute 2021

Two Values?

polyfit() generates an array of two values…

- If you already see where this is going, AWESOME!
- If not, that's ok… Here's where those go:

$$y = mx + b \xrightarrow{\text{yields}} y = 962336x + (-107740002)$$

To really drive this home, let's look at what has been calculated. The *polyfit* function came up with an array of two values. Those two values can be plugged into the standard formula for a line as the coefficients for each term. This means that we now know the *slope* and the *y-intercept* for the line that NumPy thinks minimizes the error (or *loss*) between the approximation and the actual data!

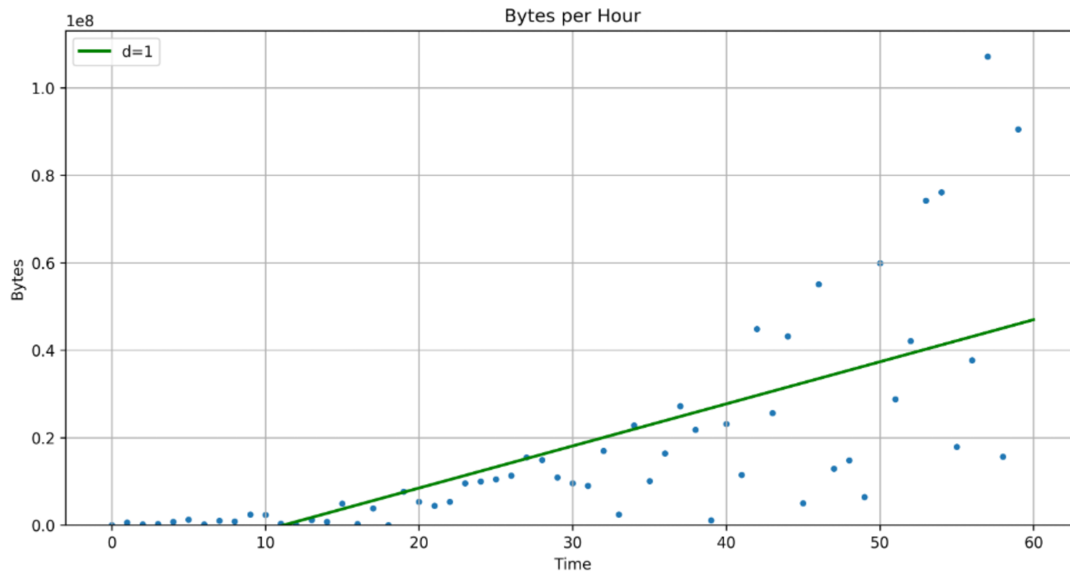
This is headed somewhere really big. We're not ready to tell you where it's heading or how it applies, so let's just recap what you should know right now.

Linear regression is an algorithm that is used to calculate the coefficients that represent the formula for a line that *approximates* the relationship between two values. This line is the *best fit* for the data, and we know that because this line *minimizes the error*, or *loss*, between the approximation and the actual data over the entire length of the line.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Here's How That Looks



What does this look like? It looks like the next graph that the notebook will produce, which is pictured in the slide above. Notice that there are values to the left, though, that don't really seem to be represented by our line. Additionally, as we move further to the right, the actual values seem to be more and more distant from the approximation….

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

What About Machine Learning?

In fact, this is about the most *basic* form of machine learning.

- The “learning” isn’t what you may have though it was!
- The learning is figuring out what coefficients to plug into the formula that best approximates the data that you see.
- This is a basic intuition to grasp.

This example can also illustrate other difficult concepts.

- Overfitting can be difficult to describe, so instead let’s intuit what it implies.

What we have just done really is machine learning! How so? We have asked the computer to automatically follow an algorithm that determines what the correct coefficients are that will minimize the mean squared error over the length of the line!

How is this learning? It is leaning in that the computer worked out the values that produced a line that minimized the error. That still might not do it for you. In fact, this is a very important point that we are at right now. It turns out that the word *learning* in the realm of machine learning doesn’t mean what a typical layperson would think that it means. Learning, in the context of machine learning, in general and linear regression specifically, means that the computer applied an algorithm that allowed it to find the values that minimized the error for the function of the slope of this line. That’s it!

Clearly, this isn’t how most humans would define “learning.” There isn’t any real understanding that has happened. All that has really happened is that the computer followed an algorithm that allowed *the computer to adjust the values of the coefficients*. That in itself is huge... and it is *learning* in this context… but it’s, perhaps, not what you (or your CISO or CEO) might have expected learning to mean.

We have chosen the example of linear regression because it is one of the simplest mathematics applications we can use to illustrate machine learning, but there are other reasons too. Some of those will become apparent as we get deeper into the material covering deep learning and neural networks. There is, however, another term that people wrestle with that we think linear regression can be used to illustrate very simply: *overfitting*.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Another Intuition: Added Complexity

A line is close… a higher order polynomial must be better!

- Let's ask NumPy to figure out the coefficients for a couple of other orders:

```
In [13]: # Clearly this is insufficient.  We can quickly try to fit to other functions by using polyfit to find
         # coefficients for higher order polynomials:

         num_hours = range(0,len(bytes))

         fp2 = sp.polyfit(num_hours, bytes, 2)
         f2 = sp.poly1d(fp2)
         fp3 = sp.polyfit(num_hours,bytes, 3)
         f3 = sp.poly1d(fp3)
         fp4 = sp.polyfit(num_hours,bytes, 15)
         f4 = sp.poly1d(fp4)


         # And let's look at the errors:
         print("f1: ", error(f1, num_hours, bytes))
         print("f2: ",error(f2, num_hours, bytes))
         print("f3: ",error(f3, num_hours, bytes))
         print("f4: ",error(f4, num_hours, bytes))

f1:  1.6341676893883622e+16
f2:  1.3642022432146718e+16
f3:  1.3281158881951732e+16
f4:  1.1988110391641146e+16
```


Before we jump straight to *overfitting*, let's think about this a bit more deeply. What we've done so far is *linear regression*, which by definition uses a straight line. Looking at the data, though, it is likely apparent that a straight line is unlikely to ever represent growth in a reasonable way. Over time, the error will likely become worse and worse.

What if we used higher order equations to attempt to model the data? This seems like a great intuition! What if we tried a second or third order equation? Would that better represent the data?

To try this, we will make use of the *polyfit* function again. Previously, we asked it to create a first order equation (a line). Let's ask it to create a second, a third, and a fifteenth order equation to represent our data! The line wasn't bad, so clearly more or higher orders must be better!

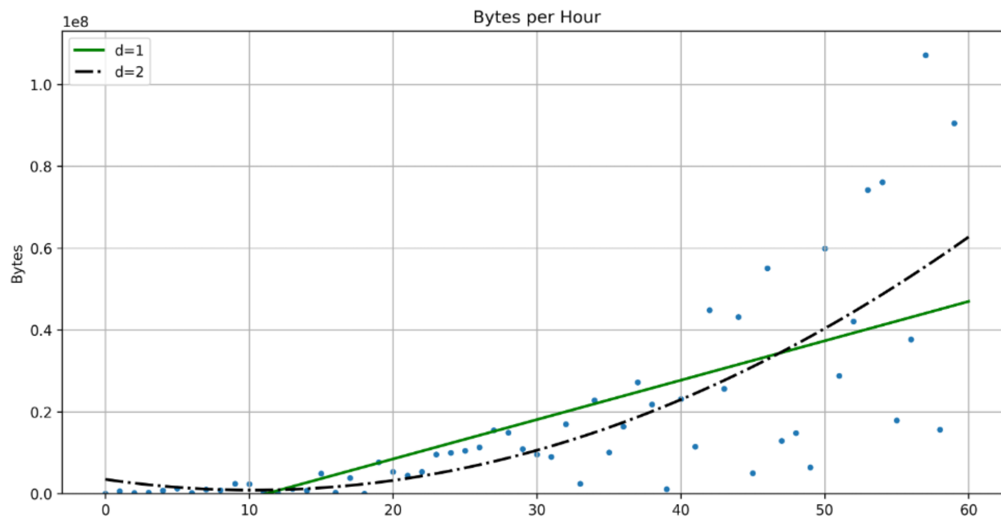
You can see that in the notebook we have started with a numerical approach. Using the same loss function, we ask it to compare the loss or error for our first, second, third, and fifteenth order equations. We can see that the error or loss is definitely going down!! That must be good, right?

Let's have a look at the graphs of these functions.

© 2021 David Hoelzer
https://t.me/learningnets

©SANS Institute 2021

Second Order Polynomial

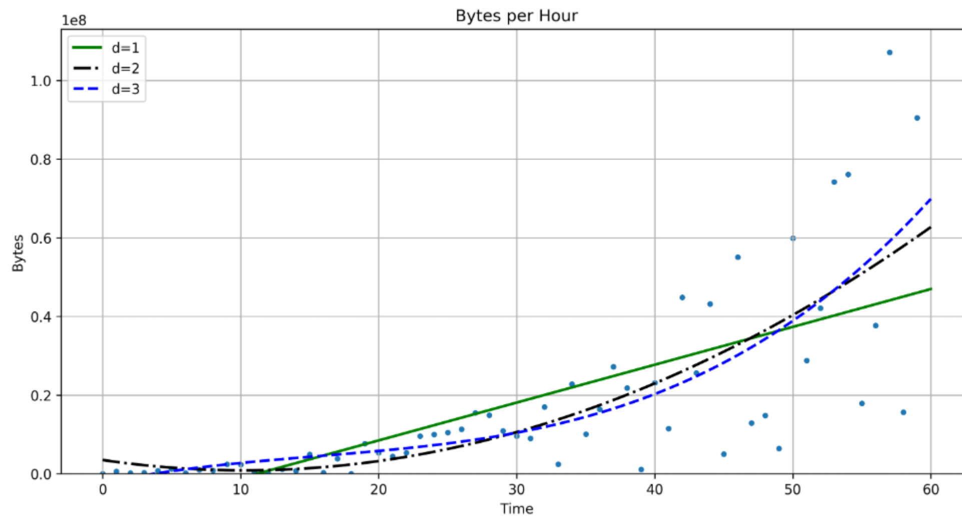


The second order equation definitely looks better. It is now capturing at least some of the data to the left, though it seems a bit high at hour zero. More importantly, we can see that as time goes by, this is a much better approximation of the data over time! It is clearly curving up, following the trend in our actual data!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Third Order Polynomial



If a second order polynomial is better, a third order must be *even better!* In fact, it is! Notice that the dotted line actually drops down toward zero near hour zero, and it more closely follows the increase over time!

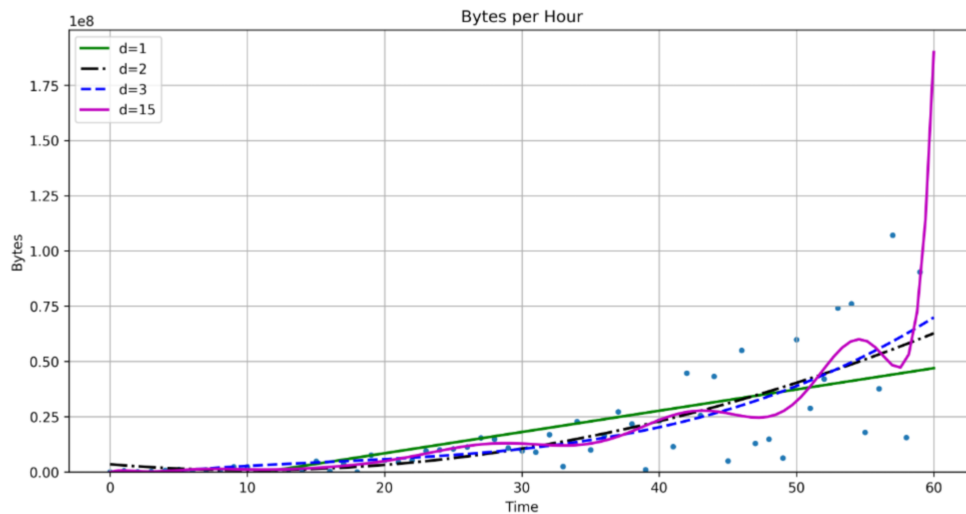
If a third order equation is this much better, let's just crank it way up!

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Second and Third are Better… Fifteenth Must Be Awesome!!!



Our fifteenth order equation, however, looks less promising. Certainly, it appears to be doing good things toward hour zero, but the further we go to the right, our function begins to have oscillations that look less and less right, and then it seems to shoot off to infinity!

As you look at this, recall that, among the functions that we fit to this data, this one had the lowest error overall! Let's zoom into this equation a bit because there is a really important lesson here.

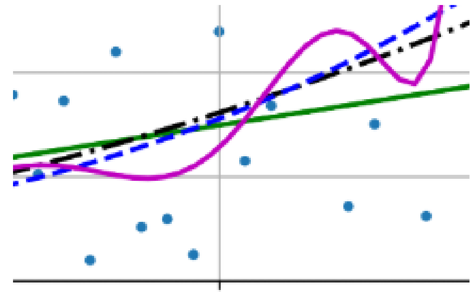
© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Overfitting/Overtraining Intuition

Clearly something's wrong.

- Are we modeling potential behavior?
- Or are we simply trying to hit every single point in the data set?!



When you hear "overfitting," this is exactly what's happening.

- If you check the notebook, the fifteenth order polynomial has the lowest MSE.
- However…

When we zoom way in and look at the behavior of the function, we can get a much better feeling, or intuition, about what's happening. Have you heard the English expression, "Missing the forest for the trees?" In a sense, that's what has happened.

This fifteenth order function really isn't approximating the entirety of the data anymore. Instead, it is really minimizing the error at specific points in our data. In a sense, it is so focused on the trees that it can no longer see the forest, or the overall behavior.

Of course, the function can't "see" anything at all. The behavior that we are seeing, though, is known as *overfitting*. Our function is really just trying to hit every single point that it can rather than trying to approximate the overall behavior of the data. When working with machine learning algorithms, if we *overtrain* or *overfit,* our model will be exceptionally good at properly categorizing or classifying the training data, but it will perform very poorly with real-world data!

This tells us a couple of things. First, overfitting is bad! If we overfit, we can have a model that seems great but works very poorly. In fact, that same model may have worked far better had we *stopped the training process sooner!*

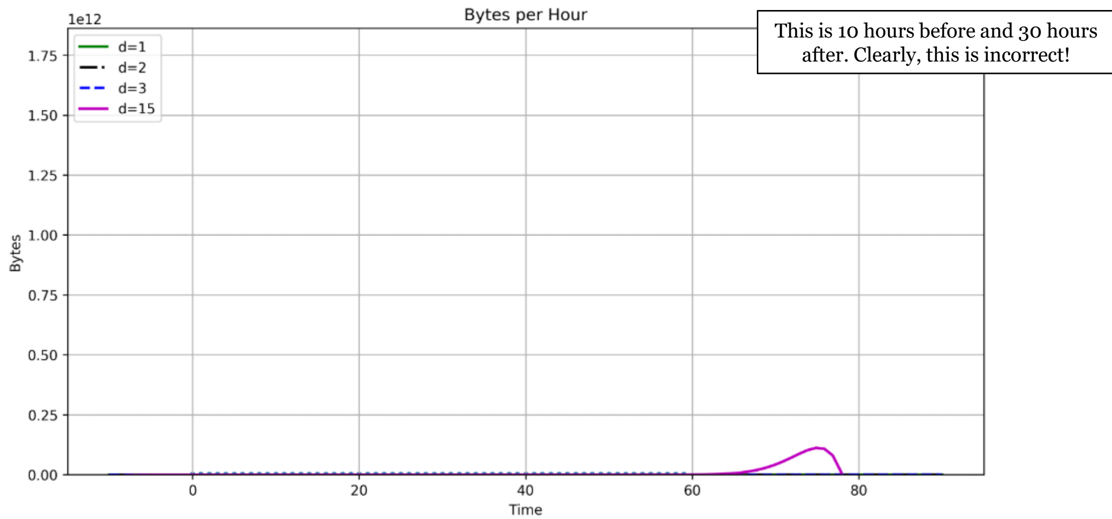
There's another thing this tells us. When we are monitoring the *loss function* or the *error* (these are the same thing), we should probably stop training when the error or loss begins to change very slowly. When the loss number becomes stable, chances are that we are heading into the realm of overfitting.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Overfitting Destroys Predictive Ability



Just to drive this point home, take a look at the predictive ability of our fifteenth order equation. In the short term, it had the lowest overall error, but if we push this out to 30 hours after the sample data, you can see that it predicts that our network data will plummet to zero bytes somewhere around hour 78! This is clearly wrong.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

What We Know/Suspect So Far (1)

- Linear regression finds the coefficients for the equation of a line that best fits a set of data.
  - "Best fits" means "minimizes the error"… in our case, mean squared error.
- Creating higher order equations might approximate the data better.
  - MSE is lower
  - The algorithm is "learning" what the best-fit coefficients are for the polynomial.
- It is easy to cross over from approximating general behavior to simply hitting every single point.
  - This is called "overfitting."
  - The resulting function will match the *training* data very well, but it will perform *very poorly* with any other data.
    - In other words, it is very unreliable when it comes to making *predictions*.

Let's summarize where we are so far and some key takeaways.

- Linear regression is an algorithm or process that is used to attempt to find a line that best fits a set of data points. This means that we are attempting to minimize the error, or the loss, between the actual data and the line itself.

- Using higher order functions can sometimes approximate or fit the data much better. This leads to a lower loss, or error.

- Whether we are using a linear regression or a higher order regression, when the computer uses the algorithm to revise the coefficients used in the function to minimize the loss, this process is called *learning*.

- There is a very real risk that we can cross over from approximating (or fitting) the data to *overfitting*, which simply means that we now fit our training data *so* well that we are unlikely to have actually captured correlations, making the model very unreliable for predicting future events.

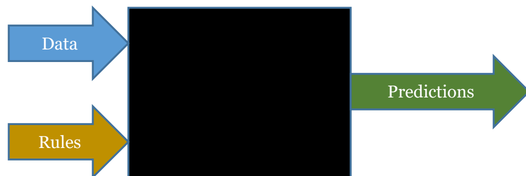
© 2021 David Hoelzer

https://t.me/learningnets

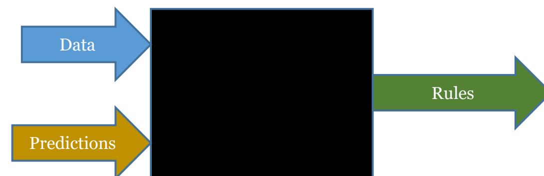
©SANS Institute 2021

What Machine Learning Is Generally About Today

We spent many years doing this... Remember expert systems of the 1990s?



Now we give the algorithm data and known labels, and it produces rules.



SANS SEC503 | Intrusion Detection: In Depth 78

We're now ready to talk about more modern methods of machine learning and how it has changed. In the past (say, the 1990s and early 2000s), machine learning was all about creating a system where we would provide data and a set of rules governing relationships in that data, and the machine (or program) would produce a set of predictions about that data. If you have ever used an "expert system," which were all the rage into 2000 or so, this is how that system worked. A programmer spent a lot of time analyzing the data and wrote a whole bunch of rules, often in the form of a *decision tree*, that was then used by the system to create predictions. The problem with this is that the program cannot "learn" how to handle new data; the programmer must create new rules.

The ideas behind deep learning and neural networks have actually been around for a couple of decades, but it's only since the evolution of high-end graphics cards that we could implement them practically. This might seem entirely unrelated, but the modern approaches to deep learning make heavy use of linear algebra techniques. High-end graphics cards, specifically GPUs (Graphics Processing Units), are highly optimized *vector processors*. That term might sound familiar! Remember, we said that our array of values was really a *multidimensional vector*.

Anyway, the video game industry largely created an enormous market opportunity, resulting in lots of research and development into high-end vector processors for graphics, which make use of *quaternions* (four-dimensional vectors… Okay… A quaternion is something much more complicated and specific than that, but just go with it if you're in the know…☺) for high-speed 3D translations. These devices, however, are *perfect* for handling any kind of general high-dimensional vector arithmetic (linear algebra)!

What does this mean? It means that we could implement algorithms that allow us to feed data and predictions to the algorithm, and the algorithm can give us a set of rules! Truly, an innovation in machine learning!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

What Kind of Rules?

In a linear regression, “rules” means “coefficient values.”

- What if we had a lot of different facts about a problem and we fed those all into a series of simultaneous regressions?
  - Facts = Features
  - Categorization/predictions = Labels
- While the above isn’t 100% accurate, it’s good enough for now because we are effectively doing this:

$$\hat{Y} = \sigma(wX + b)$$

- Of course, there are lots of math things hidden in there... For example, X is the matrix representing all the features and sigma is some nonlinear *activation* function.

When we say that the algorithm will give us rules, what exactly do we mean? You might think, “If it’s round and if it’s red, it’s an apple.” While this is a set of rules, it is a very high-level set of rules. A computer would have to understand what red *means* and what round *means* before rules like this could ever be generated. (Never mind that the computer doesn’t know what an apple is!) Even so, this is effectively what’s happening… but at a mathematics level.

What if we had a lot of facts about some problem and we were able to transform those facts into numbers? What if we used those numbers to create, not one regression analysis, but multiple simultaneous regression analyses? This is at the heart of what deep learning means today.

When we talk about deep learning, we need to provide a set of *features* that will be used for learning. These *features* are the *facts* that we will feed in. More specifically, the features will ultimately be represented as multidimensional vectors (arrays of values).

When we try to transform that data from a set of features to some set of predictions (possibly categories), we are coming up with *labels*. In fact, think of it this way: We are trying to come up with an equation that, when we feed the *features* into it, it will *transform* the features into a *label*. What do we mean by *transform*? Well, remember $y=mx+b$. This formula takes a value x and translates or *transforms* it into a corresponding value for y. That’s it! Except that our vectors might have 10,000 features!

That brings us to the really scary formula in the slide. As scary as it is, there’s definitely something familiar about it! This says that the predictions (Y-hat) are equal to some function, sigma, applied to a coefficient, w (the weights), multiplied by the vector of features plus a value, b (a bias).

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

A Few Last Things

Instead of *telling* you about it, we want you to *try it out*.

- The next lab walks you through the experiment, demonstrating concepts and helping you to understandings and intuitions.
- It is critical that you understand that this is an *empirical* science. How do you know what shape to make your neural network? *You just guess!*
  - I know that sounds fishy, but that's how it is. Remember, this is an evolving discipline!

A *very important* key to all of this is linear algebra (matrices).

- Converting trigonometric translations into quaternions is what allows us to apply highly optimized *vector*-based arithmetic using GPUs for video games.
- *This is the same type of mathematics that most ML uses!*

That still sounds incredibly complicated, but at the heart of it we hope that you can see that it looks very much like the formula for a line! Sure, there are other things happening, and we're now calling it the *weight* instead of the *slope* and the *bias* instead of the *y-intercept*, but that formula is in there! It's true that we are applying this arbitrary thing called *sigma,* which is known as an *activation function*, to add *nonlinearity* (to make it not act like a line), but this is a very valuable thing to recognize! You will see why very soon when we talk about how we minimize error in deep learning.

Before we get to that, there's a super important thing to understand at this point. Since we're dealing with mathematics, it feels like everything should be very formal and there should be well-defined methods for determining precisely how we go about building a machine learning system. *This is false!*

While there might be some function, not yet discovered, from which the large variety of deep learning methods can be derived (Long-Short Term Memory, Convolutional Neural Networks, Recurrent Neural Networks, etc.), *today* this is very much an experimental science. What this means is that we know that a certain design of a neural network works well to solve a problem because of *empirical* means, not because of some formal proof.

Empirical here is a big fancy word that means that we know something works based on our experience and based on something that we can show experimentally… not based on a formal or logical proof. In fact, it is not unusual for data scientists to spend a great deal of time trying to figure out how a modern machine learning algorithm comes to a specific prediction!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Let's Not Learn Linear Algebra

Vector-what?

- A *vector* is simply an ordered list, array, or set of numbers.
- We describe its "shape" and "dimensions."

$$[15.2 \quad -3.1 \quad 0.01]$$

- This is a 1 by 3 vector; that is, 1 dimension (rows) by 3 dimensions (columns).
  - This is also called a "row vector."
  - If it were of shape 3x1, it would be a "column vector" (three rows stacked in a column).
  - If it has more than one row and column, we call it a "matrix" (e.g., 3x4).

We've said several times that all of this makes heavy use of vectors and linear algebra. Linear algebra sounds as though it is the algebra of lines, but that's really not accurate. In fact, in many ways, the word "linear" has nothing to do with this branch of mathematics. More than anything else, linear algebra is a way of representing related (or possibly related) mathematics problems as *matrices* (grids of numbers), vectors, and scalar values. Linear algebra is a set of rules and principles that govern how arithmetic can be performed on these matrices in efficient ways. This is a world of fancy words with specific meanings. This can make the field seem opaque, but as is true in other specialized fields, using unusual words that have very specific definitions is quite valuable; there can be no confusion over what is meant if you understand the word. For example, *quaternion* sounds quite fancy… but all that it means is **a vector having four terms**. The four terms, in computer graphics, are the three-dimensional coordinates with a 1 in the added term.

The real power of linear algebra is that quite complex ideas can be represented as relatively simple matrices. This allows us to apply some very well-defined arithmetic rules, potentially resulting in very powerful computational results that would be far more difficult to accomplish with other methods.

While this isn't a Python class, there's a chance that you might intuitively pick up some Python recipes as we work through this section. This is even more decidedly not a linear algebra class. We definitely will not dig deeply into this topic.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

A Bit More Not-Linear Algebra

A set of vectors taken together is called a *tensor*.

- To be more accurate, a vector is a rank 1 tensor, and a matrix is a rank 2 tensor; these are special cases of the general idea of a tensor.
- That's all we need to know about it, but this should give you some intuition about what *TensorFlow* does!

We represent our "features" with vectors.

- Linear algebraic operations place some requirements on us.
  - For example, all our feature vectors must be of the same *shape* or *dimensions*.
- We often spend far more time preprocessing the data into something that will work with matrix operations than we do building the neural network!

SANS | SEC503 | Intrusion Detection: In Depth 82

Still, there are a couple of other ideas that we're going to introduce to you. Our purpose in doing so is much less to teach you linear algebra and more to familiarize you with terms that you will commonly run into when researching machine learning approaches and reading documentation related to machine learning algorithms and tools.

We already have a feel for what a vector is. It turns out that a collection of vectors has a very fancy name: *tensor*. That's really all that we need to know. A *tensor* is effectively an array of vectors. Said another way, a *tensor* is a matrix.

This term comes up because of a wildly popular Google project called *TensorFlow*. Even with the limited knowledge that we have at this point, this tells us that it's some kind of software that manipulates sets of vectors.

We use this approach because linear algebra, particular the vector capabilities in GPUs that apply linear algebraic functions, allows us to manipulate and transform the data very quickly. Therefore, we will represent our *features* as numeric values that reside within *vectors*.

One of the most important rules in linear algebra is that our matrices typically have to be of the same shape or number of dimensions (this isn't strictly true, but remember, this isn't a linear algebra class!). This means that, if we are going to represent data that has variable lengths, we must somehow manipulate that data so that it is all of the same length. If we can't do this, we can't apply this approach for machine learning! This, though, creates a situation that is very common in machine learning: It often takes far more effort to massage the data into a condition where you can use it than it does to actually create the machine learning model!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Multi-hot Encoding/Vectorization

To get the features to be of the same shape, we will *vectorize* them.

- As it turns out, we almost always want our features to be represented as values between 0 and 1 or -1 and +1.
- Conveniently, there is a "shortcut" encoding that we can use to vectorize many types of data.

Let's build an intuition about it.

- Consider the following data involving three packets where we are using TCP options as features:

Packet 1:	SACK, MSS, WSCALE, Fast Open, NOP
Packet 2:	MSS, WSCALE, MultiPath
Packet 3:	MSS, SACK, Timestamp

One of the approaches to turning variable-length data into static-length vectors is that of *vectorization*, of which *one-hot* and *multi-hot encoding* are special cases.

Vectorization simply means that we are taking the raw data and transforming it into a vector. This naturally implies that we are also converting it into numeric values. In most cases, especially when dealing with deep learning, we will want to convert the data to a value between -1 and 1 (or 0 and 1, which is a subset of that range). This is also known as *normalizing* the data. Really, this just means that we are scaling the data so that the full range of values is represented by some value between -1 and 1.

Let's explore vectorization and *multi-hot encoding* with a practical example. Let's imagine that we want to capture the TCP options in a packet as a set of features. If we look at three different packets, we find that they have different TCP options turned on. Not only are there different options, but the options might even be in a different order. How can we turn these into vectors?

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Vectorization

Packet 1:	SACK, MSS, WSCALE, Fast Open, NOP
Packet 2:	MSS, WSCALE, MultiPath
Packet 3:	MSS, SACK, Timestamp

They aren't all the same shape... and they have different values.

	SACK	MSS	WSCALE	TS	MultiPath	Fast Open
Packet 1	1	1	1	0	0	1
Packet 2	0	1	1	0	1	0
Packet 3	1	1	0	1	0	0

They are now all the same shape... they've been *vectorized*!

What if we were to think of the complete set of possible TCP options as a series of columns in a spreadsheet. Then, for each packet, we set each column to zero if the option isn't present and to one if the option is present.

The advantage of this is that we have taken a variable-length set of options and turned it into a static-length vector! Along the way, we also translated the options into numbers, and those numbers have automatically been *normalized* since they are either zero or one! Any value that is "hot," or on, is represented as a 1, while missing values are represented with a 0. Since multiple options can be turned on at the same time, what we have created is a "multi-hot" representation. It's true that we have lost the order that the options appeared in.

We did mention this other term, *one-hot encoding*. This is very similar, but not the same. In fact, look at the name. Literally, *one* thing will be *hot*. In other words, in the resulting vector, only *one* element will be set to *one*, or *hot*. If we were to one-hot encode the TCP options for the first packet in the slide, we would get the following matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Remember, this represents *just one packet!* Each column represents an option, and each row represents the position of that option in the options list. Note that within each row, *only one element is **hot***.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Learning vs. Understanding

Workbook exercise "IMDB Experiment"

That's enough theory for now. Please follow the directions in your workbook to access the notebook for the IMDB experiment!

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

A Word on Learning

Clearly, "learning" doesn't mean what you thought it meant!

- Learning is really minimizing the error between the training data and the known labels.
  - If we haven't said it already, this is *Supervised Learning* because we are *telling* it what the right answers are in the training data.
  - *Unsupervised Learning* simply means that we do not provide correct labels while training.

Minimizing the error in a linear regression is easy to picture and draw.

- What if it were a 3D graph? The line would become a *plane*.
- What if we had a 4, 5, 10, 15, or 20,000-dimensional tensor?
  - We are now adjusting the slope of a *hyperplane*.
  - The learning process involves *back-propagation*, which is working out the slope (gradient) of the hyperplanes, attempting to find the lowest error (or *global minimum*).
    - This is called *Gradient Descent* and is the most common way we do machine learning today.

The IMDB example that we just worked through is a classic problem in the machine learning space. While the classification of movie reviews may not seem directly applicable, there are some important lessons, and we can definitely come up with useful applications.

For a useful application, consider using an approach very much like this one to create a model that can predict whether or not an email is junk. Certainly, that would be useful! And this approach accomplishes that without writing a signature. We could certainly reach out to a Python neural network from Zeek, passing decoded MIME content through our network to decide if its junk.

Still, there are some really important lessons. One of the most important is that even though we are *learning*, which we clearly understand to mean "automatically updating coefficients to approximate a function that minimizes loss," it is also very clear that our network does not *understand* the movie reviews *at all!* The specific approach that we are using here is a *supervised* approach, which means that we have provided it with pre-classified training data. This approach is called a "Bag of Words" approach because we are simply tracking which words are used, completely disregarding the order that the words are in (which might help with context). Certainly, we could use *word vectors* as a way of preserving the order, but we're not going to expand this particular network.

The other major point to take away is that we have moved from minimizing the loss of a line to *representing the overall loss of our function as the gradient (slope) of the multidimensional hyperplane,* which we attempt to minimize. The slope of a line is the change in y relative to the change in x. The gradient of a surface is its slope, but in more than two dimensions. *gradient descent* is one of the most common approaches used today to minimize the value of the loss function when performing deep learning.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

What We Know/Suspect So Far (2)

- *Deep learning* neural networks operate very much like a linear regression.
  - It's definitely not the same... but the algorithms learn by minimizing the error.
  - We can wrap lots of fancy math names around this:
    - We are using *n-dimensional tensors*, which are effectively acting as a representation of the *multivariate* coefficients of incredibly complex *hyperplanes*.
    - The *gradient descent* technique is used to find a *global minimum* of the error function across these *hyperplanes* by using *back-propagation* to derive the *partial derivatives* (just think "slopes relative to other features in the tensor").
- The machine makes good predictions, but definitely does not *understand*.
  - Really, we've produced a very complex statistical model that determines the likelihood that something is true based on the terms that happen to appear in the data.

This means that, at a very high level, deep learning neural networks really operate very much like linear regression. Rather than looking at fitting to a single line, however, we are passing in multidimensional tensors with perhaps tens of thousands of values, which represent *hyperplanes*. What's a hyperplane? Well, we know what a line is. If we take a line and extend it in the *Z* axis (third dimension), we get a *plane,* that is, a flat surface extending in the *x* and *y* dimensions. A hyperplane is that three-dimensional plane now projected into four or more dimensions! What does it look like? No one has any idea. ☺

Still, the point is that deep learning neural networks are intuitively an n-dimensional application of multiple simultaneous high-order regressions being applied to the data all at the same time. When you read about *back-propagation*, this refers to determining the current gradient of the loss function, which can be used to *learn* (as in, adjust the coefficients or *weight* vectors and *biases*), progressively improving the predictions that can be made!

This seems a good place to point out the literal meaning of the term "artificial intelligence." Really, this means "fake intelligence." While this isn't what people normally *mean* when they say this term (people usually mean computer intelligences taking over the world because they're smarter than us), whenever we can use an algorithm that *appears* to do something that takes intelligence, we are looking at artificial intelligence. Obviously, it's possible for this term to be used very loosely on products!

What we've seen is that even a model that can make very nearly perfect predictions has no real *understanding* of what's happening. This problem sits at the crux of modern research in the machine learning and artificial intelligence space.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

The IMDB example is interesting, and we've certainly managed to learn quite a few very fancy sounding terms… but how can we apply this to our networks? Let's experiment!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Practical Applications and Experiments

Movie reviews are nice, but how can we use this?

- Let's create a model that can tell IPv4 packets from non-IPv4 packets.
  - That's a binary classification problem.
  - It's trivial, but still... the model will differentiate between them without us telling it how to!
- Categorical classification can be much more interesting.
- Could we take this further?
  - How about a classifier that can distinguish operating systems based on TCP options!

Let's kill two birds with one stone. We want to see a way that we can directly translate what we've done so far to network-related activities.

We are going to try to develop a proof-of-concept model that can tell the difference between IP packets and packets that are not IP. While this is certainly a trivial problem, it has the potential to sit at the foundation of a much more complex model. For example, what if we had a model that could not only tell IP from not IP, but it could distinguish common transport layer protocols? What if we took that even further and had models that were able to determine if something that was TCP and appeared to be HTTP because of the port that it is on *behaves* like HTTP? Now that would be useful!

We won't go quite that far… we only have a limited amount of time. However, we will absolutely be covering everything that you need to know to be able to build a set of models that can do precisely this! If you want help taking things to that level, SEC595 is definitely the place to go.

One of the last things that we will do is attempt to perform real-time anomaly detection for protocols on a network. That's something that we could use *tomorrow.*

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Is It IP?

Workbook exercise "IPv4 Binary Classification"

Please follow the directions in the workbook under “IPv4 Binary Classification,” which will give you directions on how to access the associated Jupyter notebook.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Thoughtful Discussion

Don't minimize what this network does.

- We provided frame headers as bytes.
- We didn't tell the network where any of the header fields are.
- It still works out to identify IPv4 packets pretty well!

What could go wrong?

- Your training data *really* matters... at least as much as feature selection.
- For a binary problem, you'd like your training/test data to be 50/50.
  - Extend that for the number of classifications that you have.
- Feature selection matters!
  - In this case, we gave it the first 14 bytes of every frame, IP or not.
  - Could our model inadvertently correlate the addresses rather than the "0x0800" we use?

Let's think this over. What we've built isn't really that "smart," but it's still important. You and I can look at these packets and very readily decide which protocol they are. We have tools that can be directed to select packets based on the protocol in use! But in this case, our model is able to tell the difference between protocols *without us ever defining where the protocol header values are or what they mean!* That's huge!

There are definitely things that can go wrong. One of the critical parts of machine learning is figuring out what the features should be (known as *feature selection*). For example, if we provide our network with features that include MAC addresses, the model could easily develop false correlations that have nothing to do with the protocol in use! This could mean that we've selected our features poorly… but it could also mean that our training data is insufficient!

Generally, the more training data you can provide, the better. That training data needs to be representative of real-world conditions. If we were creating a model that is intended to generally predict protocols on *any* network, we would either need to reject the addresses as features (since every network will be different) or include training data that adequately covers all the addresses likely to be seen in the real world!

It's also important that our training data is well *balanced*. This means that we don't want it skewed toward any particular outcome. If we are training a neural network for binary classification, we really want a 50/50 mix in our training data.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Improving That Model

Can you improve that model? How?

- Ultimately, we've given the model lots of data that is unrelated to our outcome.
- The model has found false correlations, likely involving the MAC addresses.
  - This is strongly influenced by our training data.
  - If we had lots of data from lots of networks, our network would be *much* better.
- Our main problem here is feature selection.
  - Sometimes we don't know what the features are, so we rely on the system to find them.
  - In this case, we should tell it because we *definitely know* where the features are.

Your mission:

- Adjust that network to narrowly use the ethertype as the feature.

How could we improve the model that we've built so far? We've given the model a great deal of data, not all of which is related to the problem at hand. Should we reduce that? Or could it turn out that there actually are important correlations in there that we are completely unaware of. For example, imagine that we started passing IPv6 data through a network. If we weren't aware that multicast is really a layer 2 protocol, our network might discover the relationship between the destination MAC addresses that would be found in IPv6 multicast traffic, actually learning something that we weren't aware of!

Still, it does seem that we could define this network more reliably if we were more selective about our features. Let's see if we can do that. Can we modify this network so that it uses only the ethertype in the frame to determine the type of traffic that is present?

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Is It IP? (Take Two)

Workbook exercise “Binary Classification with Better Features"

This page intentionally left blank.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Something More Practical

Let's build a model that classifies the embedded protocol.

- You generally need to have the same number of output neurons as you have categories.

This is still trivial since we can just look at offset 9.

- Oh, the places you will go!
  - Use the first 20 bytes of data payload as your features.
  - Use the *bits* in the first 20 bytes of data as your features! (Only 160 dimensions!)
  - Pull NetFlow statistics data for the number of connections, bytes, and timing for connections.
    - Train to identify web browsing, file downloads, video watching, exfiltration, email...
  - Create a Zeek script that outputs statistics on hosts periodically that can be fed to a network.
    - Remember, Zeek has Python bindings and can execute *any* external code you want it to!

With this foundation, we're ready to tackle bigger, more practical problems. Let's try to build a classification system that determines what the embedded protocol is according to the IP header! For now, we could look directly at offset 9… but think about other ways to accomplish this task if we have a great deal more data!

What if we used all 20 bytes of the IP header as our features?

What if we took the 20 bytes in the IP header and exploded them out into a 160-dimension row vector that represents each of the bits in the header! This would allow our network to potentially learn really granular things about fields like the fragmentation bits and the differentiated services byte! If we were looking at TCP, it should trivialize the task of differentiating the TCP flags!

What if we periodically used *rwflowstats* to pull data on the number of connections, along with the number of bytes and the timing of connections, and built a network that could classify different types of network behavior? Could we even build something in Zeek that periodically calls a model, or perhaps there is a model running full time that subscribes to a broker channel and reacts to behaviors in our network?

Oh, the places you will go!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Classifying the Unknown

In the examples so far, we know or can guess how many categories there should be.

- Are there more operating systems than Windows (7, 8, 10, 2008, 2012), Linux, FreeBSD, Solaris, and macOS? How many more can you name?
- Are there more behaviors than web browsing, watching videos, sending email, and exfiltration data?
- Are there more than a handful of application protocols? How many are there?

We could use *unsupervised learning* to create a model that identifies anomalies or *outliers*.

- This can be challenging and requires massive amounts of data.
- Could we accomplish something similar with a multiple classification system?
  - Let's try!!! (In the next lab... ☺)

It might have surprised you that our neural network performed so poorly on that problem! As an educated guess, we'd say that this is most likely because of one of a couple of problems, or perhaps a combination of problems. These are the problems in the order of what we suspect to be most likely:

1. We really need vastly more training data.
2. We need to figure out some other important and relevant features.
3. A dense network just isn't good at this; likely a recurrent or convolutional network would perform better on this particular problem.

There's yet another problem with both these approaches. Have you already figured out what it is? All of these require us to tell the system how many different categories there are! This leads to an important realization: We need some way to classify something as a thing that our network has no idea how to classify!

One way to approach this problem is to feed one model from another model. For example, perhaps we create a decision tree that first asks, "Is this something that we have seen before?" If the answer is yes, it can then be fed to a neural network that tries to classify the system. If the answer is no, it's time for human intervention and more training!

Another, related possibility is an unsupervised approach. Remember that the difference between supervised and unsupervised is whether or not we provide labeled data for training. One of the approaches that seems most appropriate (to us) is Stochastic Outlier Selection, which goes back to the 1990s (if not earlier)! Unfortunately, this requires massive amounts of data, which is why most systems will require that you allow them to learn about your network for a month or more.

Could we accomplish something similar using far less data? Maybe. What if we created a model that could classify data into known protocol classifications? This is a tractable problem, and we might be able to leverage it to find the unusual!

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Finding Different

Workbook exercise “Finding Different"

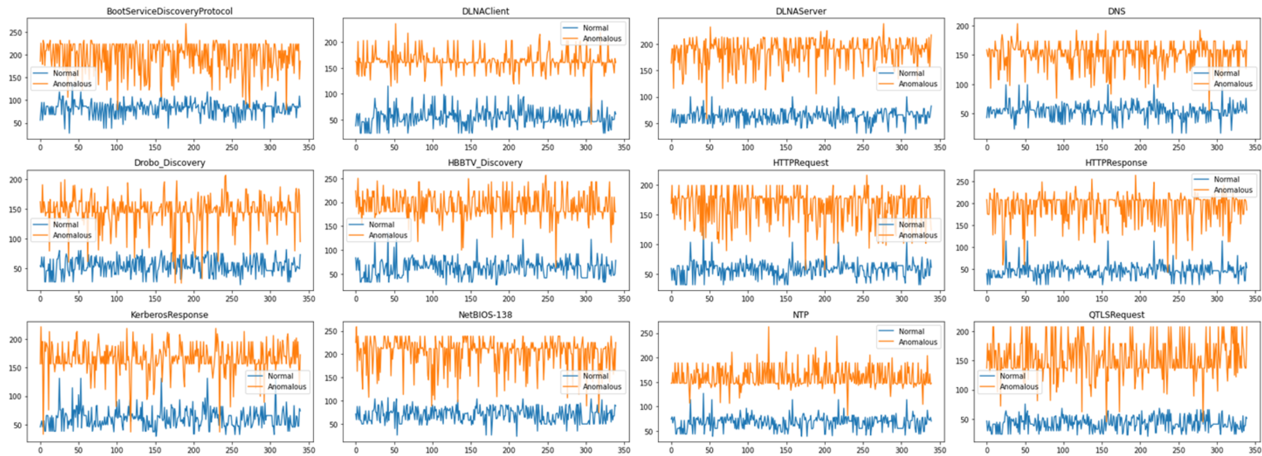
Let’s experiment a bit with a neural network that identifies outlier protocols!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Making This Better

A set of autoencoders makes this incredibly accurate.



What we've built in class with this last experiment is something that you can take and use tomorrow. We've been asked, "Have you tried this on a live production network?" Absolutely. Will it have false positives? Will it have false negatives? Yes to both. However, even something as simple as this has proven to be incredibly useful in finding previously unnoticed protocols running on networks that we monitor.

A more advanced version of this approach using a slightly different technique is the idea of an autoencoder. An autoencoder is sometimes described as an unsupervised learning technique, though we would describe it as self-supervised. When we train an autoencoder, it becomes very good at reconstructing whatever it has been trained to analyze. For an easy-to-understand example, we use autoencoders to take noisy or grainy video, or an old photo with missing pieces, and reconstruct a good facsimile of the missing bits.

In a similar way, if we train an autoencoder to identify SSH traffic and then give it NTP traffic, we can observe a measurable reconstruction error that is significantly higher than how SSH traffic is interpreted. Consider the image in the slide. Here we show twelve autoencoders, each of which has been trained to identify a specific application protocol. The blue line (the lower line) shows the reconstruction loss when we ask the each autoencoder to analyze data for which it has been trained. The orange line (the upper line) shows the reconstruction loss for any other protocol. You can see that these are easily linearly separable, making this incredibly accurate for protocol identification!

When we group these together into a single solution, we now have a machine learning tool that can quickly identify whether something is known or not, generating very useful real-time alerts.

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Important Conclusion

These experiments represent what are being sold as products *today*.

- Tinkering with and discovering a useful model can be time consuming.
- The training data and features *really matter*.
  - If not done very well, the model is useless on other networks.
  - Even so, the more general the model, the less "sensitive" it is.
    - Otherwise, we would have to train it on massive amounts of data from *our network!*

All commercial systems, when finding something that doesn't fit the current model, rely on human analysts to manually classify the data.

- This is often hidden between terms like, "Sent to the cloud for analysis."
- Misclassification is a real possibility.
- The model must learn over time.

We've covered a great deal of material in this short section. In fact, we've covered much of what you would expect to learn in a full semester class on machine learning just in this section!

It's our hope that this gives you enough of an intuitive feel for how machine learning works that you can do one or more of the following:

- Accurately interpret marketing claims about AI and machine learning products.
- Communicate intelligently and usefully with your decision makers about how to invest in these technologies.
- See ways that you can possibly begin to apply machine learning in your network today.

Currently, all the commercial systems that we have reviewed are effectively training networks and decision trees, but if those networks or trees find things that they don't understand, they alert a human. Those humans *must* understand the network deeply or they cannot make good supervision decisions for the algorithms. Perhaps most importantly, we have seen examples of a network having a high degree of confidence that it has classified something correctly… but it was obviously wrong. These technologies are wonderful, but they aren't silver bullets!

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Learning Over Time

You have taken a few first steps.

- We hope that this gives you really cool ideas of what to do when you get home.
  - If you want a deeper dive into building practical ML solutions, ***"Applied AI/Machine Learning for Cybersecurity Professionals"*** (SEC595) could be the answer!

"I want to characterize *current* behavior on my network!"

- Take what you've learned so far and research:
  - *Long Short-Term Memory networks,* which are generally excellent for behaviors over time
  - *Recurrent neural networks* operate differently from LSTMs but are also excellent for data of a sequential nature (like language and audio).
  - *Convolutional Neural Networks* are typically used for image and video analysis.
    - These *might* be very relevant for network behavior since they are especially good at finding patterns that "move."
      - Finding cats in pictures, even though the cats could be in different positions or places
      - To us, this seems potentially very applicable to network behaviors.

We'd encourage you to hit the ground running when you get back home. In order to do really good research with these tools, you absolutely need to have data. We'd encourage you to capture every packet that you can! Is NetFlow already deployed? Do you have the data going into a repository? If you don't, make that happen! The more data you have, the more intelligence you can begin to pull out of that data.

Especially for network-based behaviors, the next step in your research should likely be *Long Short-Term Memory* networks. These tend to be very useful with data that can be represented as a time series, which is certainly how communication functions. *Recurrent neural networks* similarly have good outcomes with sequential data.

You might even decide to go more deeply into this space. Imagine creating an AI solution that not only classifies data types but also classifies user behavior… perhaps even categorizing the videos that your employees are watching on YouTube so that you, too, can find out where the best cat videos are!

If you like the approach taken here but want to engage in a deeper exploration of practical applications in information security, you might look for the new SANS video-based course, ***"Applied Machine Learning for Information Security."***

Clearly, there are incredibly powerful possible applications. It's up to you to decide what you need to know!

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

Summary

Machine learning and AI aren't what people think they are.

- They are simply statistical models applied in interesting ways.
- Figuring out how to preprocess the data is often the most difficult part.

Visualization is useful, but only if it's meaningful.

- There are many simple tools that create beautiful network visualizations.
  - Most are of only limited value.
  - Rarely do they help you to find the unknown.
- Being able to create your own visualization can be very powerful.
  - Allows us to be data driven rather than alert driven

This page intentionally left blank.

© 2021 David Hoelzer
https://t.me/learningnets
Technet24

©SANS Institute 2021

Bootcamp!

Scenario 1, Real-World Scenarios
Wrap up any remaining Bootcamp work from sections 1-5.

For our final bootcamp, we have a real-world scenario that we'd like you to work through. You will find this at the end of the Section 5 workbook material.

There are several labs in the workbook labeled "Real World Scenario." Your assignment is the first of these. The remaining scenarios are provided for your enjoyment (and possibly exam preparation), but need not be completed before proceeding to the capstone.

You should also use this time to work through and complete any outstanding bootcamp questions on the score server (other than extra credit questions).

© 2021 David Hoelzer

https://t.me/learningnets

©SANS Institute 2021

COURSE RESOURCES AND CONTACT INFORMATION

AUTHOR CONTACT
David Hoelzer
dhoelzer@enclaveforensics.com
BEST CONTACT: The Slack Channel!

SANS INSTITUTE
11200 Rockville Pike, Suite 200
N. Bethesda, MD 20852
301.654.SANS(7267)

#SHOWMETHEPACKETS
https://www.showmethepackets.com

SANS EMAIL
GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

© 2021 David Hoelzer
https://t.me/learningnets
Technet24