572.4

Commercial Tools, Wireless, and Full-Packet Hunting

FOR572.4 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response

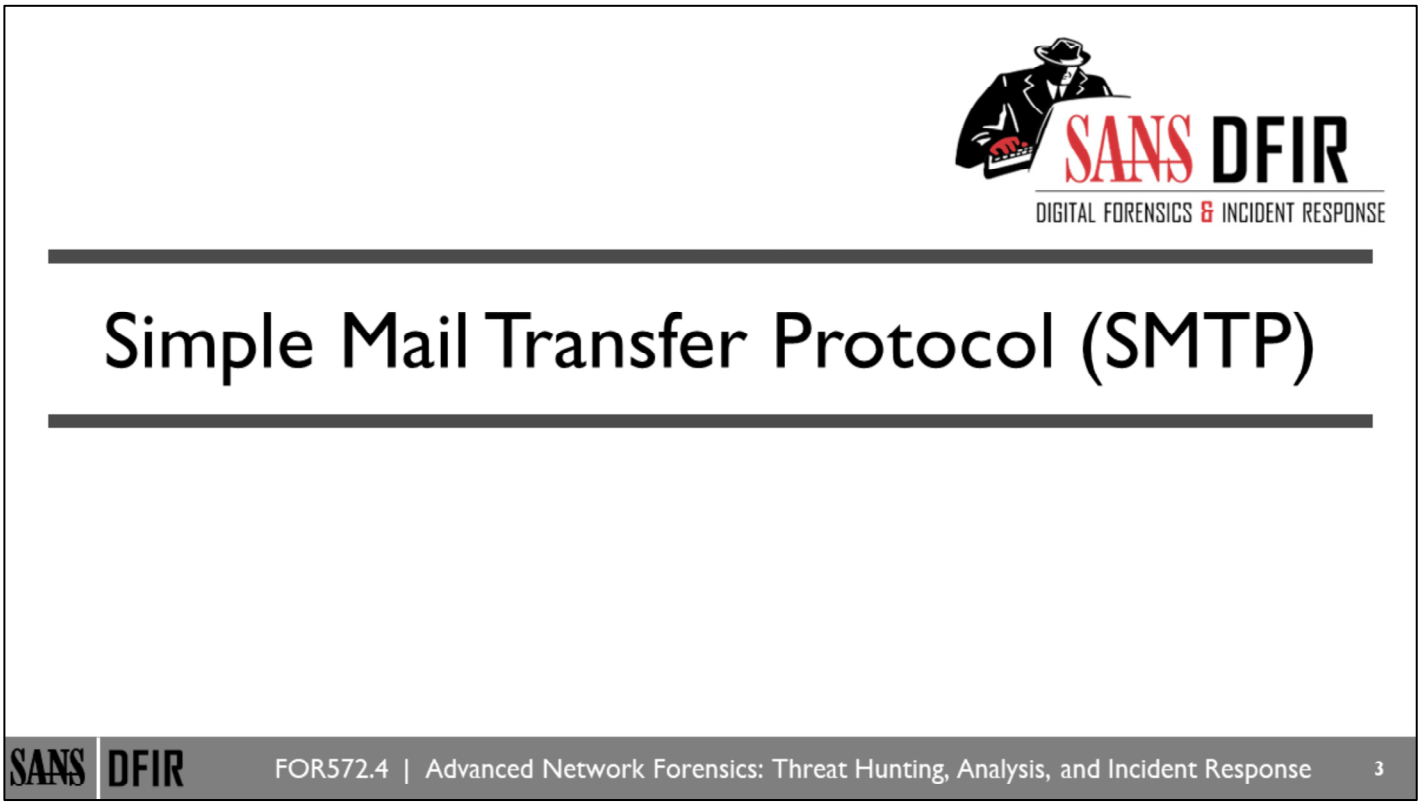
Commercial Tools, Wireless, and Full-Packet Hunting

Authors:
Phil Hagen, Lewes Technology Consulting, LLC
phil@lewestech.com | @PhilHagen
Mat Oldham
mat.oldham@gmail.com | @roujisecurity

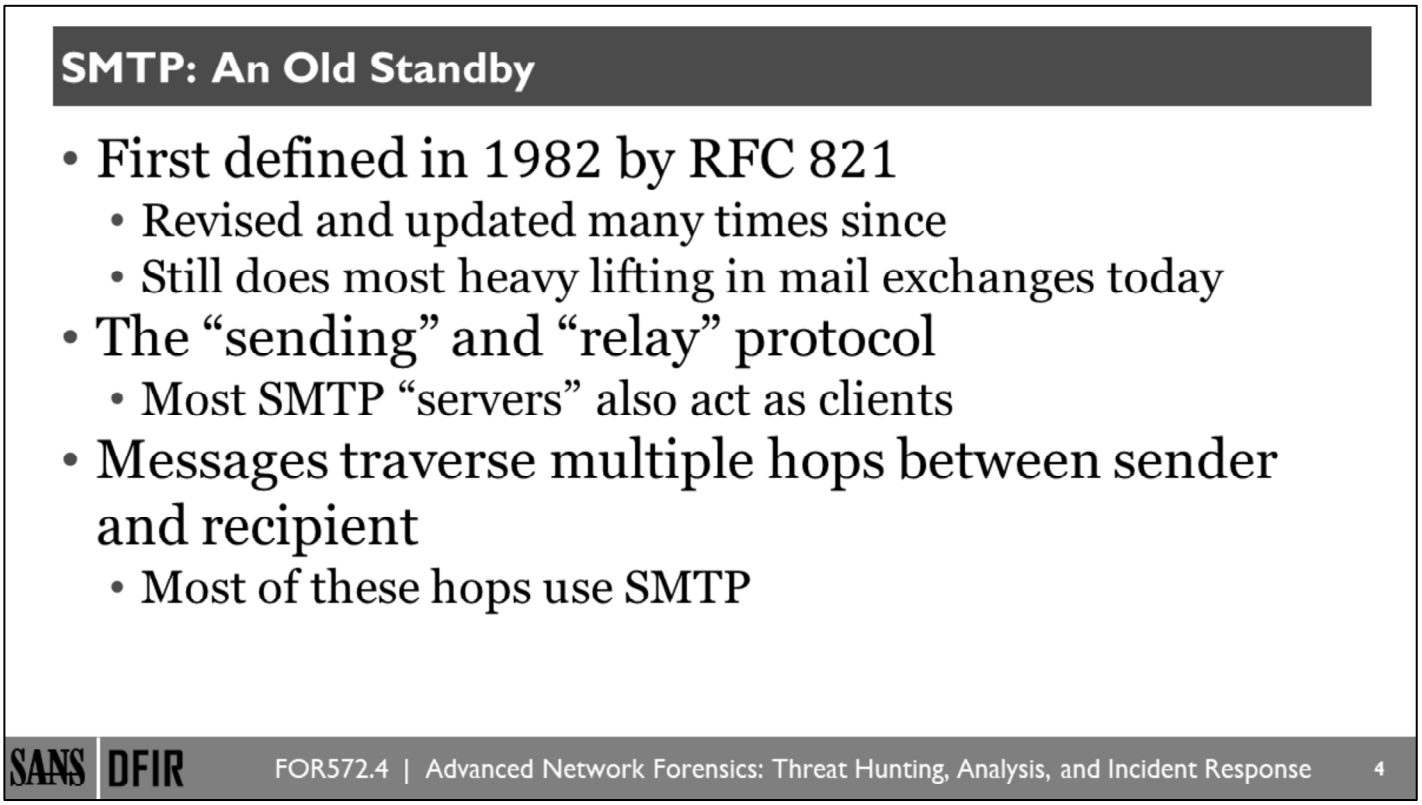
bf18bea88fa3f65fb487b45e25ee99ea68e14d7a2073a53e91f11bbdd5a39dfb

TABLE OF CONTENTS	PAGE

This page intentionally left blank.



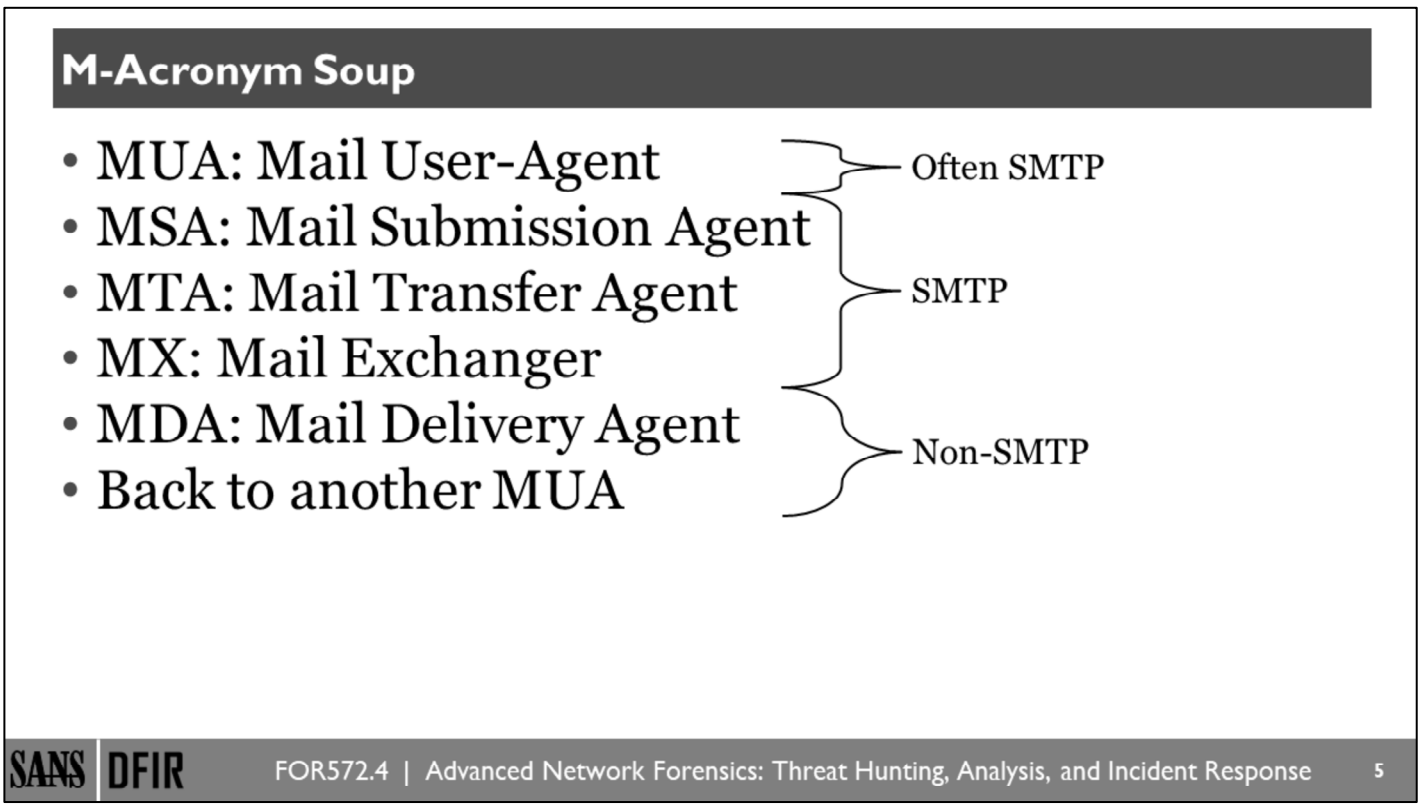
This page intentionally left blank.



Email is one of the oldest and most lasting, ubiquitous protocols on the Internet. What started in the early days of ARPAnet remains a fundamental protocol used to send messages between humans and systems around the world. Simple Mail Transfer Protocol, or SMTP, is the "sending" and "relay" protocol that ensures email messages arrive at their intended destinations.

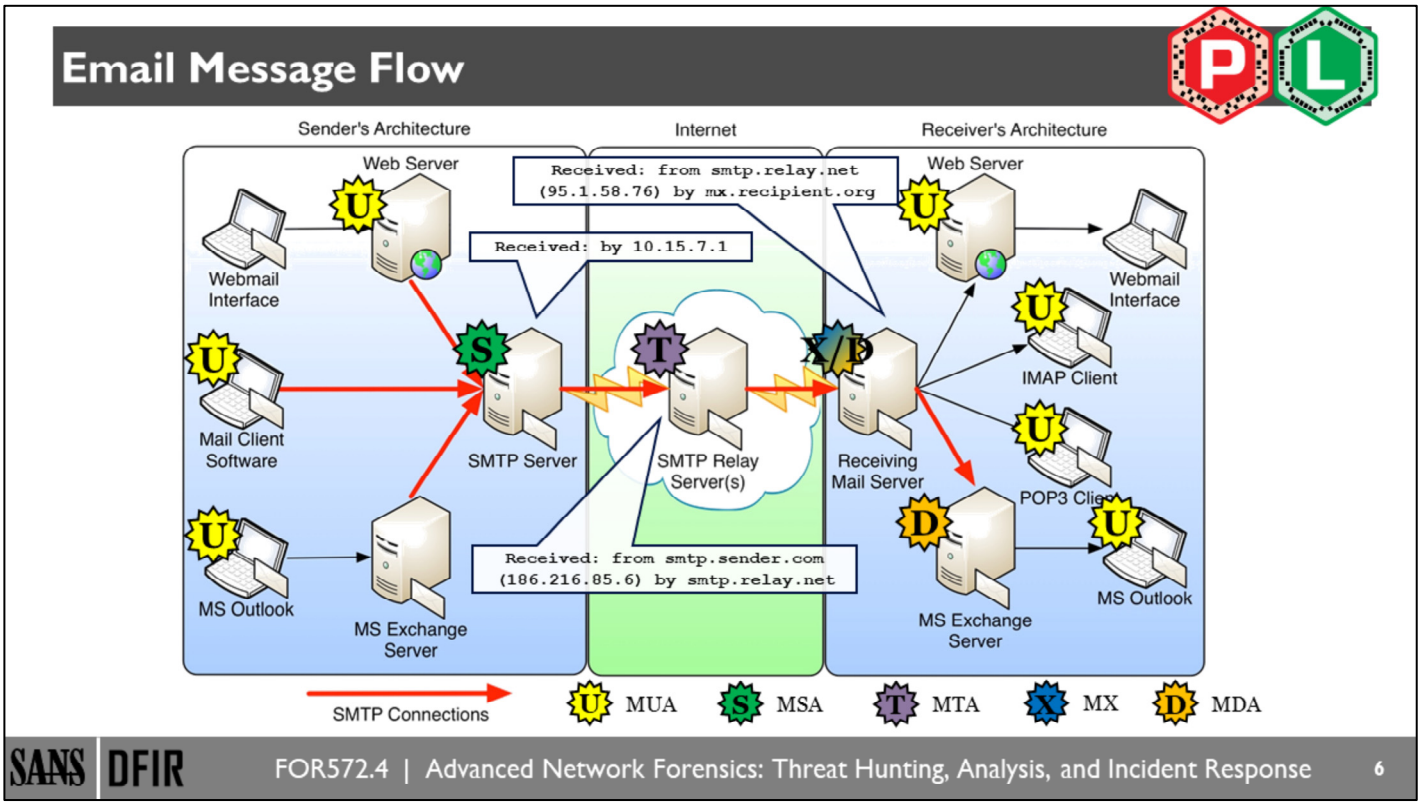
Just as a piece of paper mail or a package crosses multiple mail stations, delivery trucks, or other physical points on its destination, email traverses a series of hops between the originator clicking "Send" and the recipient seeing it arrive in their Inbox. There are a number of roles that different servers and server processes fill during this end-to-end process. As we'll see next, SMTP takes care of most of them. Even in a webmail model, the HTTP-based webmail application running on the server conducts SMTP transactions to put an email message into play.

In this "relay" role, it's important to recognize that an SMTP server then becomes an SMTP client when passing the email message on to the next system.



Each step in the overall end-to-end SMTP process is performed by a specific piece of client or server software, identified by the following acronyms:

- **Mail User-Agent (MUA)**: The client software an individual uses to send email. For example, Outlook, macOS's Mail.app, etc.
- **Mail Submission Agent (MSA)**: The server software that receives a message after the user sends it in the MUA. The sending user's MUA connects directly to the MSA and submits the message into the overall SMTP process.
- **Mail Transfer Agent (MTA)**: The server software that passes the message along to other servers (MTAs) in the series of hops between sender and receiver. There can be (and often are) multiple MTAs involved in the transfer of each email message. MTAs use SMTP in each step of relaying the message along the delivery path. An MTA is both an SMTP client and server in these steps.
- **Mail Exchanger (MX)**: The system identified as a responsible receiver for mail sent to a given hostname or domain. This system is most often designated in a DNS record with the "MX" record type. Each destination host or domain can have multiple MXes for redundancy and load balancing purposes. The MX serves as a "last-step" MTA in this role, using an SMTP connection.
- **Mail Delivery Agency (MDA)**: The server software that provides mail messages to a user after successful authentication. This is not an SMTP step, but rather uses other protocols such as Post Office Protocol (POP3), Internet Message Access Protocol (IMAP), or Messaging Application Programming Interface (MAPI). The receiving user's own MUA makes a connection to the MDA for retrieval of mail data.



Here, you can see examples of several common email paths. Although there are too many possible configurations to detail here, these cover those you'd expect to see in most environments.

Note that although some clients (MUAs) such as the webmail and Outlook systems don't natively use SMTP to send email messages, their immediate upstream systems (MSA) pass sent messages via SMTP-speaking paths very soon into the message's life cycle. As soon as the message is routed via the Internet, SMTP is the name of the game. Usually, one or more relays (MTAs) will receive a message, and then pass it along based on the preconfigured routing rules. Finally, the designated recipient server (MX) receives the message and places it into the receiving user's mailbox (MDA). At the other end of the process, the user's MUA receives and displays the message. (Unless they ignore or delete it…)

What's interesting to note is that at each stage of the transmission, servers generally add a message header indicating the server's hostname, date, and time the message was processed, and some other details that can be valuable in determining the path a message took on its journey from sender to recipient. Of course, these headers are only required by RFC and not law—so some servers may alter the existing or expected header values.

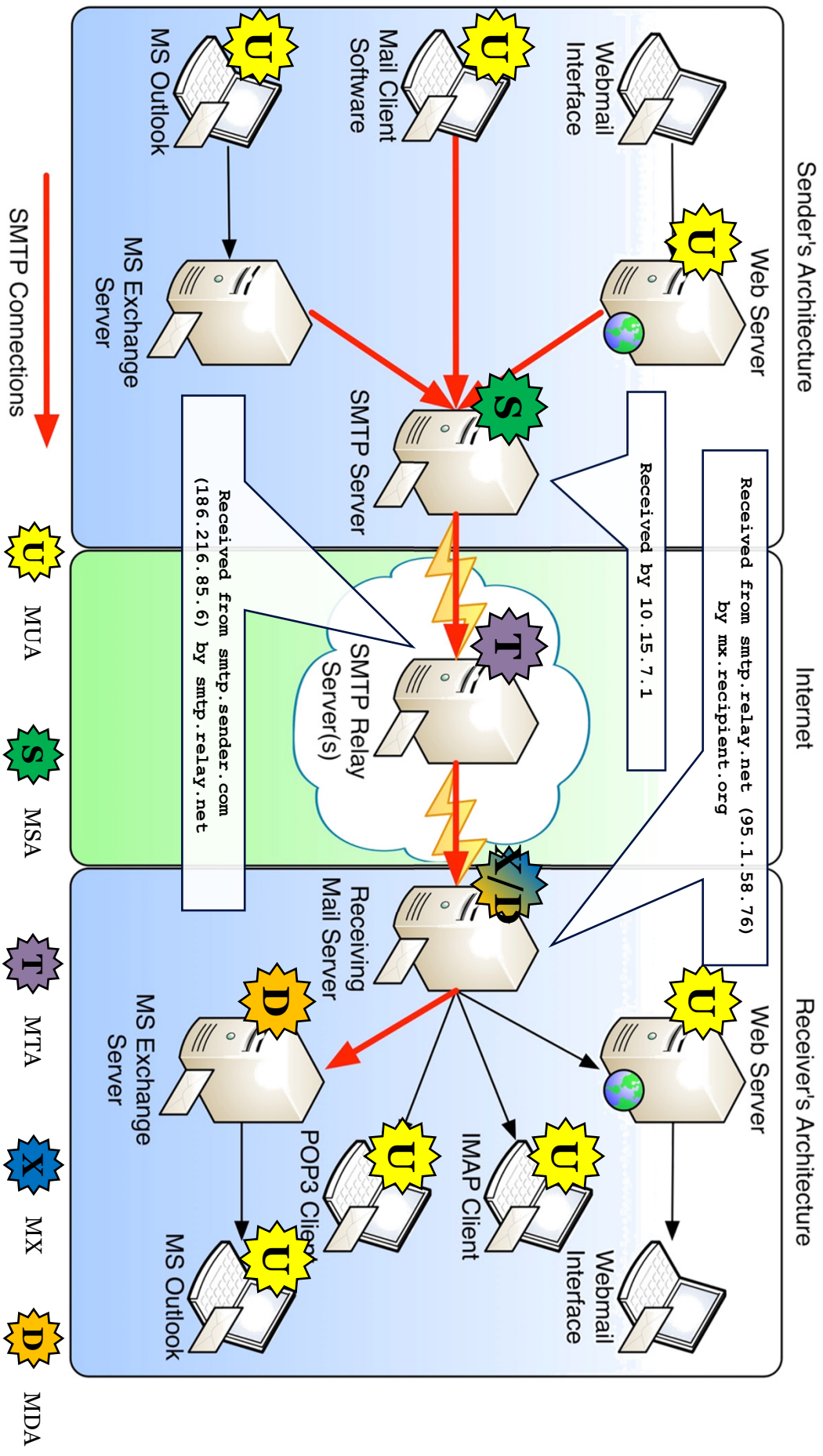
An example of a message a Gmail user sent to the SANS DFIR mailing list is on the next page. The headers are typically in reverse chronological order. Therefore, the last header indicates that the originator used the web interface to send the message, with each subsequent node adding its identity as well as the previous node. This establishes a clear path that the message took, including both hostnames and corresponding IP addresses (public and private). A second set of sample headers reflects a spam message that was apparently sent with the Gmail API.

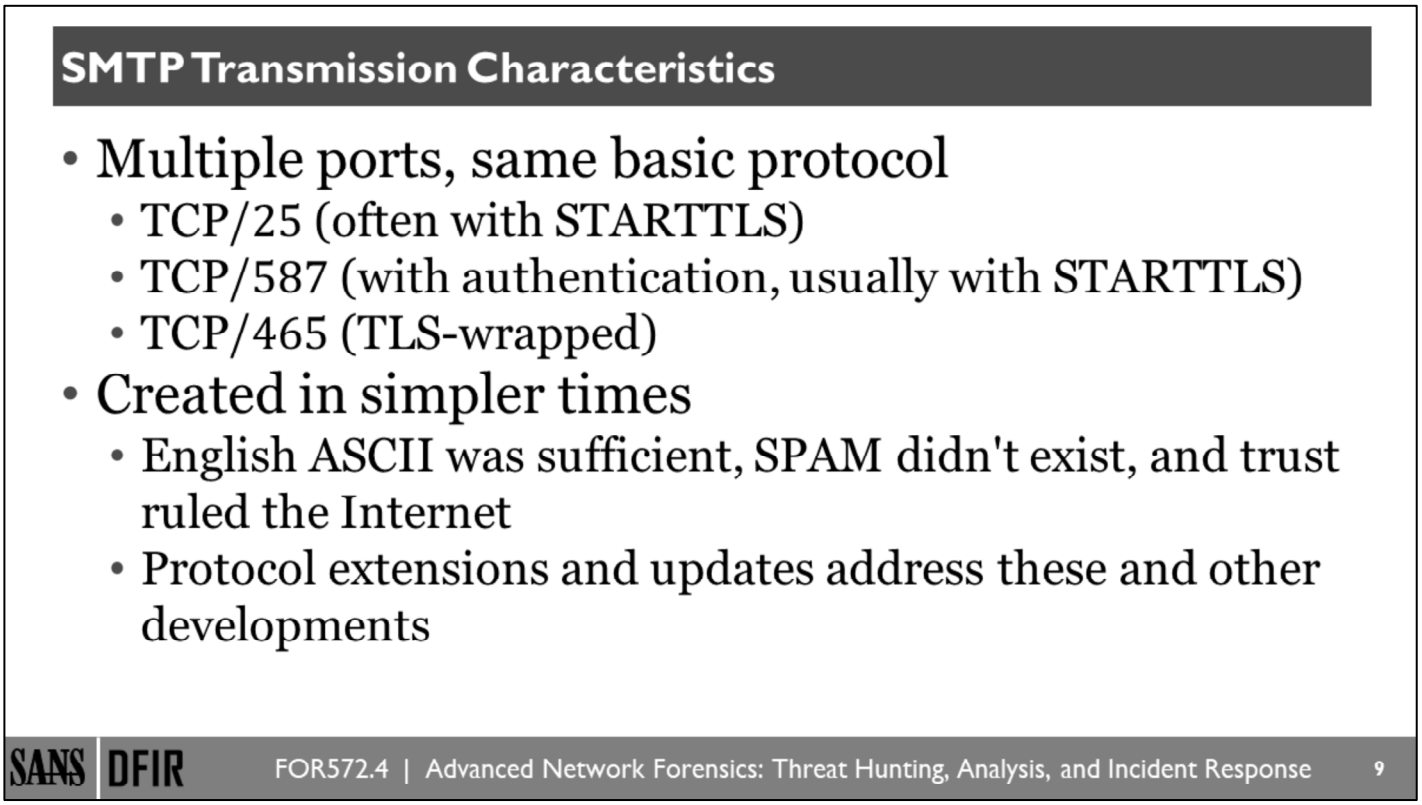
Making sense of these headers, specifically the "`Received:`" data points, can be confusing. Fortunately, tools such as the G Suite Messageheader web app[1] can visualize them, provided OPSEC is appropriately addressed.

References:

[1] https://for572.com/dvg9l

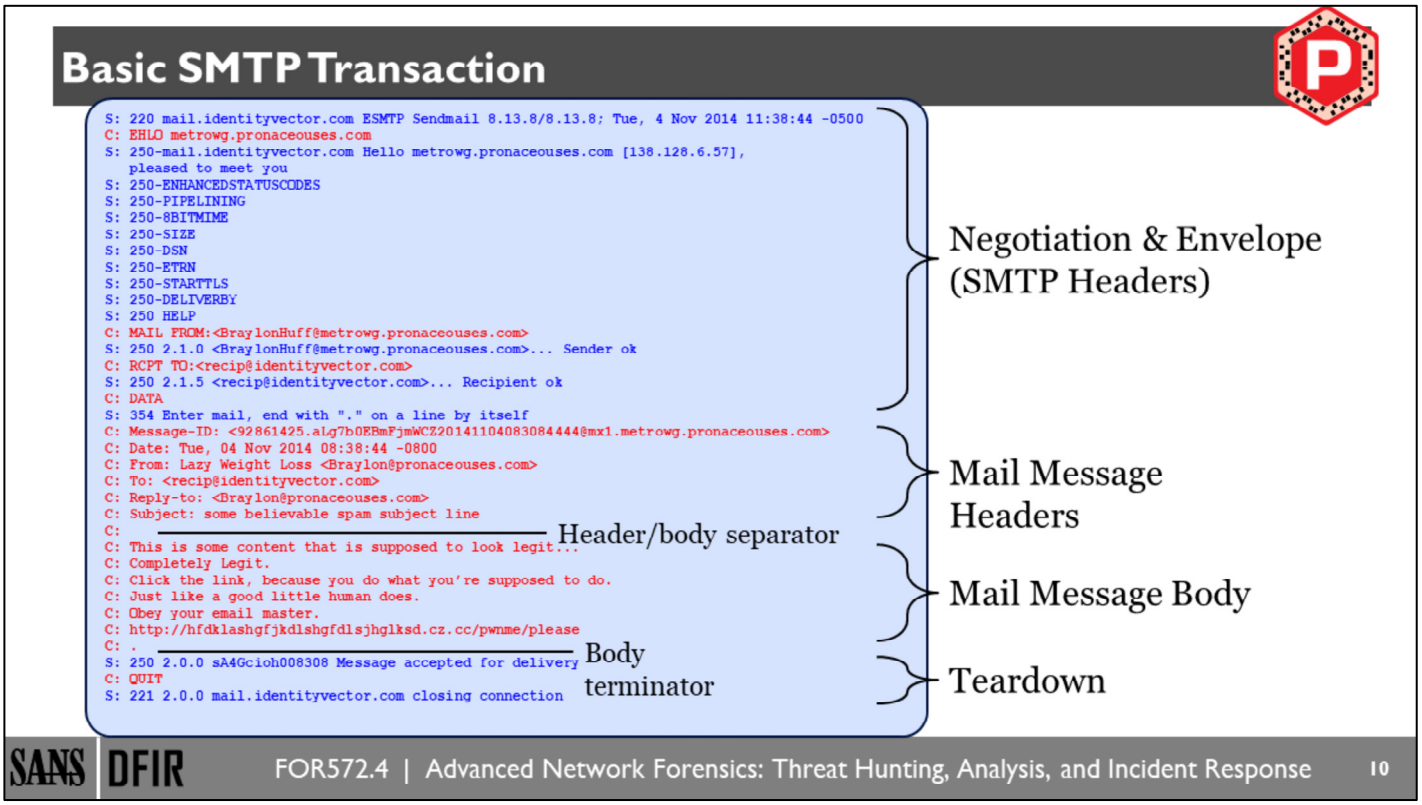
```
Return-Path: <dfir-bounces@lists.sans.org>
X-Original-To: dfir@lists.sans.org
X-Google-Smtp-Source:
   ACcGV62kXOpg6x548N/yP6miH4G73wEY7cmpT9SPTdBiF408MejjdrbI+5QMtyFDHtGymYTw+y
   k8iWmipjhnPGqMcnQ=
From: Sending User <sender@gmail.com>
To: Recipient User <recipient@gmail.com>
Message-ID:
   <CAAtPsn=SxnCpdfkwqVUofhfpT69A6FeJU83g-d9Gvi_yE3r2Sw@mail.gmail.com>

Received: by simcoe.identityvector.com (Postfix) with ESMTP id 811C2615A6
   for <phil@lewestech.com>; Thu, 27 Sep 2018 23:50:43 +0000 (UTC)
Received: from lists31a.clp.sans.org (localhost.localdomain [127.0.0.1])
   by lists.sans.org (Postfix) with ESMTP id 51829402A4
   for <phil@lewestech.com>; Thu, 27 Sep 2018 23:50:43 +0000 (UTC)
Received: from smtp21a.den.sans.org (smtp21a.den.sans.org [10.2.2.12])
   by lists.sans.org (Postfix) with ESMTP id 39C63401FF
   for <dfir@lists.sans.org>; Thu, 27 Sep 2018 23:49:38 +0000 (UTC)
Received: from mail-it1-f175.google.com (mail-it1-f175.google.com
   [209.85.166.175])
   by smtp21a.den.sans.org (Postfix) with ESMTP id DBCAC40209
   for <dfir@lists.sans.org>; Thu, 27 Sep 2018 23:49:37 +0000 (UTC)
Received: by mail-it1-f175.google.com with SMTP id h23-v6so629703ita.5
   for <dfir@lists.sans.org>; Thu, 27 Sep 2018 16:49:37 -0700 (PDT)
X-Received: by 2002:a24:3fc6:: with SMTP id
   d189-v6mr667165ita.64.1538092177178;



From: Spammer <dumb.spammer.fakeemail@gmail.com>
To: Recipient User <recipient@lewestech.com>
Message-ID:
   <CAG_jHOgGc403TRSzLSfQtxRyEfNzJszuqi3803-41373yYu62w@mail.gmail.com>
X-Google-Smtp-Source:
   AFSGD/Xu/ey9i1zaLX/1veZinzPE4qfmuehL4JttTITQ8mIgoaSTQcVSIjJU9lVJ6fuxNy1VRp
   yEVDK3pX1n4yYS+vA=
X-Google-Sender-Auth: 8WCA33I51hGwBZ0im2PyGSUpXE0

Received: by simcoe.identityvector.com (Postfix) with ESMTPS id F31BE61D9F
   for <recipient@lewestech.com>; Thu, 13 Dec 2018 04:36:19 +0000 (UTC)
Received: by mail-vs1-f67.google.com with SMTP id b74so412194vsd.9
   for <recipient@lewestech.com>; Wed, 12 Dec 2018 20:36:19 -0800 (PST)
X-Received: by 2002:a67:98c3:: with SMTP id
   g64mr10601079vsh.225.1544675779190; Wed, 12 Dec 2018 20:36:19 -0800 (PST)
Received: from 52669349336 named unknown by gmailapi.google.com with
   HTTPREST;
 Wed, 12 Dec 2018 23:36:18 -0500
Sender: Archana <dumb.spammer.fakeemail@gmail.com>
```





Primarily, SMTP is associated with TCP port 25 but has more recently expanded to include port 587. Typically, SMTP usage over port 587 requires authentication, which we'll discuss shortly. For a while, TCP/465 was also used for SMTP wrapped in an SSL/TLS connection. However, this latter use has been depreciated in favor of the "STARTTLS" option on ports 25 or 587. STARTTLS also provides encryption, as we'll see.

It's important to recognize that many legacy protocols—including SMTP—were created when the Internet was a completely different entity than it is now. Today's globally ubiquitous and often hostile environment demands a different mindset for protocol design. For example, the original SMTP specification had no considerations for authentication/authorization, non-English text, SPAM abatement, or even attachments. These features, which we consider integral to the modern email landscape, were all established after SMTP was already in widespread use. We'll look at a few of the protocol extensions that brought such features into being and how they affect our analysis of the protocol.



This is perhaps the most rudimentary example of an SMTP submission, as seen from a network vantage point. In this case, the “metrowg.pronaceeouses.com” server is relaying a message to the “mail.identityvector.com” server. For the purposes of this illustration, we'll state that the “mail.identityvector.com” server has been designated as an MX for the “identityvector.com” domain, so this transaction represents a transaction between an MTA and an MX.

First, the exchange contains a series of greetings between the client and server that establish the mutually supported protocols and protocol extensions. This starts with an “EHLO”, or “extended hello” command, but some exchanges may include an older “HELO” instead.

After the SMTP negotiation, the sender sends SMTP header data. Long headers are wrapped and indented on subsequent lines. The server knows to treat indented lines as continuations of their immediate predecessor.

The separator between headers and body content is a “double-CRLF”, consisting of a carriage return and newline followed by another carriage return and newline. In hex, these bytes are `0x0D0A0D0A`. This sequence is a common header/body separator in other ASCII protocols such as HTTP as well.

The message body here contains HTML content, and the client indicates it is done with the message by sending a single period character on a line by itself.

After the body—and therefore the entire message—is complete, the server provides the queued message identifier (which is generally available in the mail server’s logs) and the client initiates a clean teardown procedure.

```
S: 220 mail.identityvector.com ESMTP Sendmail 8.13.8/8.13.8; Tue, 4 Nov 2014 11:38:44 -0500
C: EHLO metrowg.pronaceouses.com
S: 250-mail.identityvector.com Hello metrowg.pronaceouses.com [138.128.6.57],
   pleased to meet you
S: 250-ENHANCEDSTATUSCODES
S: 250-PIPELINING
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250-ETRN
S: 250-STARTTLS
S: 250-DELIVERBY
S: 250 HELP
C: MAIL FROM:<BraylonHuff@metrowg.pronaceouses.com>
S: 250 2.1.0 <BraylonHuff@metrowg.pronaceouses.com>... Sender ok
C: RCPT TO:<recip@identityvector.com>
S: 250 2.1.5 <recip@identityvector.com>... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Message-ID: <92861425.aLg7b0EBmFjmWCZ20141104083084444@mx1.metrowg.pronaceouses.com>
C: Date: Tue, 04 Nov 2014 08:38:44 -0800
C: From: Lazy Weight Loss <Braylon@pronaceouses.com>
C: To: <recip@identityvector.com>
C: Reply-to: <Braylon@pronaceouses.com>
C: Subject: some believable spam subject line
C:
C: This is some content that is supposed to look legit...
C: Completely Legit.
C: Click the link, because you do what you're supposed to do.
C: Just like a good little human does.
C: Obey your email master.
C: http://hfdklashgfjkdlshgfdlsjhglksd.cz.cc/pwnme/please
C: .
S: 250 2.0.0 sA4Gcioh008308 Message accepted for delivery
C: QUIT
S: 221 2.0.0 mail.identityvector.com closing connection
```

Header/body separator

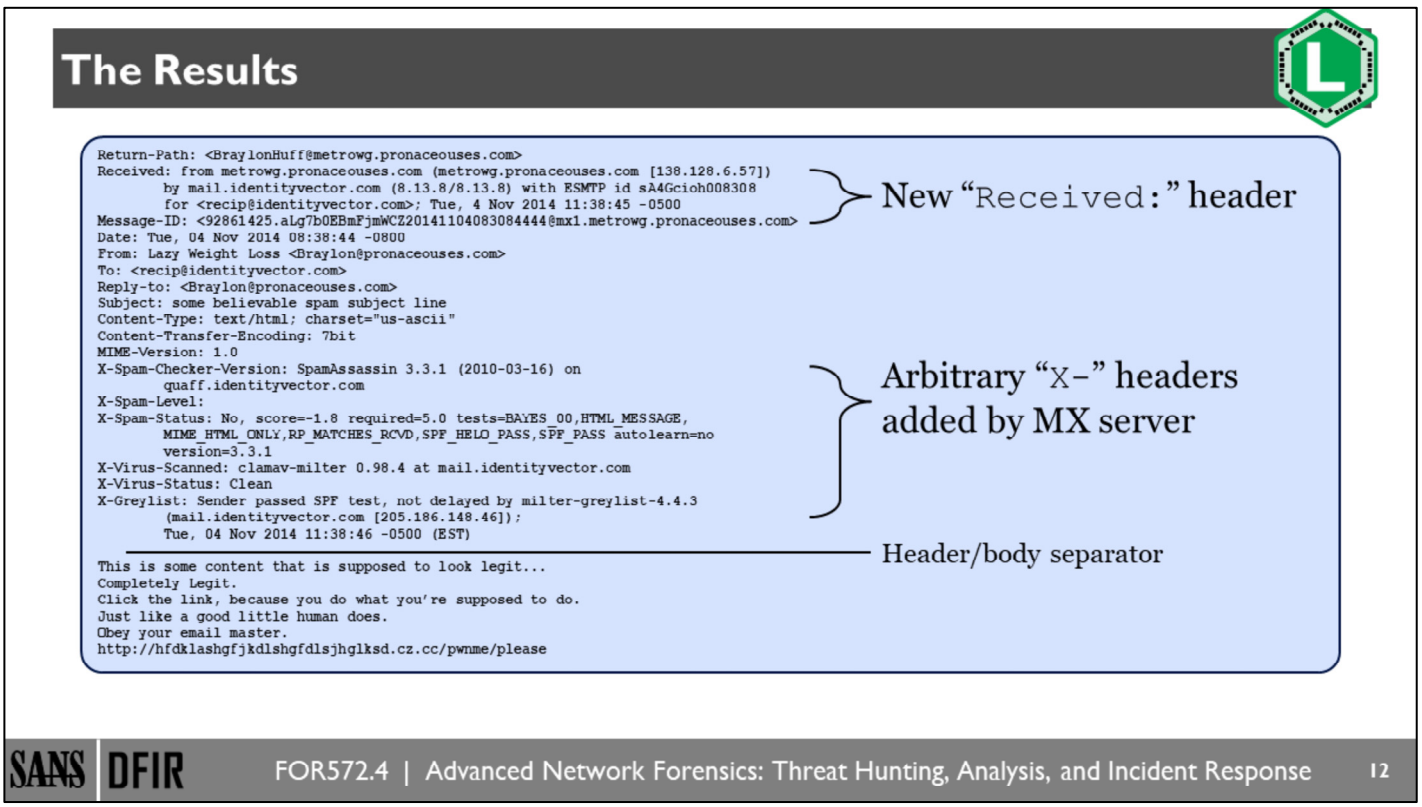
Body terminator

Negotiation & Envelope (SMTP Headers)

Mail Message Headers

Mail Message Body

Teardown



The resulting message, as delivered to the recipient's Inbox (and possibly stored on disk, depending on the MDA software), is shown here. You can see that the headers included by the sending SMTP server remain intact during the MTA-to-MX transaction, but that the MX has added some headers of its own. Most important to note is that an additional "`Received:`" header is added at each hop in the transaction. It's also useful to recognize that any SMTP server in the path can also add, remove, or alter whatever SMTP headers or body data it's configured to mangle. Although this is uncommon, it does bear consideration when thinking from a forensic point of view.

Despite the few header changes from the MX's own delivery process, it's easy to see that the majority of the network transaction from the previous slide has remained intact.

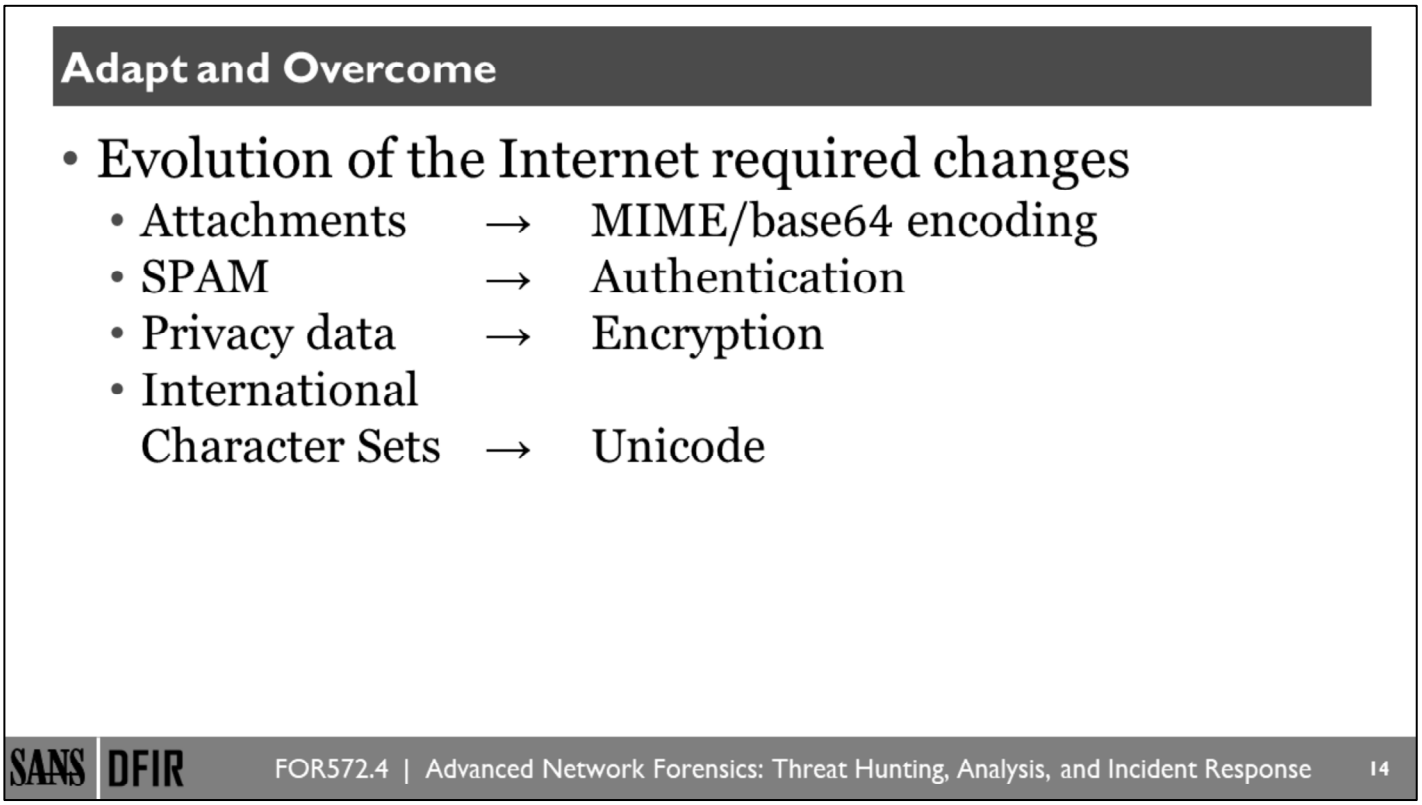
Any MTA or MX along the way can optionally add whatever "`X-`" headers, indicating free-form extension data. Often this includes antivirus, antispam, and other features, but can truly be anything the server is configured to add. Most webmail providers now include a header containing the encoded or encrypted IP address of the message originator. These can aid in tracking down abuse and other malicious activity.

```
Return-Path: <BraylonHuff@metrowg.pronaceouses.com>
Received: from metrowg.pronaceouses.com (metrowg.pronaceouses.com [138.128.6.57])
        by mail.identityvector.com (8.13.8/8.13.8) with ESMTP id sA4Gcioh008308
        for <recip@identityvector.com>; Tue, 4 Nov 2014 11:38:45 -0500
Message-ID: <92861425.aLg7b0EBmFjmWCZ20141104083084444@mx1.metrowg.pronaceouses.com>
Date: Tue, 04 Nov 2014 08:38:44 -0800
From: Lazy Weight Loss <Braylon@pronaceouses.com>
To: <recip@identityvector.com>
Reply-to: <Braylon@pronaceouses.com>
Subject: Too Self-Conscious about your body?
Content-Type: text/html; charset="us-ascii"
Content-Transfer-Encoding: 7bit
MIME-Version: 1.0
X-Spam-Checker-Version: SpamAssassin 3.3.1 (2010-03-16) on
        quaff.identityvector.com
X-Spam-Level:
X-Spam-Status: No, score=-1.8 required=5.0 tests=BAYES_00,HTML_MESSAGE,
        MIME_HTML_ONLY,RP_MATCHES_RCVD,SPF_HELO_PASS,SPF_PASS autolearn=no
        version=3.3.1
X-Virus-Scanned: clamav-milter 0.98.4 at mail.identityvector.com
X-Virus-Status: Clean
X-Greylist: Sender passed SPF test, not delayed by milter-greylist-4.4.3
(mail.identityvector.com [205.186.148.46]);
Tue, 04 Nov 2014 11:38:46 -0500 (EST)

This is some content that is supposed to look legit...
Completely Legit.
Click the link, because you do what you're supposed to do.
Just like a good little human does.
Obey your email master.
http://hfdklashgfjkdlshgfdlsjhglksd.cz.cc/pwnme/please
```

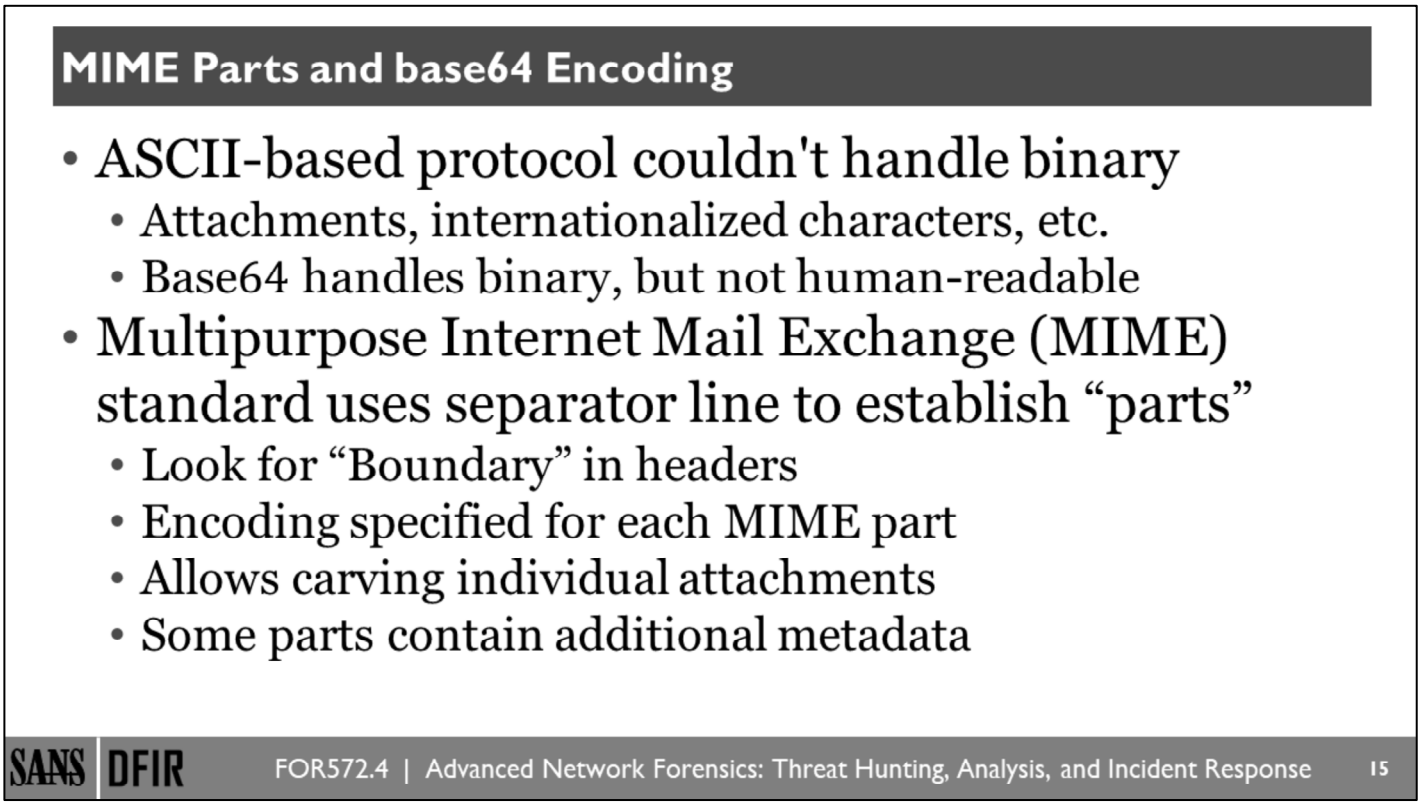
New "`Received:`" header

Arbitrary "`X-`" headers added by MX server

Header/body separator



As we mentioned previously, SMTP has evolved significantly since its early incarnations. We'll take a look at each of these more significant modifications in turn.



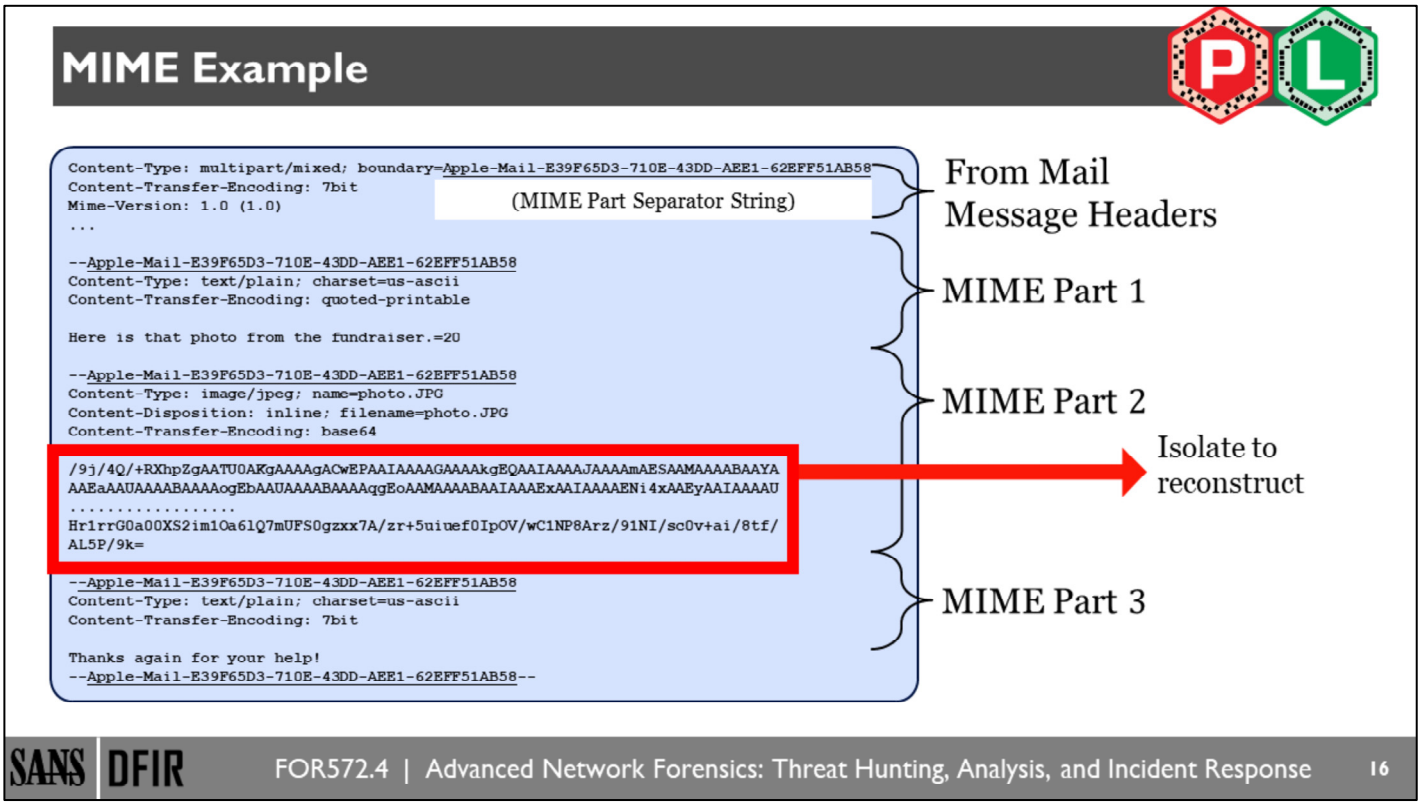
Although it was great for (English) text, SMTP was never designed to handle that 175MB PowerPoint file, endless JPEGs of kitty cats, or even non-Western languages. To bring these now-fundamental features into existence, base64 encoding was selected. This encoding method represents arbitrary binary bytes in an all-ASCII character set consisting of 64 possible characters. However, dropping massive blocks of base64 text into the body of an email was not an option that most humans would be able to deal with.

To address this, the Multipurpose Internet Mail Extension, or MIME, standard was adopted for use in SMTP. Email messages that include MIME data contain a "boundary" indicator in the headers. This string designates the start of each new MIME part in the overall mail message.

Each part can also include its own headers, which often contain useful metadata such as the attachment's filename, encoding method, etc.

By tracking these boundaries and referencing each part's header metadata, we can easily discern each subsequent message part, making for rather simple attachment carving.

References:
https://for572.com/9kof7
https://for572.com/ap-86
https://for572.com/0dry7
https://for572.com/71qhr



This slide depicts several excerpts from an SMTP exchange including three MIME parts.

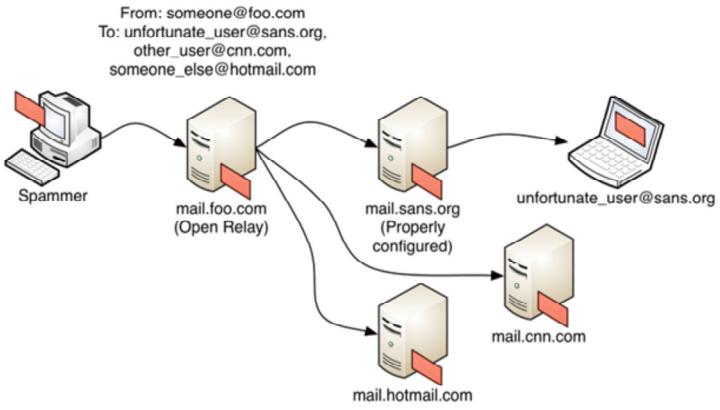
The headers indicate the MIME version, but more importantly for us, the boundary string. This unique string will be used within this individual message to designate the start of a new part.

By looking into the message body, we see that each part consists of its own headers and body. Each part is preceded by a `0x0D0A0D0A` separator followed by two hyphens and the boundary string. The headers may include encoding and other metadata, such as the attachment's filename. The headers continue until the separator, again a `0x0D0A0D0A`. After that separator, the body continues until the next boundary starts. In the case of base64-encoded parts, we can simply extract the encoded section and decode the original attachment or content part to its original form.

Each MIME part also has its own headers, which may be useful to characterize the nature of the message or its components. For example, the "`Content-Disposition: inline`" header used on the JPEG in this example requests that the recipient's mail client display a thumbnail of the image instead of an icon. This may be useful for an attacker who wishes to trigger a vulnerability in a rendering library rather than one in a client-side helper application.

Authentication

- Authentication prevents wanton SPAMming



- Multiple authentication methods
  - Plaintext (should require encrypted SMTP)
    - LOGIN, PLAIN, OAUTHBEARER – base64-encoded strings
  - Encrypted (generally considered "secure")
    - CRAM-MD5, DIGEST-MD5, GSSAPI, NTLM, etc.

Before spam became such a pervasive problem, there was no need to authenticate users who were requesting to send email. Server operators had a reasonable expectation that most or all SMTP transactions were legitimate and MTAs unquestioningly passed email messages on toward their final recipients.

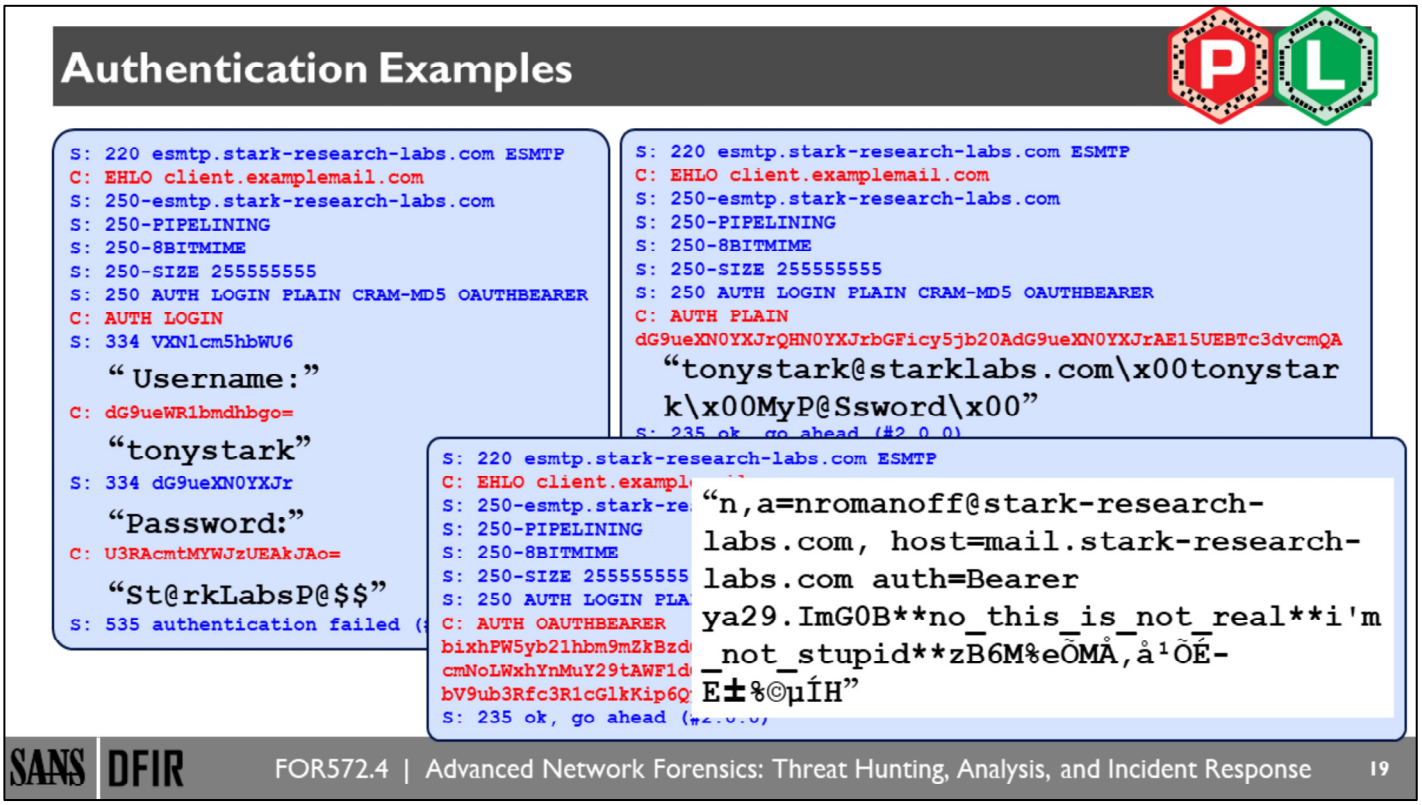
In the example illustrated on this slide, a spammer is using a poorly-configured SMTP server at "`mail.foo.com`" to relay a forged message from "`someone@foo.com`". Although this depicts just one message, even a small spamming operation can send many hundreds of thousands of messages per day using this methodology.

Obviously, that model was soon exploited by spammers, so RFC 2554 was created in 1999, establishing the SMTP-AUTH standard. (This RFC has since been replaced by RFC 4954.) SMTP-AUTH establishes multiple authentication methods, some of which are considered more secure than others.

The LOGIN and PLAIN methods are widely used. As you'll see in a moment, these are simple, consisting merely of base64-encoded username and password credentials. Remember: Encoded is not encrypted! These methods should never be used over unencrypted connections, and many SMTP servers will not advertise their availability unless TLS is in use.

A method commonly seen in modern configuration is the OAUTH family of authentication methods, used for numerous mail services including Gmail. This method involves a granted token that can then be revoked if and when needed. Although this token does not include the account's password, it is also just a base64-encoded string representing a self-contained authorization token and should be considered plaintext.

Because of this major security shortfall, the RFCs also created a number of authentication methods that provide a greater level of security. These are generally considered acceptable even over non-TLS connections because they provide their own cryptographic protections. Although we won't examine all of these methods in detail, it's important to note that there are multiple authentication methods, and the two encoded methods can provide valuable credential evidence for the investigator.



Shown here are examples of several "insecure" authentication methods we discussed on the previous slide—LOGIN, PLAIN, and OAUTHBEARER. As shown in the server's capability listing, the `esmtp.stark-research-labs.com` server can perform authentication using all three of these methods plus CRAM-MD5.

After the client requests the LOGIN method, the server responds with the base64-encoded string "`Username:`". The client responds in kind with the encoded username. The parties then perform a similar exchange for the password. In this case, the authentication attempt was unsuccessful.

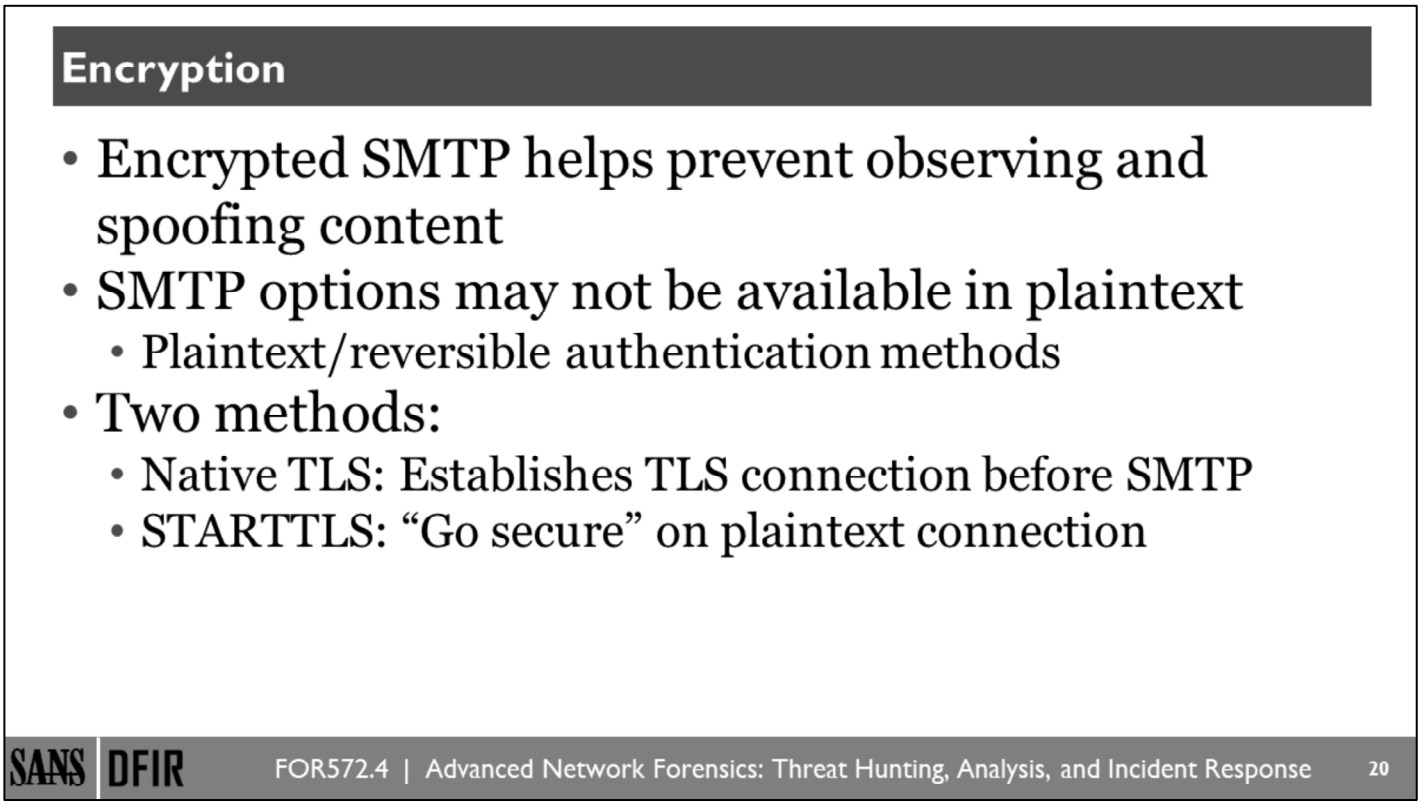
In the second example, the client requests the PLAIN method. For this method, the client can immediately send an encoded string with either two or three components, each terminated with a null byte. These components are:

- Authorization identity (optional): The entity as which the client is attempting to authenticate.
- Authentication identity: The username
- Password

In the second case, the authentication request was successful, so the client would then initiate with the email message submission process.

In the final example, a client requests the OAUTHBEARER method, which consists of a unique string allocated by the mail provider to a given user account. For our purposes, this should be considered like an alternate password for the account. Any party with this bearer token can access the account to which it is associated. The exact makeup of these tokens is solely at the discretion of their issuers.

References:
https://for572.com/37zjq
https://for572.com/kw7z3
https://for572.com/tfxkm

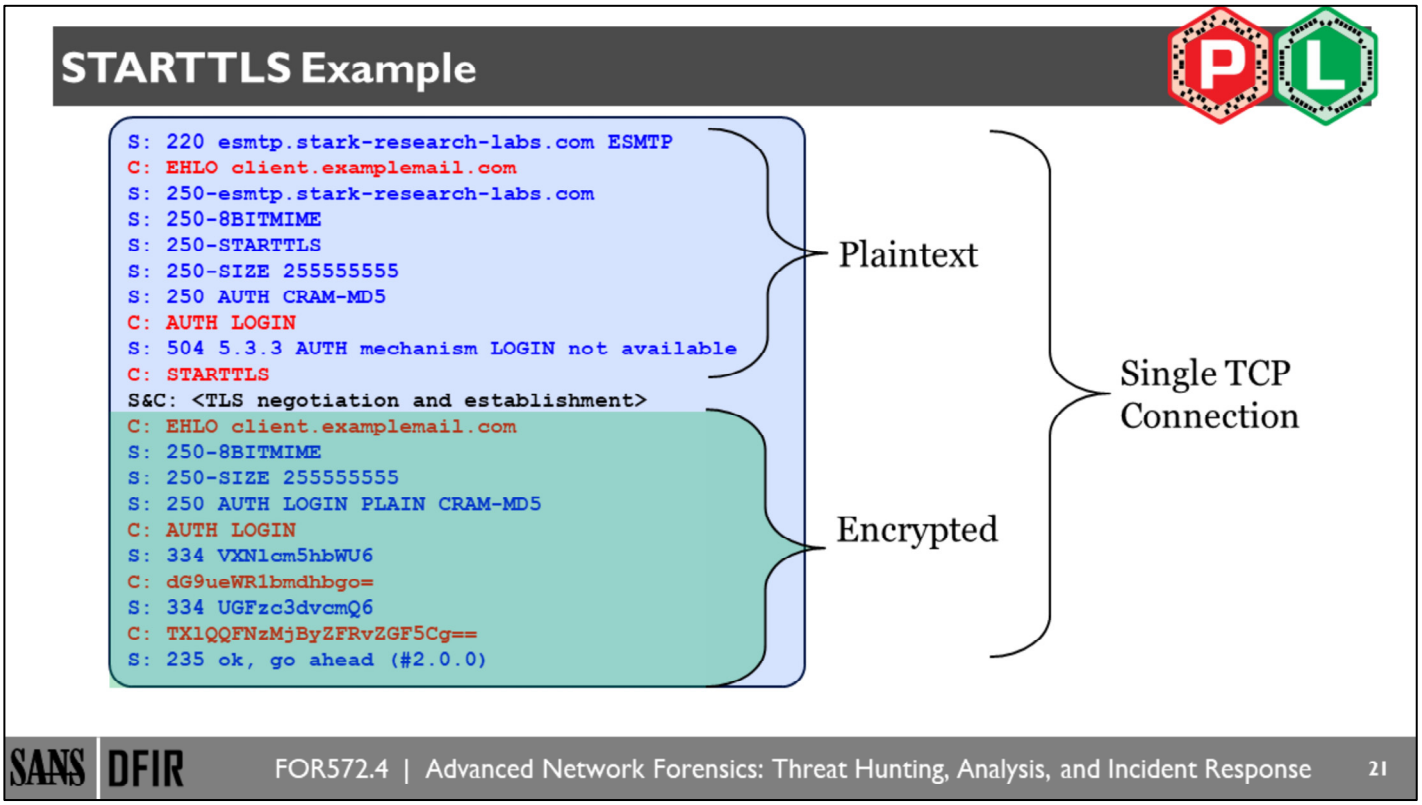


As you know, encryption can provide both confidentiality and integrity, depending on how it is implemented. We'll discuss encryption at great length later, but let's briefly touch on its application within the context of SMTP.

Considering the inadequacy of the LOGIN and PLAIN authentication methods to protect the user's login credentials, encryption becomes very important in most SMTP transactions. Most SMTP servers will not even advertise these two methods unless the connection has been secured.

There are two primary mechanisms of securing the SMTP connection: TLS and STARTTLS. Although not as common today as it once was, the TLS mechanism starts the new TCP connection with an immediate exchange to encrypt the connection. After it is secured, the SMTP transaction starts.

On the other hand, with the STARTTLS mechanism, the connection starts as a standard, unencrypted one. The client requests that the server "goes secure", at which point an TLS negotiation occurs over the same connection. The benefit in using STARTTLS is that a separate port doesn't need to be allocated and managed across an architecture. In this model, an SMTP server may present different capabilities on the same connection, depending on whether a STARTTLS command has been issued and the negotiation completed.

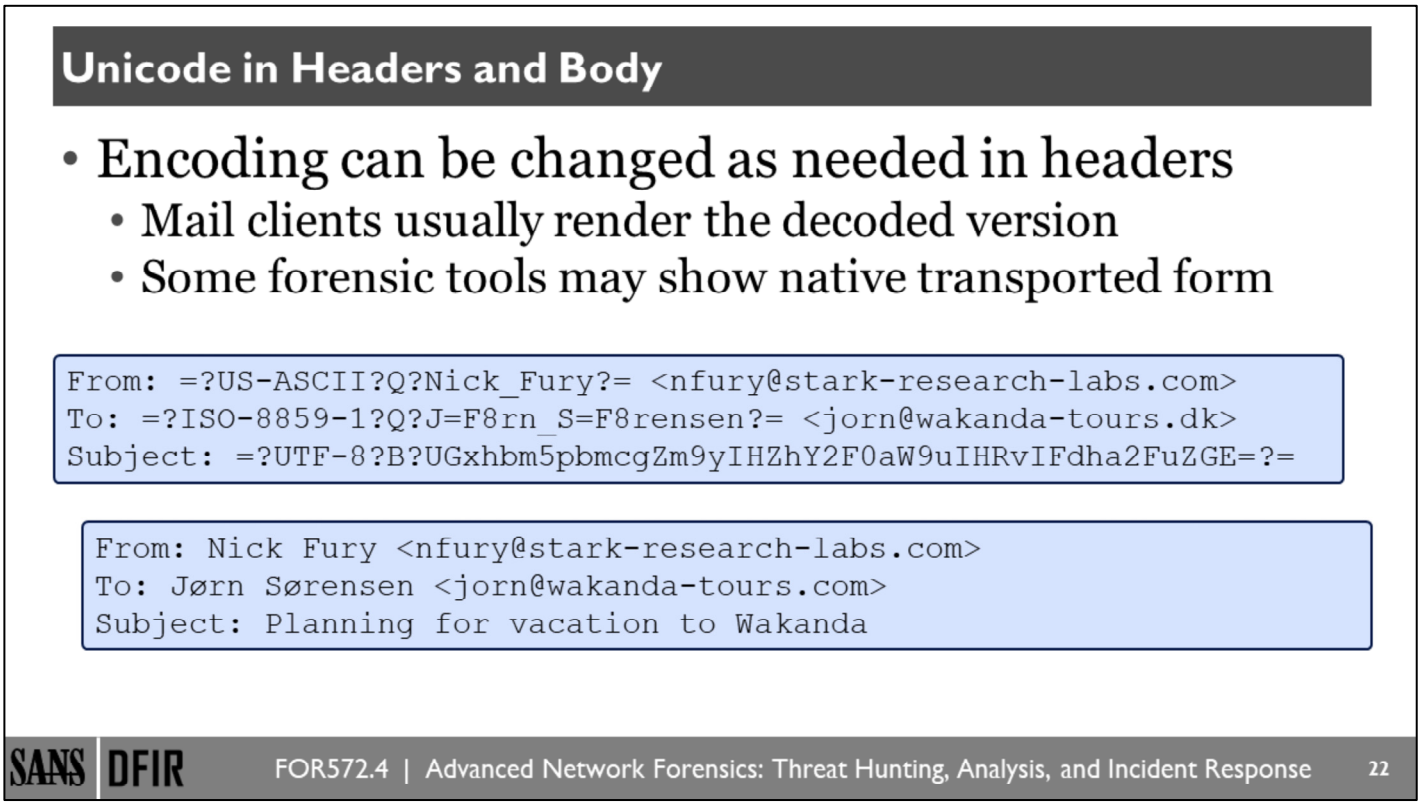


Here, you can see an example of the STARTTLS process. The client system connects to the esmtp.stark-research-labs.com server, which advertises that it can use the STARTTLS mechanism as well as authenticate using the CRAM-MD5 method.

The client initiates the negotiation, and everything shown in the shaded region represents the conversation that occurs over the encrypted channel. The key point here is that the same connection is used throughout.

During the encrypted phase of the conversation, the server advertises that it will accommodate the LOGIN and PLAIN methods in addition to the CRAM-MD5 method. The client then selects the LOGIN method and proceeds with the SMTP transaction.

References:
https://for572.com/1unt8

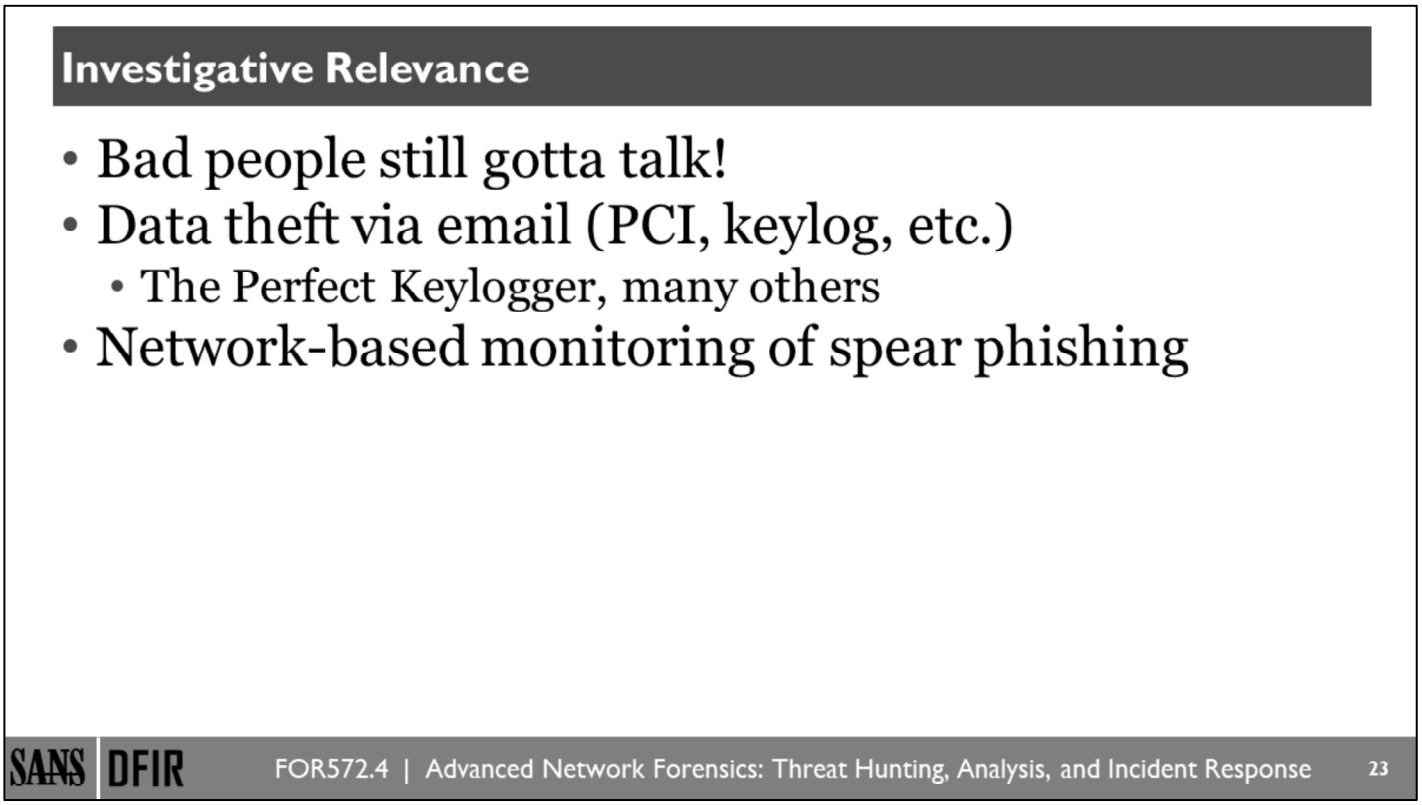


Since SMTP still exclusively uses US-ASCII for its content transport, special handling must be used for any characters outside of that set. While modern mail client software will render the text as it is meant to be read, looking at the version in flight can result in a challenging exercise in decoding.

As seen in the example above, even the US-ASCII "`From:`" header is caveated as such even though no replacements were needed. The "`To:`" header, however, shows that it is encoded with ISO 8859-1, often referred to as the "`latin-1`" character set because its 256 characters generally provide coverage for a wide variety of Latin-based languages. The example "`Subject:`" header, however, shows that any Unicode-defined character set can be used, with both base64 and UTF-8 encoding shown.

This encoding standard is defined in RFC 2047[1], and an online decoder[2] may be helpful for testing purposes. However, for any sensitive case data, an offline equivalent should be used.

References:
[1] https://for572.com/vbwe9
[2] https://for572.com/502uv

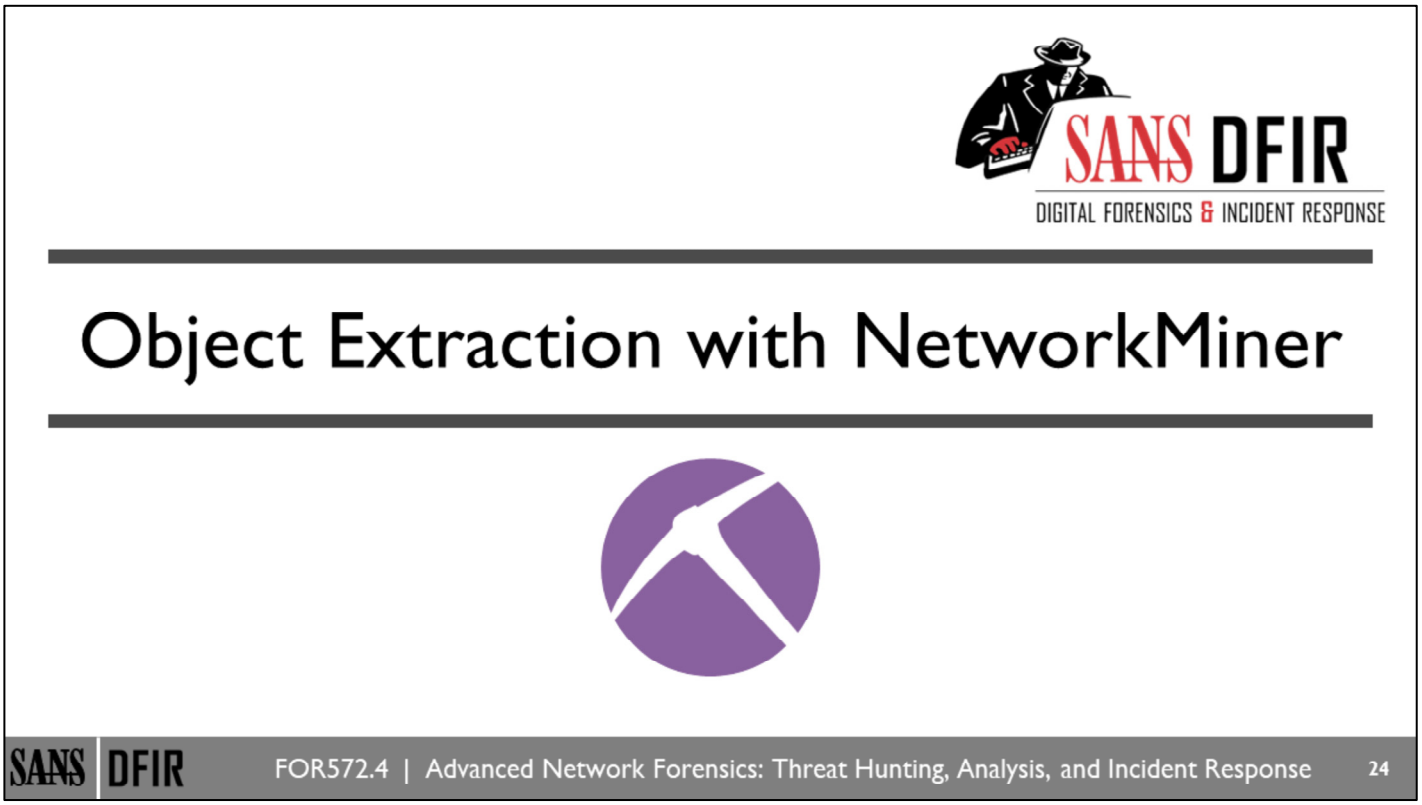


Of course, we're most interested in knowing how SMTP is relevant during an incident response or investigation. After all, SMTP is a rather old protocol, and probably not one that most malicious actors use… or do they?

One of the key things to remember is that bad guys still have to communicate somehow! In many cases, they'll use what's available—such as email. Although it's unlikely that a nation-state actor would use email within a victim's network to coordinate operations, a rogue employee might certainly use email as a communication mechanism, or possibly a means of sending privileged information outside of the network.

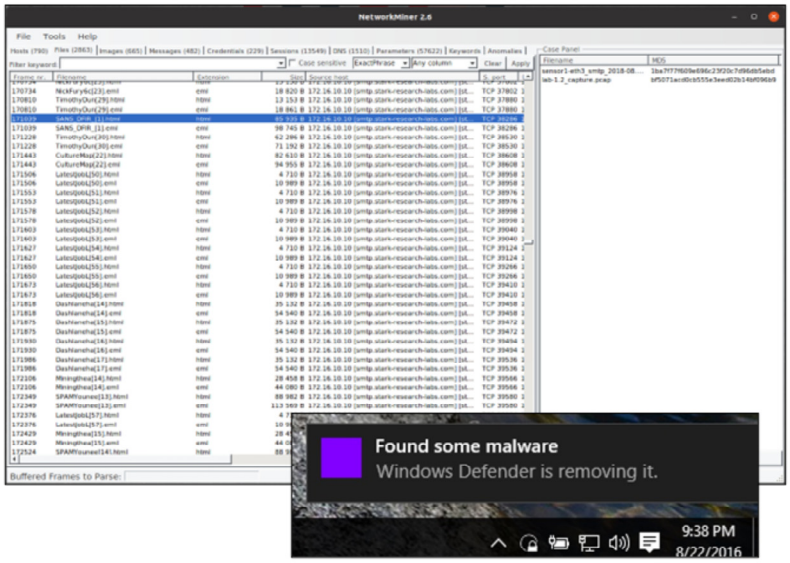
In other pockets of the pure criminal world, software keyloggers are still used quite efficiently. In fact, many credit card readers are no more than keyboard devices from the computer's perspective, so a keylogger on a point-of-sale or cash register system is a common tactic for PCI data theft. Keylogger software can be configured to send captured data to an external email account, providing attackers with an easy data theft collection mechanism.

Another key presence of SMTP in the modern threat environment is the pervasive use of spear phishing. There is no discounting the fact that this attack method is both widely and efficiently used by strategic attackers to gain footholds within their targets' networks. Of course, the penultimate leg of every such message is across SMTP—right before the MDA puts it into the Inbox of their target's client software. The ability to examine SMTP exchanges can provide useful insight into the flow of spear phishing messages and their payloads. Understanding the nature of SMTP headers can show the hops a message took in getting to its destination, and knowing how they can be forged or altered can help to keep “red herring” findings in check. Using the MIME standard to effectively and efficiently carve attachments can provide a reverse-engineering team with the initial dropper binary that an attacker used to gain access to the target's environment. Of course, analysis of this dropper may very well lead to additional network indicators that can be used to scope an incident and start to plan toward remediation.



This page intentionally left blank.

NetworkMiner



- Commercial and free editions
- Multi-protocol field/object extraction
  - Files and images
  - Mail messages
  - Credentials
- Extracts all to disk

NetworkMiner is an exceptionally useful dual-function tool that can passively sniff traffic from the local interface or read previously captured data from pcap files.

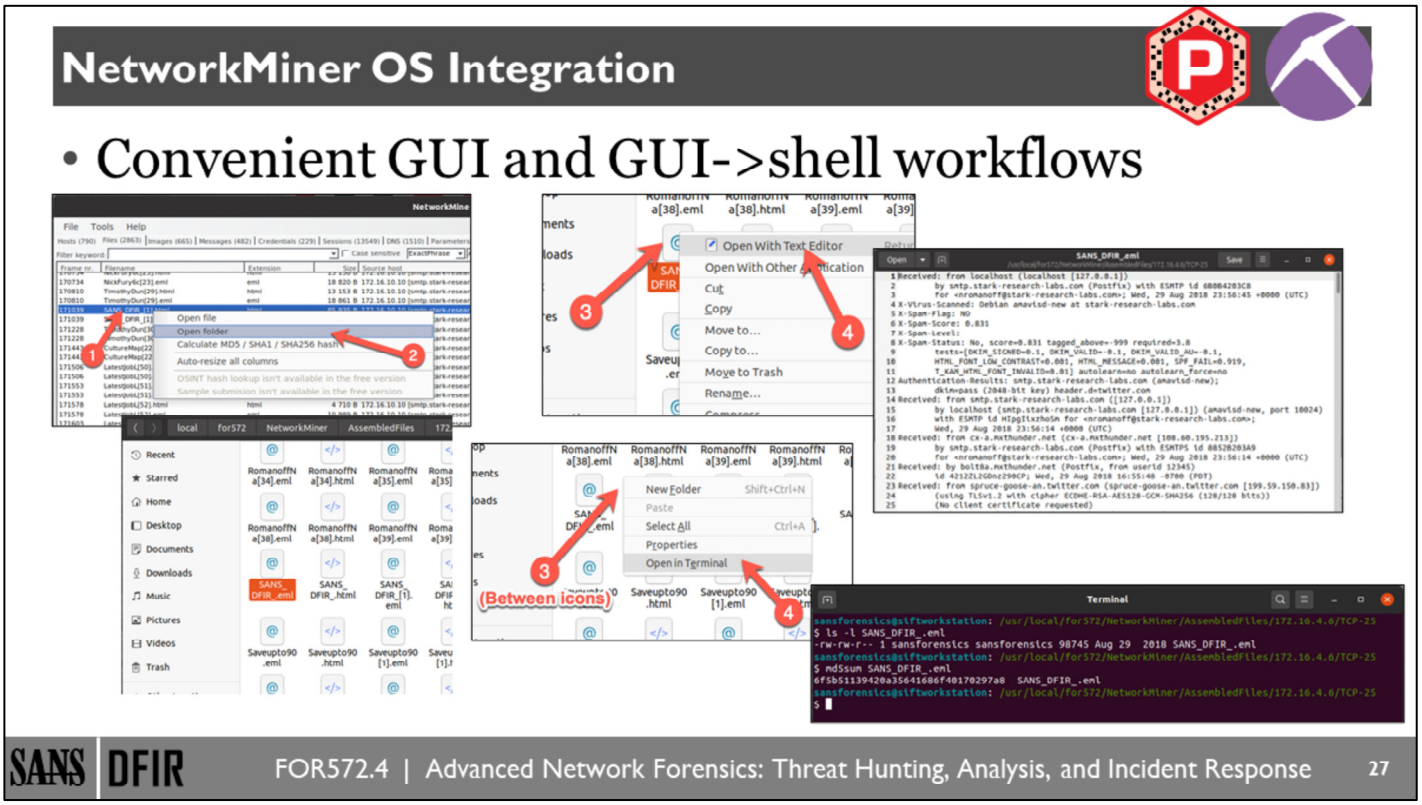
The tool reassembles session contents, extracts files and images, and decodes various client-to-server fields including cookies, URL parameters, and credentials for plaintext protocols. These objects are written to disk in real time, so it's important to run NetworkMiner on a system that doesn't have antivirus or anti-malware software, as any malicious files that are extracted would be immediately removed or quarantined.

The straightforward interface allows an analyst to explore the various fields and objects that NetworkMiner extracts while it is still processing the source network traffic. Those artifacts are displayed in tabs across the top of the window, detailing such categories as observed hosts, files, images, mail messages, and more.

As NetworkMiner identifies objects such as files, certificates, and images, they are written out to the filesystem in real time. This allows the analyst to explore or process those files through normal filesystem access methods, opening up countless possibilities to use other tools as needed.

The free version of the tool is licensed for real-world cases and is not time-limited. The commercial version adds several useful features such as a command-line interface, CSV and JSON export capability, port-independent protocol identification, and more. Also note that while NetworkMiner is a Windows-native tool, it can run in Linux and macOS using the cross-platform .NET framework called Mono. While this is a great convenience for isolating a system running in a virtual machine, running this in such a manner significantly slows down processing.

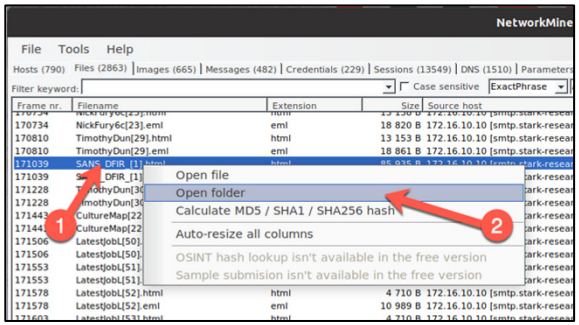
References:
https://for572.com/1hti6

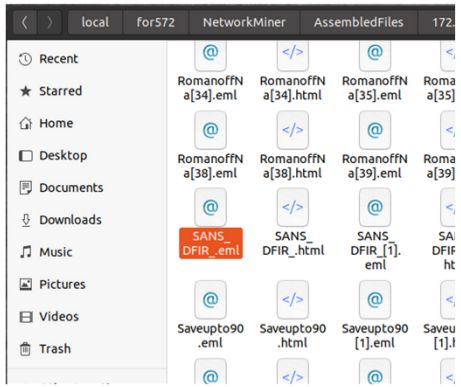


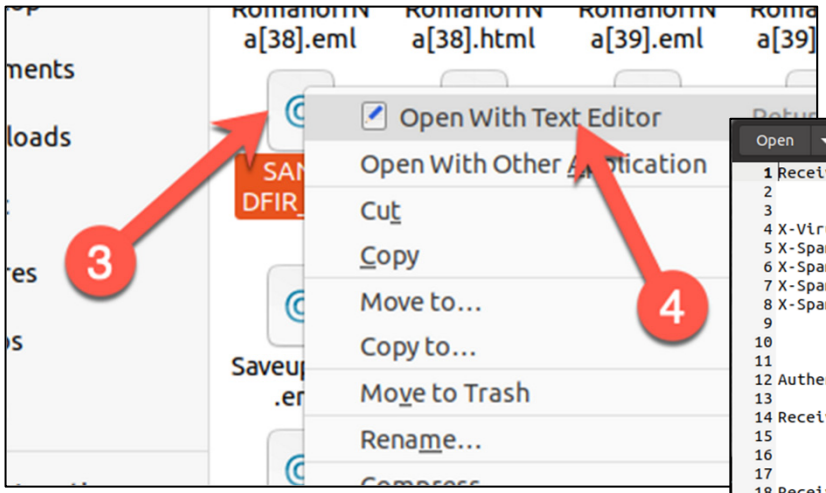
Running NetworkMiner in your FOR572 SIFT VM also provides a workflow that enables inspecting the content it extracts with both GUI and shell-based tools. Right-clicking a file or image in the NetworkMiner interface results in a dialog box from which you can open the parent folder of the extracted object. (Note that opening the file itself, while possible, may present a risk if the content is malicious.)

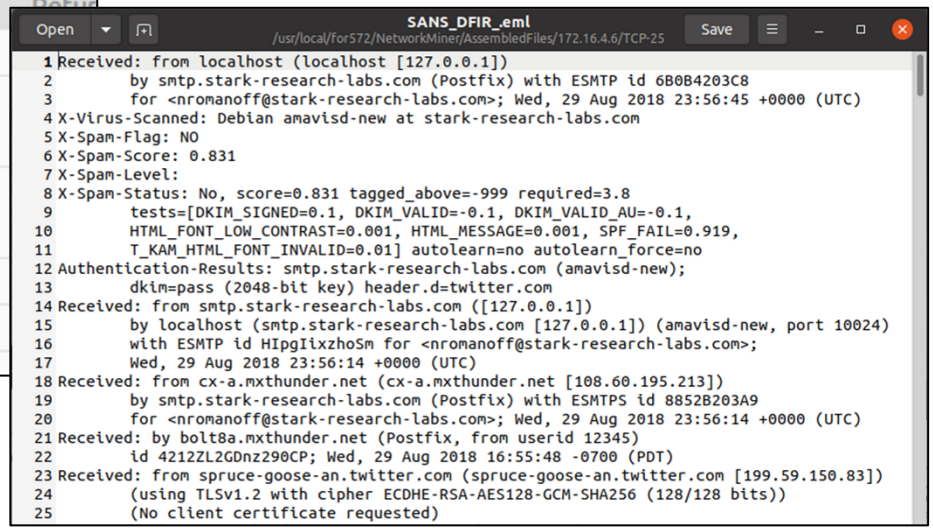
Opening the folder navigates the file browser (Nautilus, in the case of your SIFT VM) to the directory where the file was extracted. Since these directories are nested based on the IP address, protocol, and port, a few clicks can save quite a bit of navigation. From there, you can choose to open the file using an appropriate application, such as a plaintext editor as shown here.

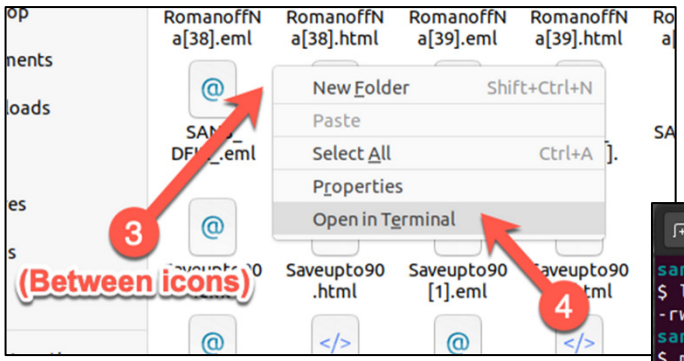
Another useful workflow involves a Nautilus extension installed in your SIFT VM that allows a quick pivot to a specific directory in a shell. To use this feature, right-click the background of a directory in Nautilus (anywhere **except for** an icon) and select “Open in Terminal”.

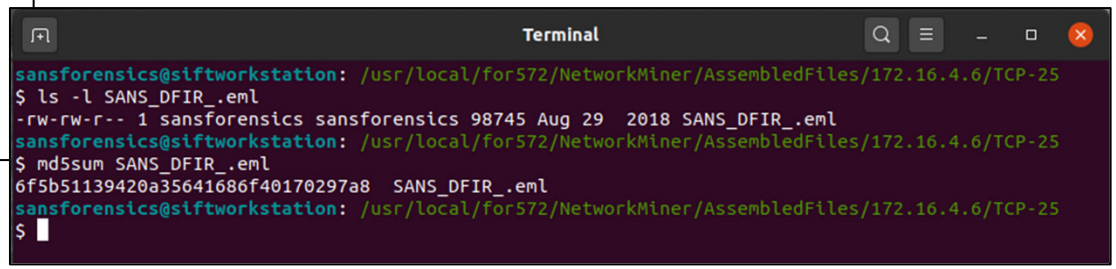






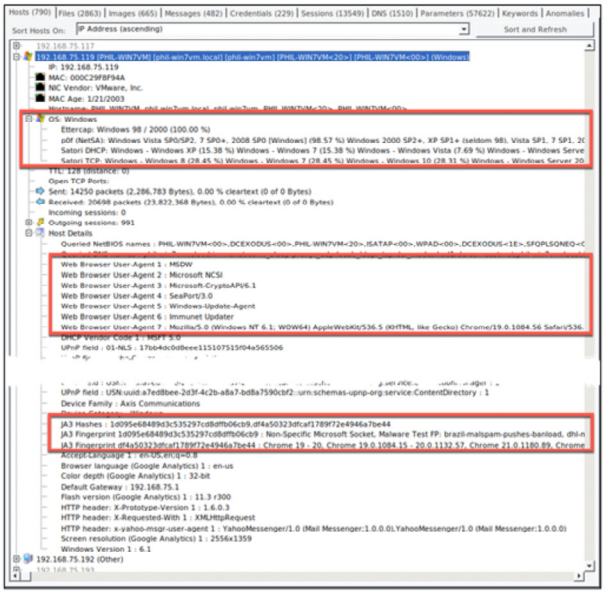






NetworkMiner Host Profiling

- Observed hosts profiled with numerous artifacts
  - OS fingerprinting libraries
  - User-Agent strings
  - JA3 hashes
- Centralized host identification using multiple data sources



SANS DFIR FOR572.4 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response 29

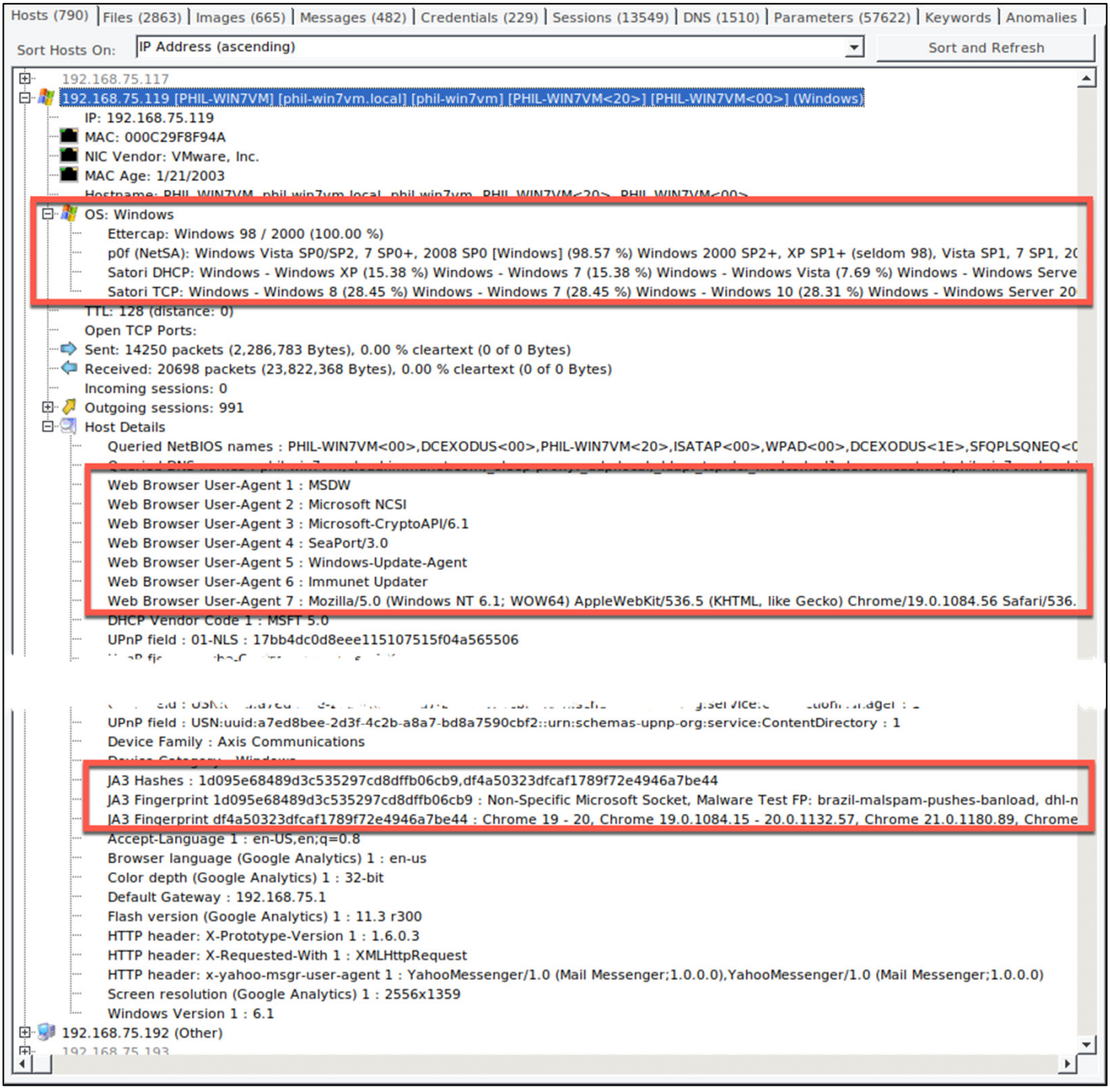
Any host that NetworkMiner identifies is profiled with a variety of libraries and methodologies[1]. The results are displayed on the "Hosts" tab by unrolling an IP address record in the interface. The results include operating system profiling using libraries such as Ettercap[2], p0f[3], and Satori[4]. These are not perfect in their identification, however. The likelihood of each profiling decision is displayed as a percentage and the user should not assume a high confidence implies absolute accuracy.

Additional profiling methods presented in the NetworkMiner interface include observed HTTP User-Agent strings and calculated JA3 hashes and fingerprints. JA3 fingerprinting[5] is a method that consolidates observed TLS negotiation requests as a means of identifying client-side software making the requests.

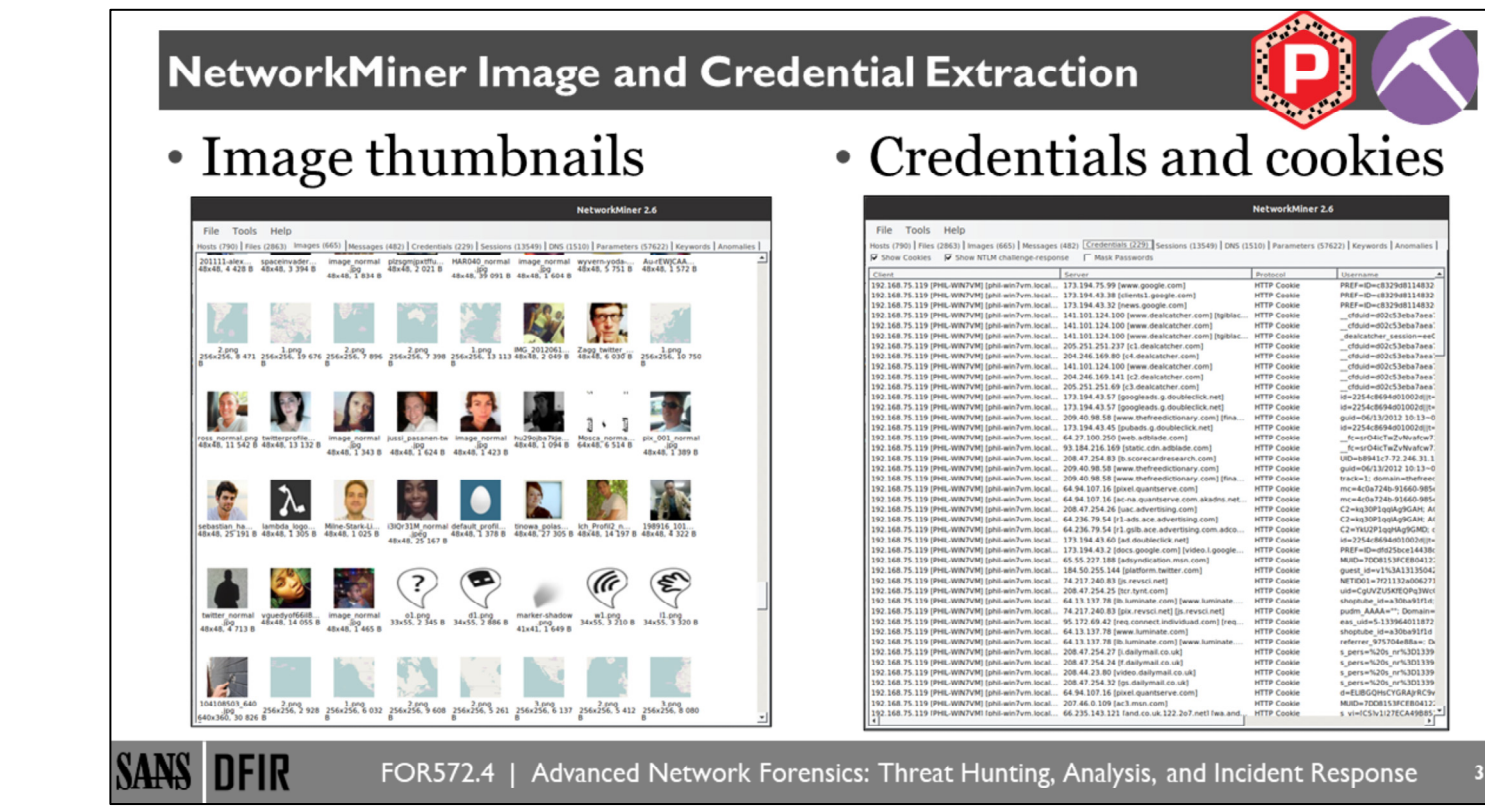
Together, these various profiling methods can be used to establish a general guideline of what operating system and application software may be in use on each host.

References:

[1] https://for572.com/zr3xs
[2] https://for572.com/sxqhk
[3] https://for572.com/su7ib
[4] https://for572.com/b-3f6
[5] https://for572.com/ja3
https://for572.com/8v5-y



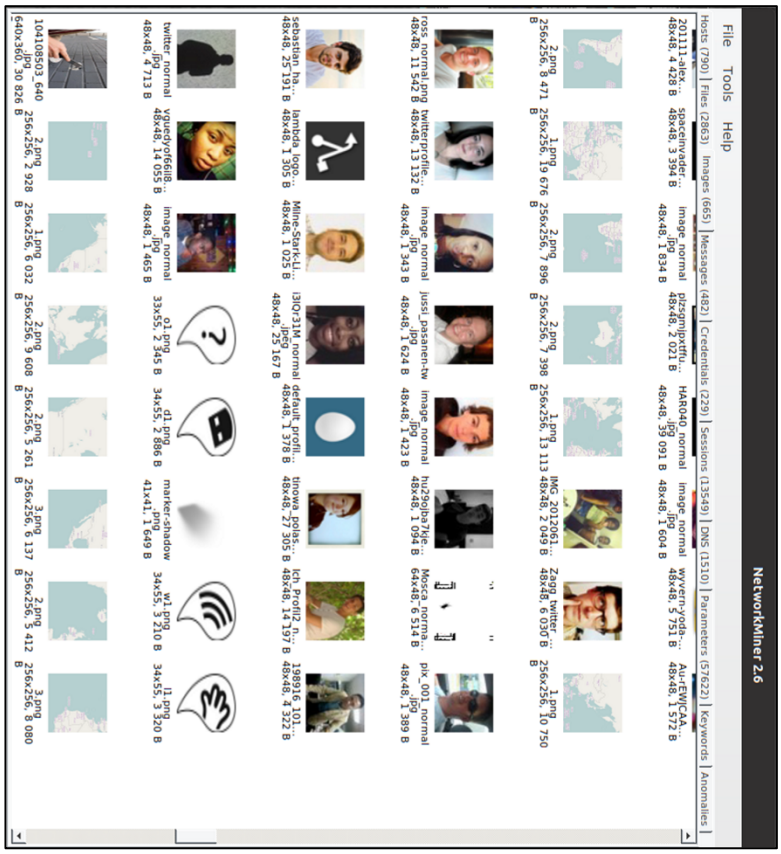
NetworkMiner Image and Credential Extraction

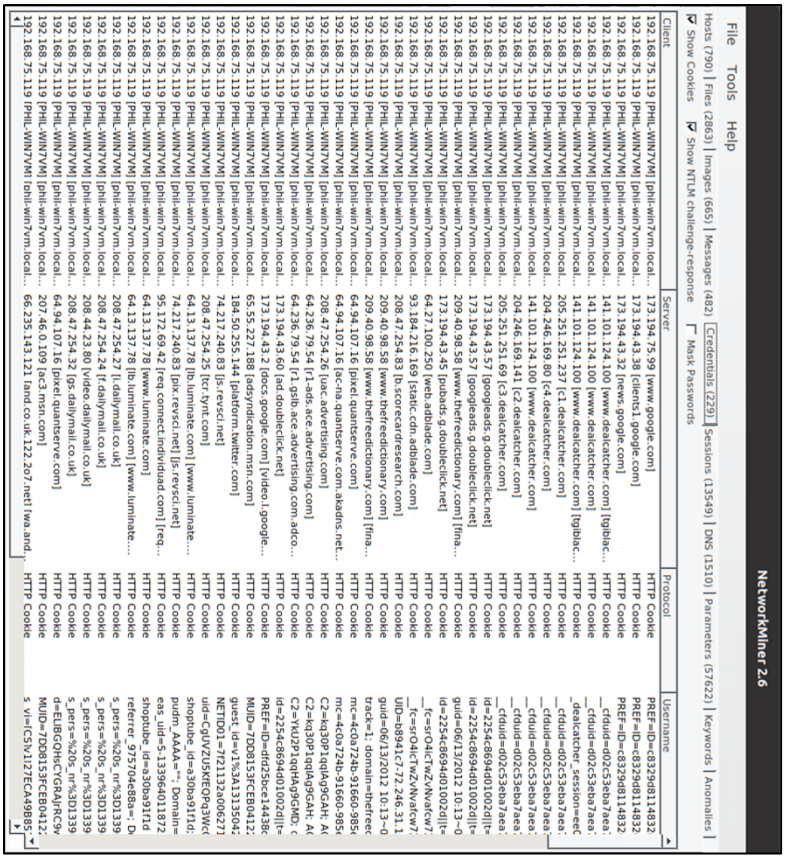


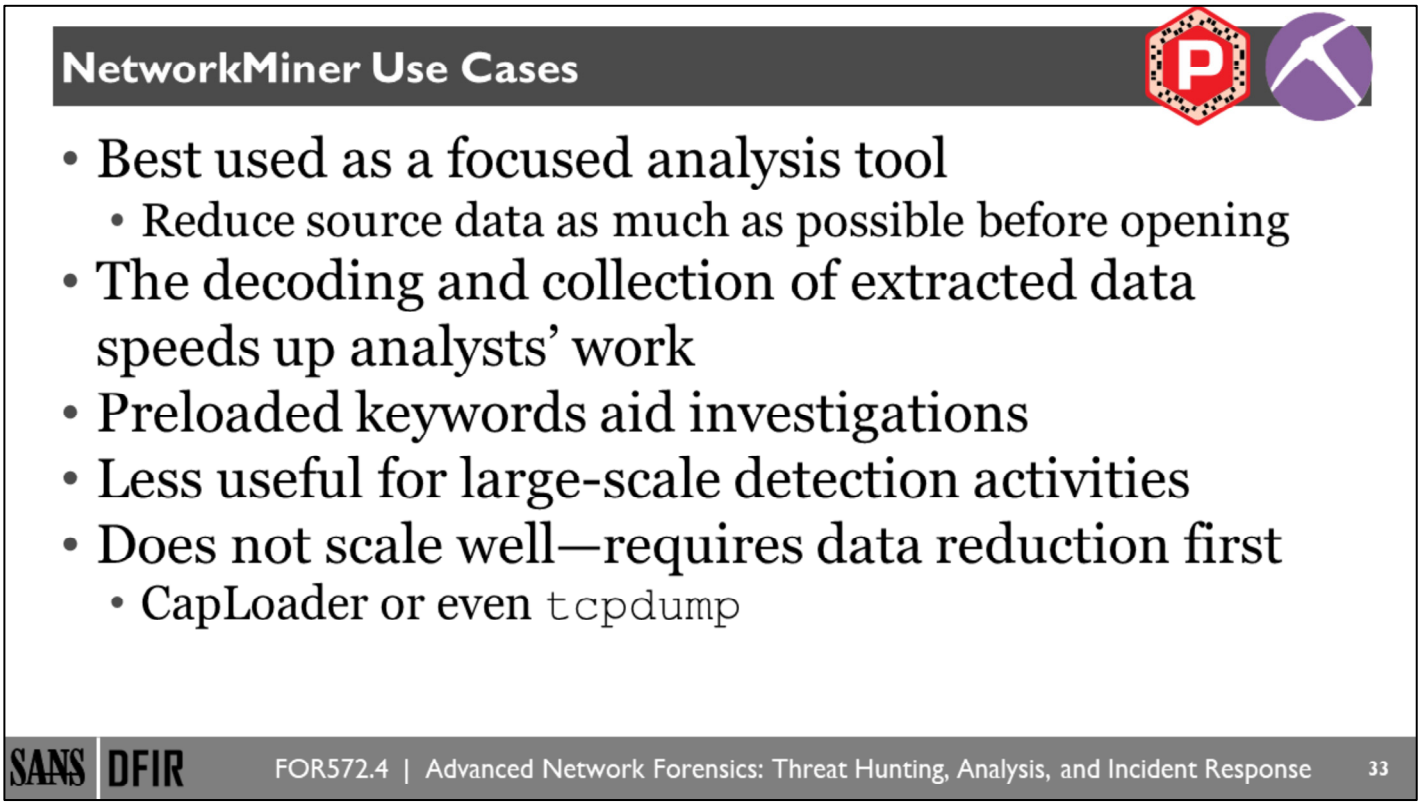
- Image thumbnails
- Credentials and cookies

When reviewing large collections of extracted images, it's often easiest to observe them at the same time. NetworkMiner provides this functionality on the "Images" tab. Here, no matter what host or protocol was used for the transfer, all reconstructed images are displayed via thumbnail renderings. From this panel, an analyst can open the original file, if desired.

Credentials are also displayed on their own tab. Where available, usernames and passwords are displayed. Additionally, any observed HTTP cookies are included. The reason they are included here is that most websites use cookies as the authentication method. Therefore, this becomes a uniquely identifying value.





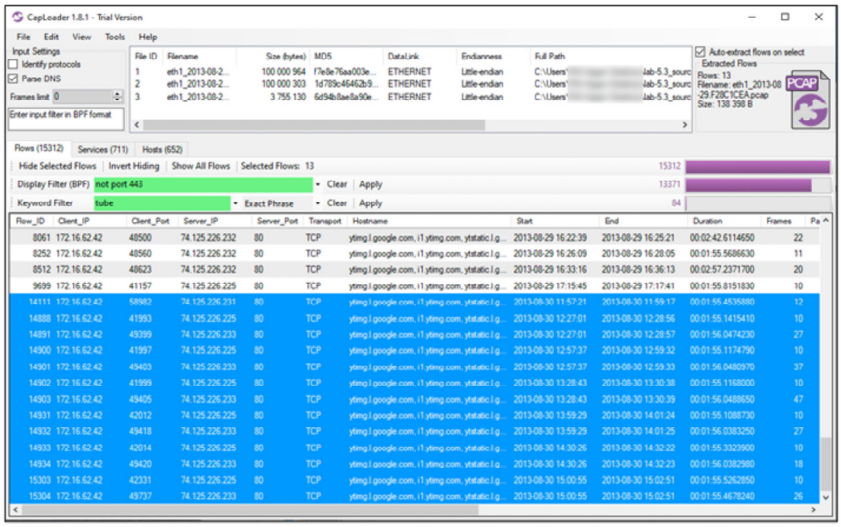


Given the extensive object extraction capabilities, NetworkMiner does suffer from somewhat slow processing speed when consuming very large packet captures. For this reason, it is best suited to use after data-reduction preprocessing with other tools – especially those that leverage the BPF. For example, using `tcpdump` to reduce the data loaded into NetworkMiner is an ideal workflow, allowing the analyst to use the best tool for each part of the task.

Although excellent for internal investigations, live investigations, and malware beaconing analysis, the tool does not scale to hunt potential APT activity, as it cannot handle the volume by itself.

CapLoader

- Reads, indexes large pcap collections
- Host/service discovery
- Export flows to NetworkMiner, Wireshark, etc.
- 30-day limited trial

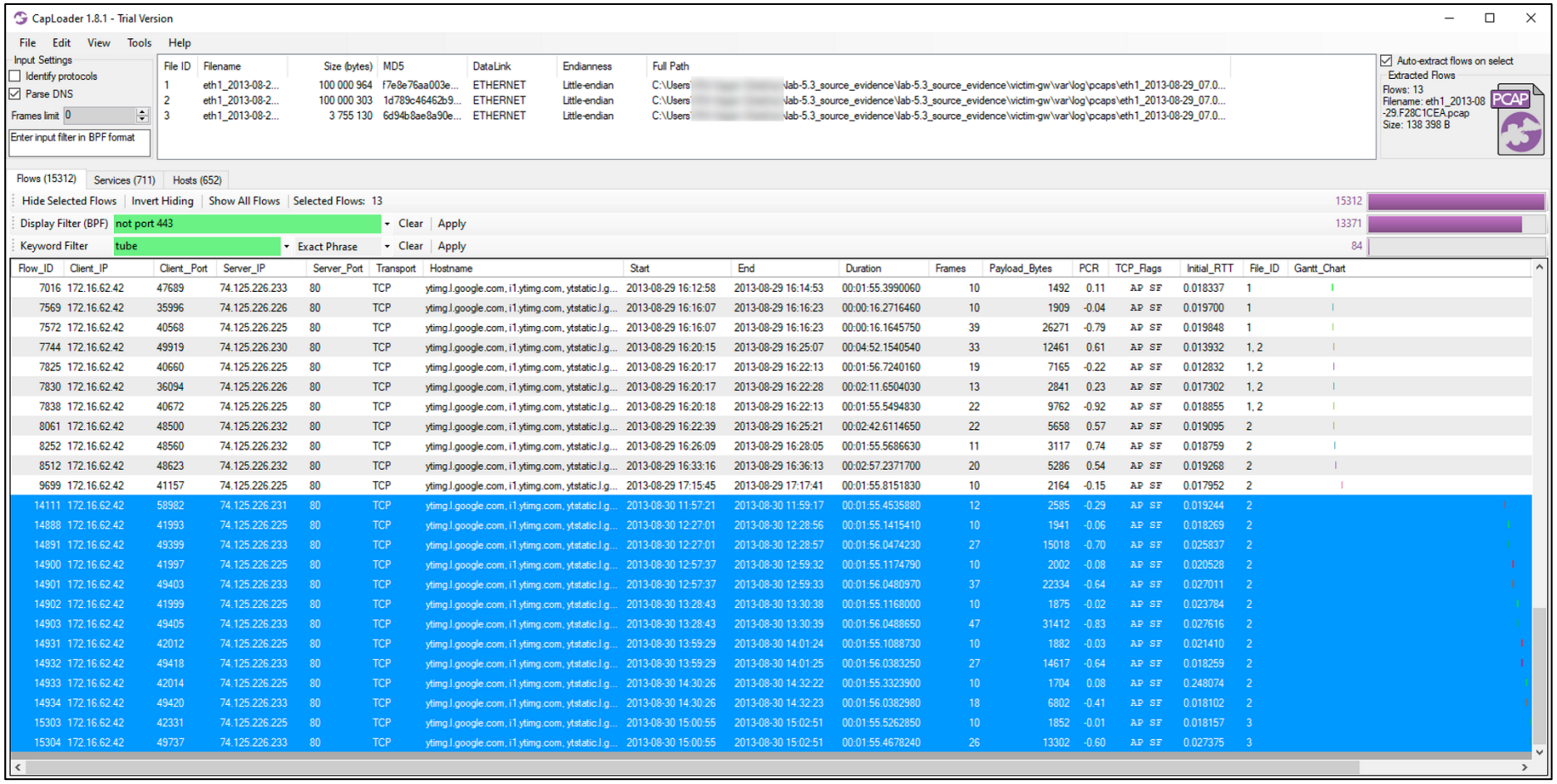


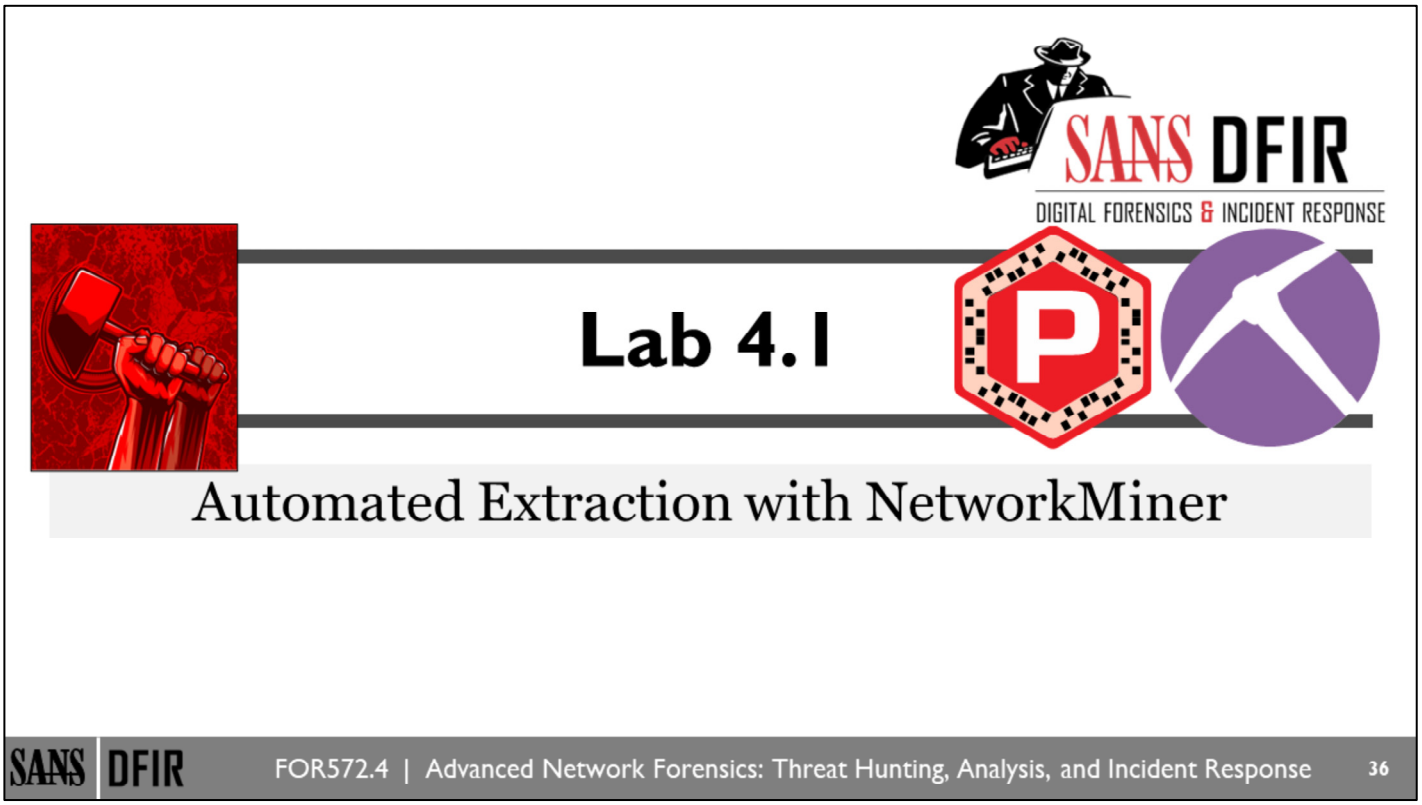
Netresec has also created a tool designed to handle large captures, called CapLoader.

After being loaded, all network traffic is indexed into flows and presented to the user. Individual flows, or all traffic to/from a host or using a specified protocol can then be exported to NetworkMiner for object extraction or to Wireshark for deep-packet exploration. This makes the tool a “GUI for data reduction” of sorts.

The tool is separate from NetworkMiner and is not a security tool. However, as a highly efficient data management tool, CapLoader can help to significantly speed up investigative and hunting workflows. There is a 30-day, feature-limited trial license version available.

References:
https://for572.com/6bs0a





This page intentionally left blank.

Lab 4.1 Objectives: Automated Extraction w/ NetworkMiner

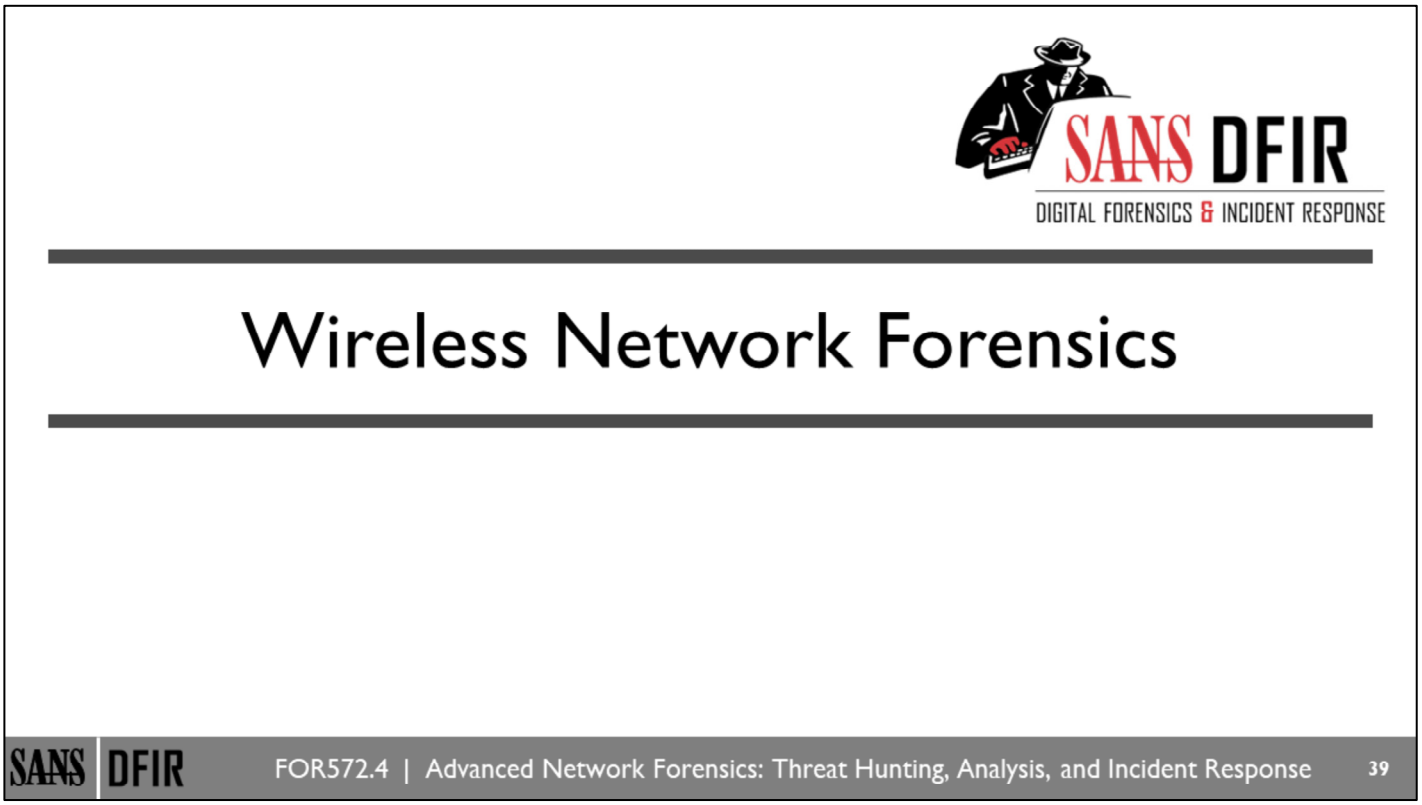
- Familiarize yourself with basic operations of commercial network forensics tools
- Load pcap into NetworkMiner, conduct analysis and mining of the data
- Consider scalability of network forensics tools in your analytic workflow

This page intentionally left blank.

Lab 4.1 Takeaways:Automated Extraction w/ NetworkMiner

- Commercial tools often provide a polished GUI, pre/post sales advice, and support
  - Exposed features limited by developer's preference
- NetworkMiner quickly extracts many different objects from pcap files
  - Comprehensive capabilities come with speed costs
- Knowing strengths and weaknesses of many tools can free up analysts' time and capacity

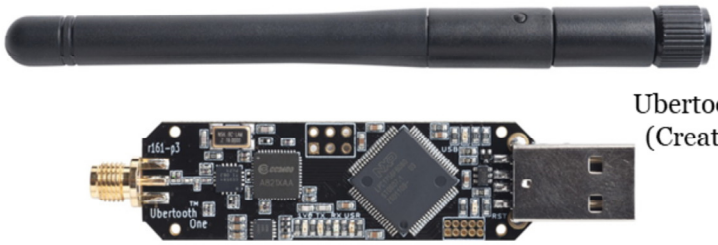
This page intentionally left blank.



This page intentionally left blank.

Limitations

- This is not a wireless security course!
- We focus on common 802.11 family of standards
- We will not cover ZigBee, Z-Wave, GSM, LTE, or Bluetooth…
  - But with the right hardware, Wireshark also parses these!



Ubertooth One BT/BTLE sniffer
(Creates BT/BTLE pcap files!)

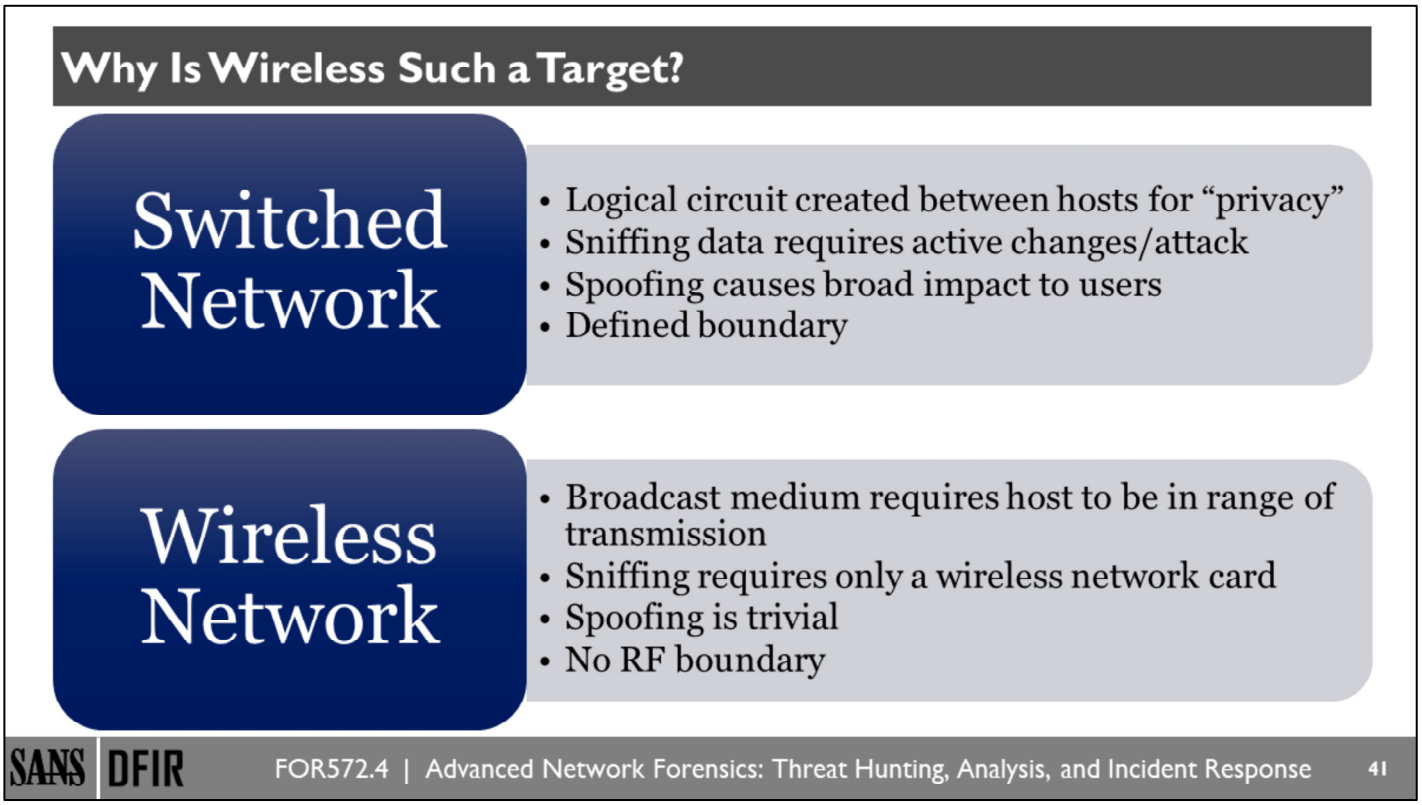
SANS DFIR FOR572.4 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response 40

With limited classroom time, we could never cover the full depth of this vast and complex subject without limiting ourselves to a reasonable set of protocols, medium, and frequency.

If you want to study this area further, we recommend you consider taking SANS SEC617, "Wireless Penetration Testing and Ethical Hacking,"[1] where you'll spend considerable time using the hardware and software to uncover the security issues of the likes of ZigBee, GSM, and Bluetooth.

However, you can apply your Wireshark skills to the Bluetooth and Bluetooth Low-Energy arenas using an Ubertooth One[2] device, which creates pcap files containing that traffic.

References:
[1] https://for572.com/g3ytb
[2] https://for572.com/1j302



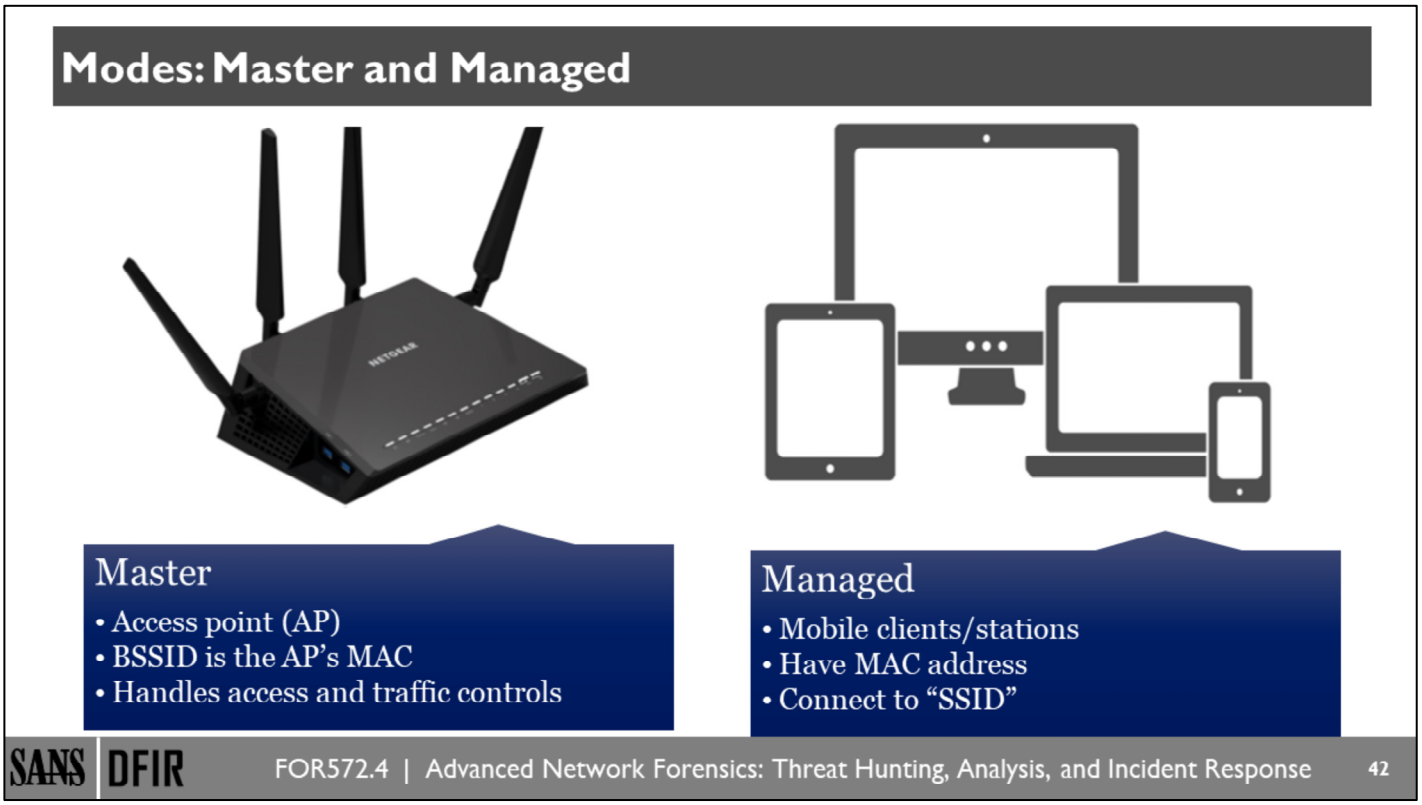
Wireless remains both a risk and exposed attack surface due to the nature of the medium through which it connects. Although wired communication has seen the introduction of switched communications as a method for reducing the number of hosts that can "see" the signal, wireless communication by its very nature is broadcast.

Although sniffing on a switch is made easier with tools like Cain and Abel, Bettercap, or Subterfuge, these attacks involve manipulating the environment in ways that have become readily detectable. Wireless sniffing, however, is completely passive and can often be conducted with the native OS and natively installed wireless card. If specialized long-range or sensitive equipment is required, it is openly available from online retailers for commodity prices. Similarly, injecting spoofed traffic to a wired/switched network generally requires a significant technology footprint and often causes a noticeable impact to the users in the victim's environment. Wireless attacks often cause an observable negative impact as well, but depending on the attacker's goal, it's quite feasible for them to achieve their goals in a short period of time—before users start to complain.

Finally, while wired attacks require the individual to be physically connected to the infrastructure (which generally requires them to be inside the target's physical spaces), wireless networks do not impart that limitation. Indeed, with inexpensive but tuned equipment, an attacker could be several miles away from the target network and still collect the data being transmitted.

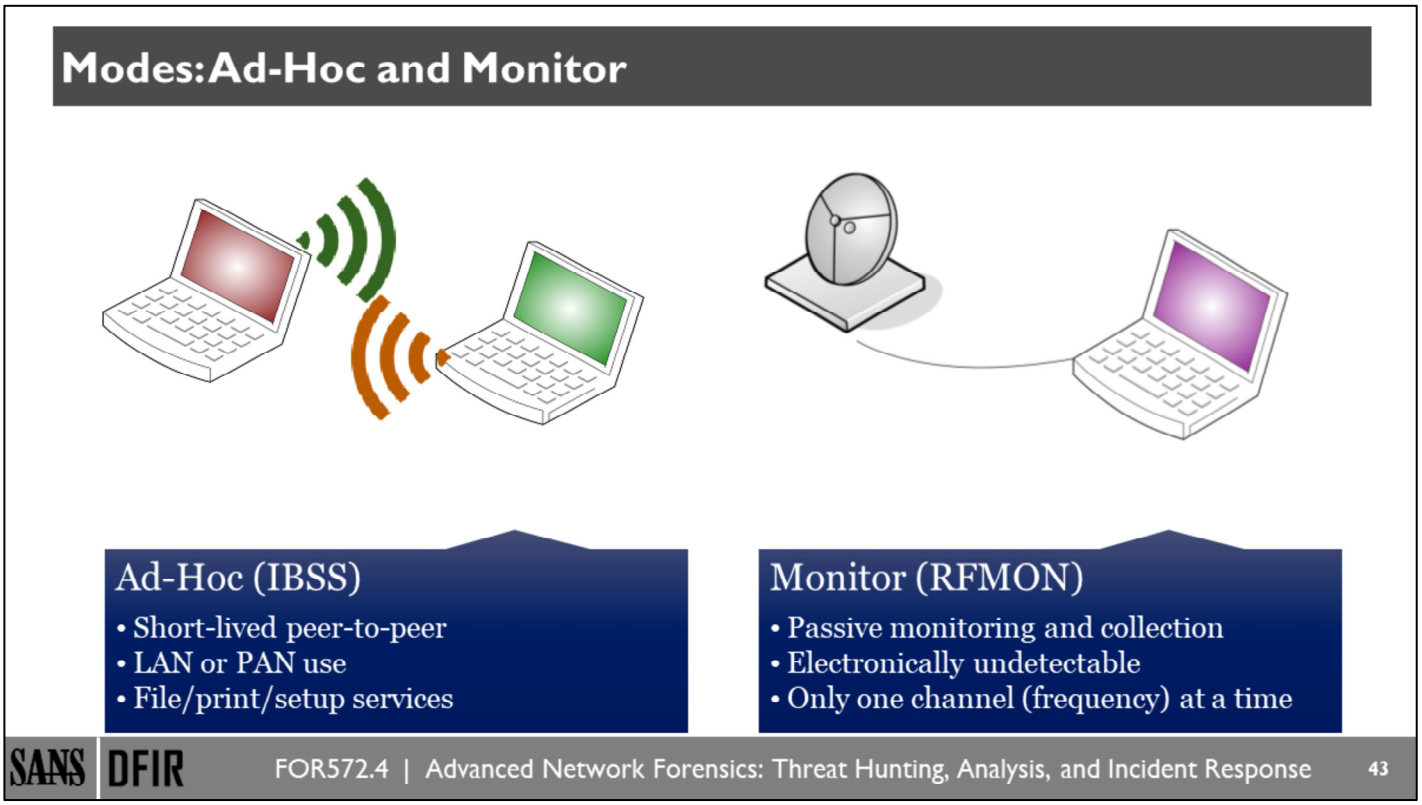
Actively injecting packets to the targeted wireless network would require the attacker to be in closer proximity to the victim's environment. However, in more densely populated areas like cities, a radius of 100 meters would include many businesses, apartments, and coffee shops—any of which could harbor a potential hidden attacker. Even in a more isolated environment, an attacker could easily conceal a miniaturized hardware platform and interact with it from a safe distance.

Put simply: Attackers focus on the weak links within their targets' environments. For all the preceding reasons, wireless is a decidedly weak link—and therefore a very attractive attack vector.



Master Mode or Infrastructure Architecture Mode
When a centralized device is given control of part of the RF Spectrum (the channel and SSID), it is deemed to be in infrastructure architecture mode. This device will be in **Master Mode** and for that BSSID, SSID, and frequency, it will determine who can communicate with it, associate to it, and following a successful challenge response (i.e., authentication), who can transmit and how much time they can transmit for.

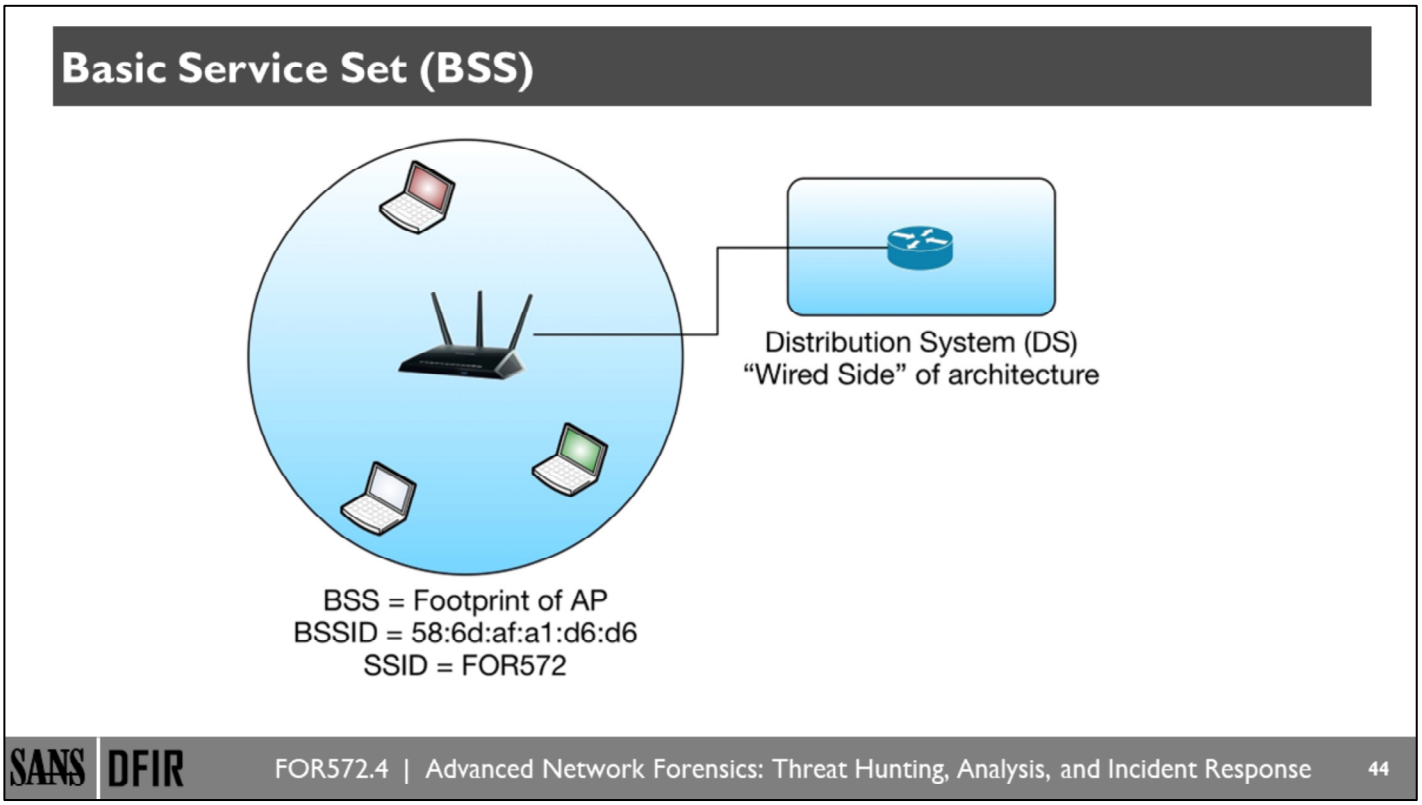
Managed Mode
Stations that connect to access points (APs) and submit to their control are deemed to be in **Managed Mode**. In this mode, they will act upon the instructions of the access point, including the encryption and authentication to be used. The AP can also instruct the station to disassociate from the access point. Managed Mode is the default operating mode of laptops and mobile devices that regular users operate. When in this mode, any network sniffing conducted by the client will reveal typical Ethernet type frames, packets, and data. In this mode, the client is not able to see any of the 802.11 management communications from the AP to the client's RF radio on their wireless card.



Ad-Hoc or Independent Basic Service Set (IBSS) Networks
When clients communicate directly with each other and don't rely on an AP to manage the wireless environment, they engage in ad-hoc networking. This is designed for point-to-point communication between devices to share small amounts of data. Common implementations are tethered phones, wireless printers, Apple's AirDrop file exchange protocol, and setup networks for embedded/IoT devices. Devices trust each other and try to manage communication between themselves by means of back-off algorithms to ensure that both get a fair share of the airwaves.

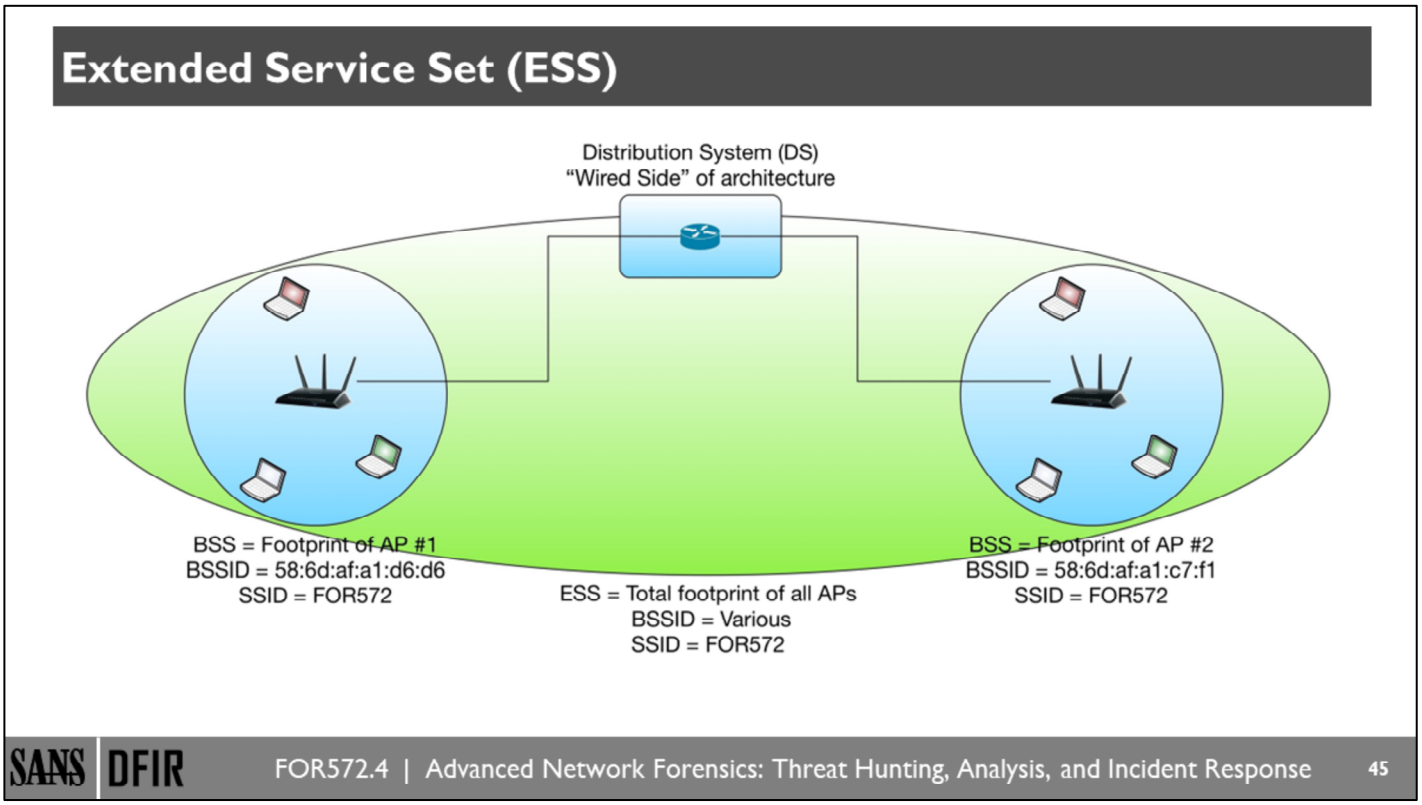
Monitor or RFMON mode
In the fourth mode of operation, the wireless card no longer transmits but instead listens to all 802.11 traffic on a set frequency. An RFMON interface itself is not dangerous; however, if a packet capture utility is connected to the monitor interface, then ***all*** 802.11 traffic observed by the network card's radio on the frequency it is listening to will be captured. This includes the other network and RF management and control frames transmitted by the target AP and all other 802.11-enabled devices on that frequency within the range of the sniffing station.

RFMON is completely passive and cannot be electronically detected by others in the vicinity. The downside, from the attacker's point of view, is that they can listen only to one channel/frequency at a time. To address this, RFMON-focused tools may hop through multiple channels or easily permit the simultaneous use of multiple network interfaces to cover as wide a swath of the radio frequency spectrum as possible.



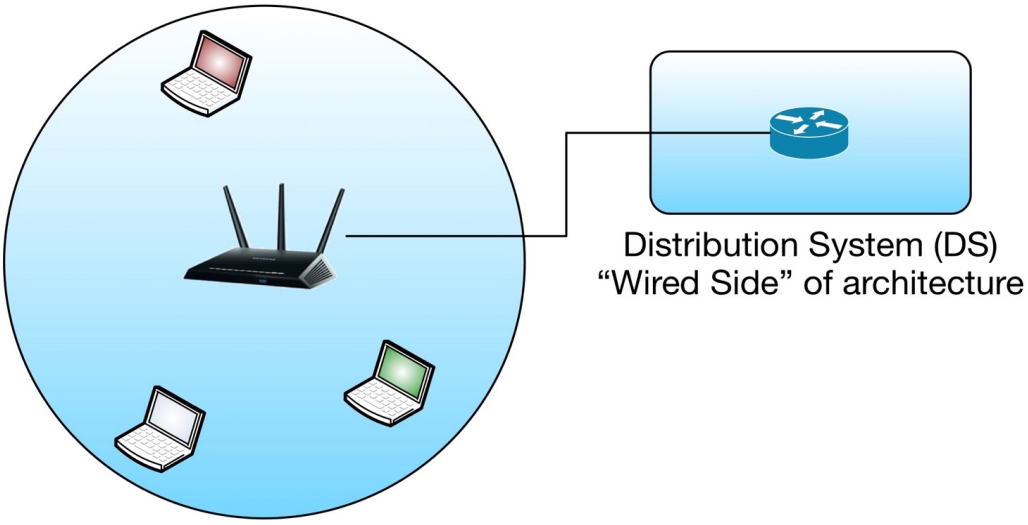
In IEEE terms, there are several other important variations of this implementation.

The Basic Service Set (BSS) is the core of the 802.11 wireless networking concepts. It is a collection of one or more stations that are associated with an access point. The BSS can be identified by the BSS Identifier or BSSID, which is the MAC address of the access point to which all the stations are connected.

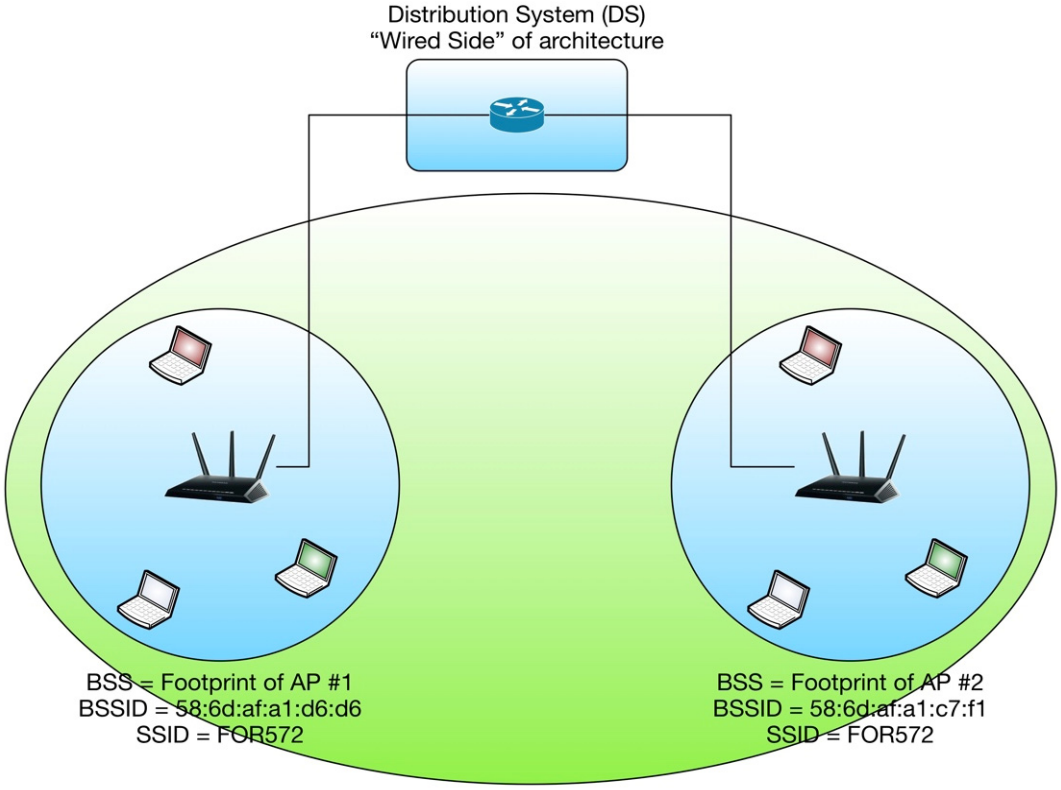


A collection of access points with the same SSID that provides access to a common infrastructure and the stations that are associated with the APs is known as an Extended Service Set (ESS).

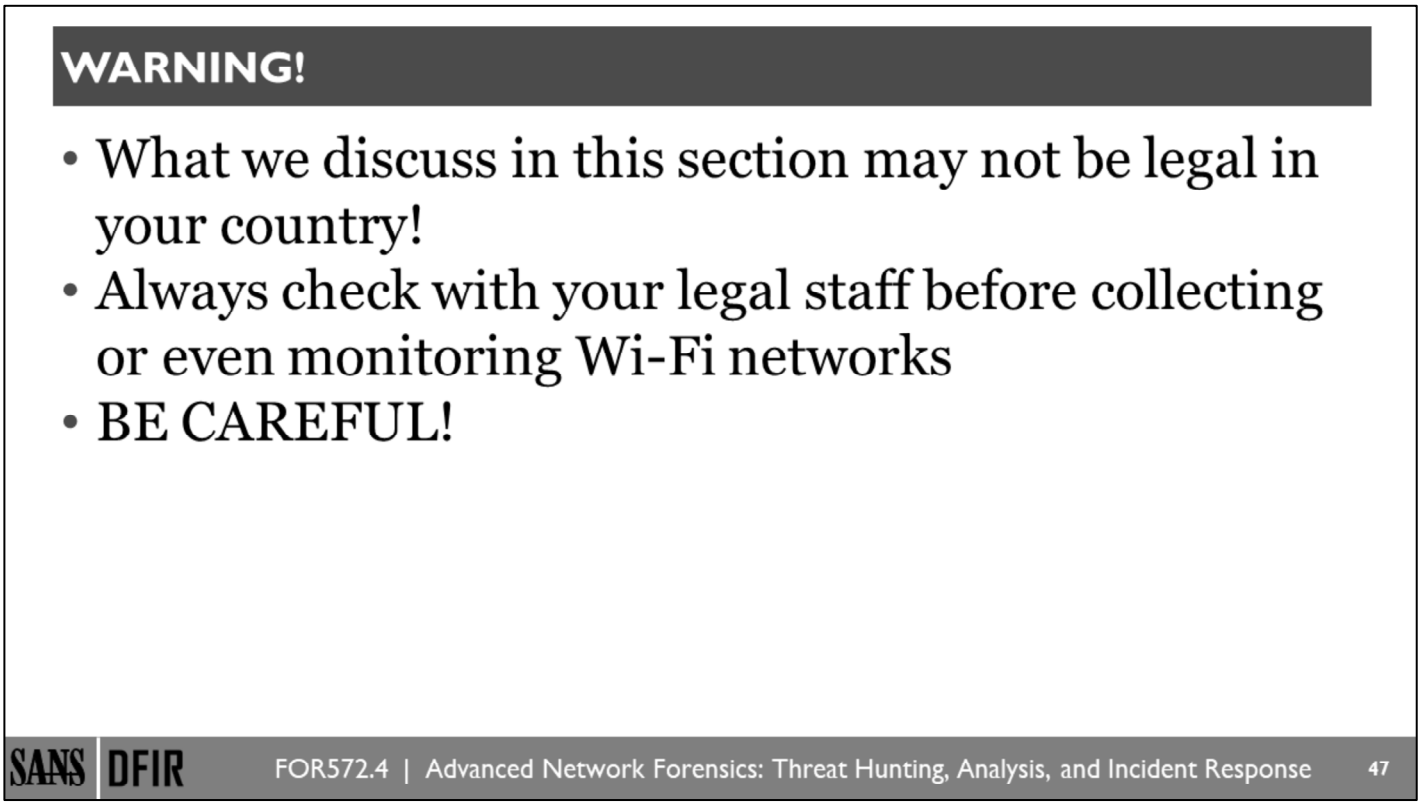
These are commonly deployed by organizations (and many homes) as users seek a seamless roaming experience. An ESS allows stations to move about the building connecting and reconnecting to different APs while still accessing internal resources.



BSS = Footprint of AP
BSSID = 58:6d:af:a1:d6:d6
SSID = FOR572



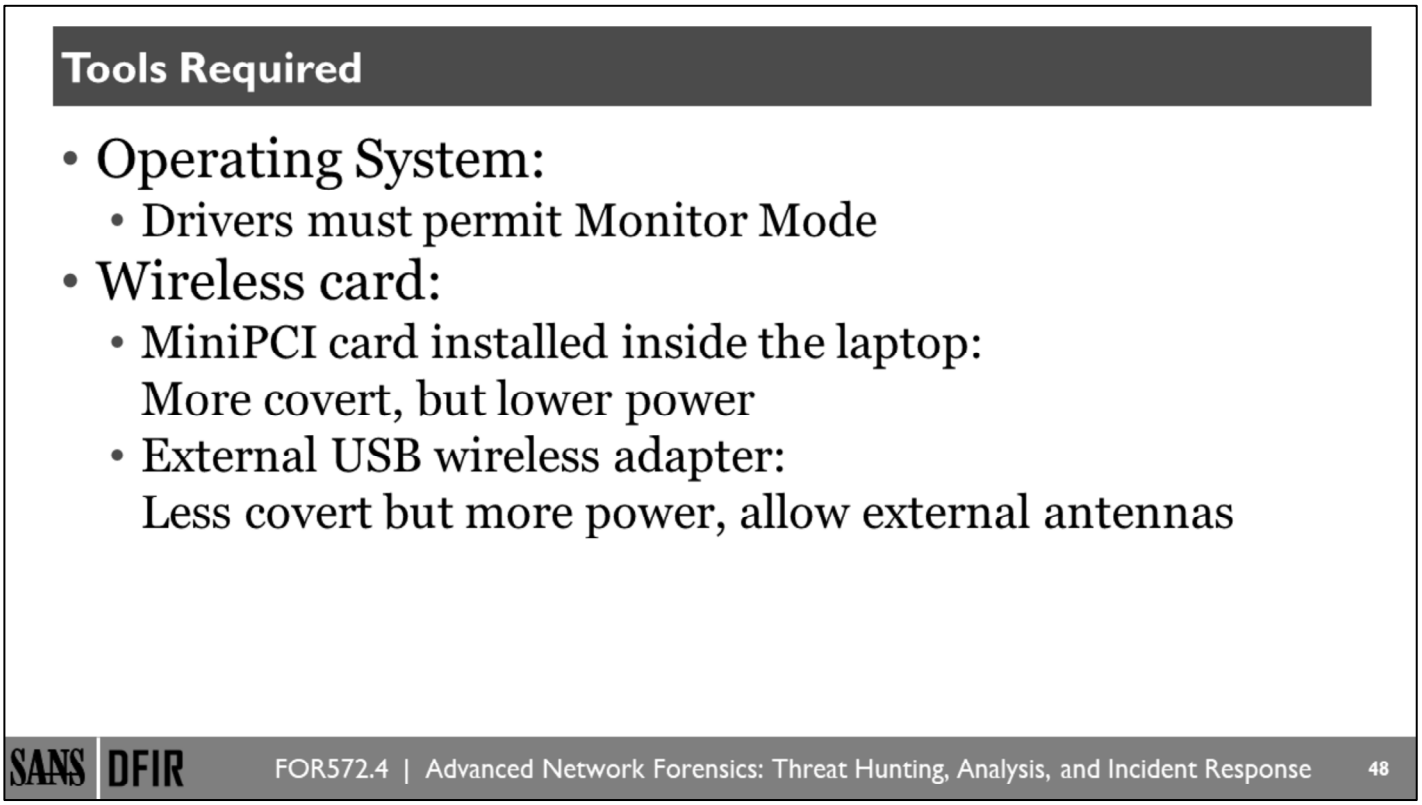
ESS = Total footprint of all APs
BSSID = Various
SSID = FOR572



Please note: In some parts of the world, even having these tools and hardware could get you arrested and potentially subject to fines or other punishments.

Always understand the laws of the country you are in, and those you are traveling to. If you do not require these tools (e.g., on vacation), consider leaving them at home.

If you require them for work purposes, consider having a letter of introduction that specifies your need for having the tools, what the tools are, and why you have them in that country/location.



The first consideration when selecting a wireless card for passive capture purposes is to ensure the driver provides monitor mode feature. While many modern drivers do allow this function across various operating systems, there are still several that do not. Confirming this before selecting the device will save potential headaches in the future. Most Linux OS distributions have good wireless manipulation tools built in, but more importantly, they provide better control of the wireless adapter through better driver support and the inclusion of libpcap by default.

Windows can be augmented with drivers for certain wireless cards; for example, the Riverbed AirPcap USB wireless card[1] is shipped with Windows drivers that will allow for promiscuous mode sniffing of 802.11 traffic.

When considering a wireless card choice, although internal cards are more covert, you are limited in that many laptops have only one internal MiniPCI slot. You may be fortunate that your laptop manufacturer has included an Atheros chipset card, which is suitable for most wireless capture activities (in a Linux OS).

With USB wireless adapters, you are not limited by slots or space, only by USB ports. Although some laptops can run only one device per port, others can power USB hubs with multiple USB wireless adapters. This can provide monitor mode visibility on multiple channels simultaneously.

Although entry-level analysis can be conducted with just one capable card, you'll likely want to invest in a good collection of USB wireless cards if you are performing wireless forensic examinations or collecting wireless data on a strategic basis.

References:

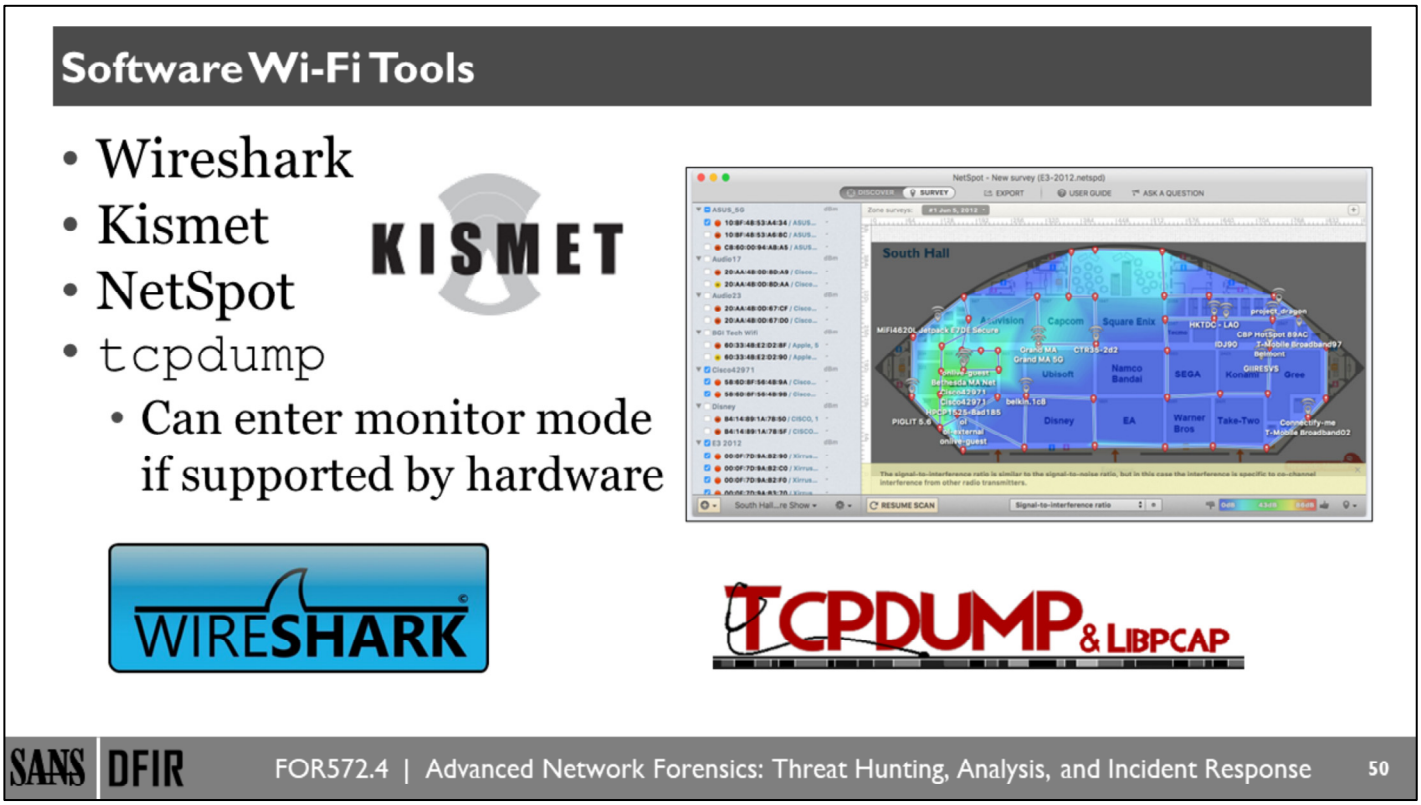
[1] https://for572.com/l0oym

Favorite USB Adapter – Alfa

- Alfa AWUS036ACH USB 802.11 a/b/g/n/ac adapter
  - 2.4 and 5GHz support
- External RP-SMA antenna connections for versatile uses
  - 2x 5dBi omnidirectional shown
  - Also high-gain, unidirectional, etc.
- Realtek RTL8812AU chipset
  - Uses Linux rtl88xxau driver

Perhaps the most popular hardware kits used in both strategic and tactical wireless network collection and monitoring operations are from the Alfa company. Although international availability is occasionally limited, the comparatively low price and the high power/high gain feature set cause investigators and penetration testers worldwide to seek out its gear.

These devices can use a variety of different antennas to best suit various use cases, whether covert or overt. The driver support is quite good on Linux and similar operating systems (including Apple's macOS). Drivers are also available for the Microsoft Windows family of operating systems, though not all audit/capture/packet injection features are supported on all platforms.



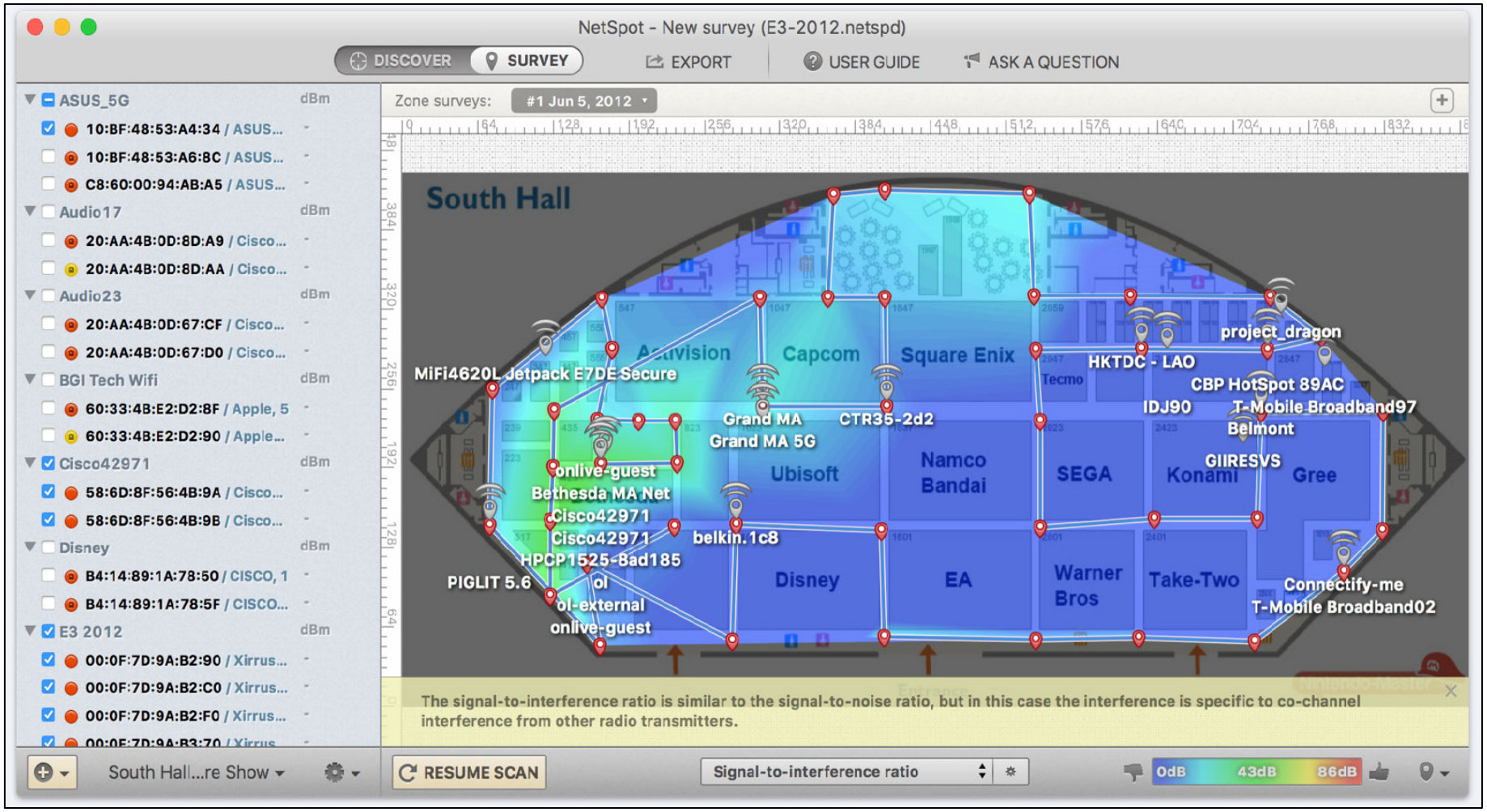
There are a variety of software tools useful for analyzing 802.11 packets and traffic, including:

- **Wireshark**[1]: This staple tool can also parse many fields from 802.11 wireless traffic.
- **Kismet**[2]: Kismet is a passive wireless monitoring tool, potentially allowing users to conduct surveys in areas where they are not licensed or authorized to transmit but still legally permitted to collect. Kismet supports GPS input for mapping purposes.
- **NetSpot**[3]: This is a mapping utility for Windows and macOS that can use an image as a background for mapping wireless coverage and available networks. It's quite ideal for locating rogue access points and dead spots alike.
- **Aircrack-ng**[4]: Aircrack-ng is a fork of the original Aircrack tools that were developed to assist in the auditing of WEP and WPA implementations. The tools have been expanded over the years and now include the code from tools like coWPAtty and AirGraph-ng.
- **tcpdump**[5]: Even tcpdump can now handle 802.11 traffic. Recent versions of this staple to every network forensicator's toolbox can handle acquiring wireless traffic when the network interface is in monitor mode.

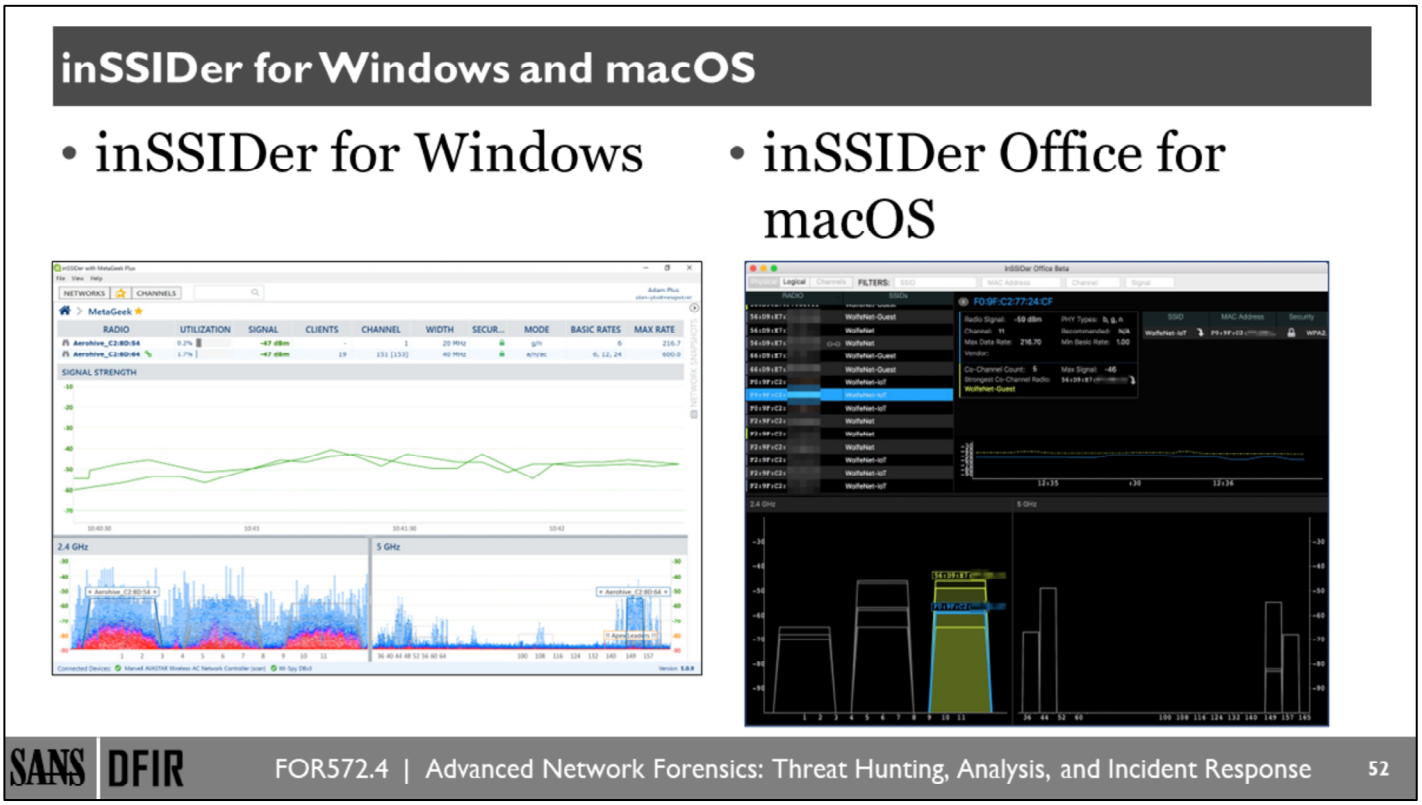
In this course, we will be focusing on Wireshark, because the other tools are best used to capture live traffic for analysis. However, understanding the other tools and how they can be useful in identifying malicious activity is important for many wireless network forensic use cases.

References:

[1] https://for572.com/3b85e
[2] https://for572.com/7d1z9
[3] https://for572.com/mcqvb
[4] https://for572.com/hk7o0
[5] https://for572.com/7itb1



https://t.me/learningnets



The Metageek company provides several software and hardware products. While these are generally targeted toward wireless network design, engineering, and troubleshooting/optimization, they are also very useful for locating rogue actors and incident sources of interference in the wireless environment.

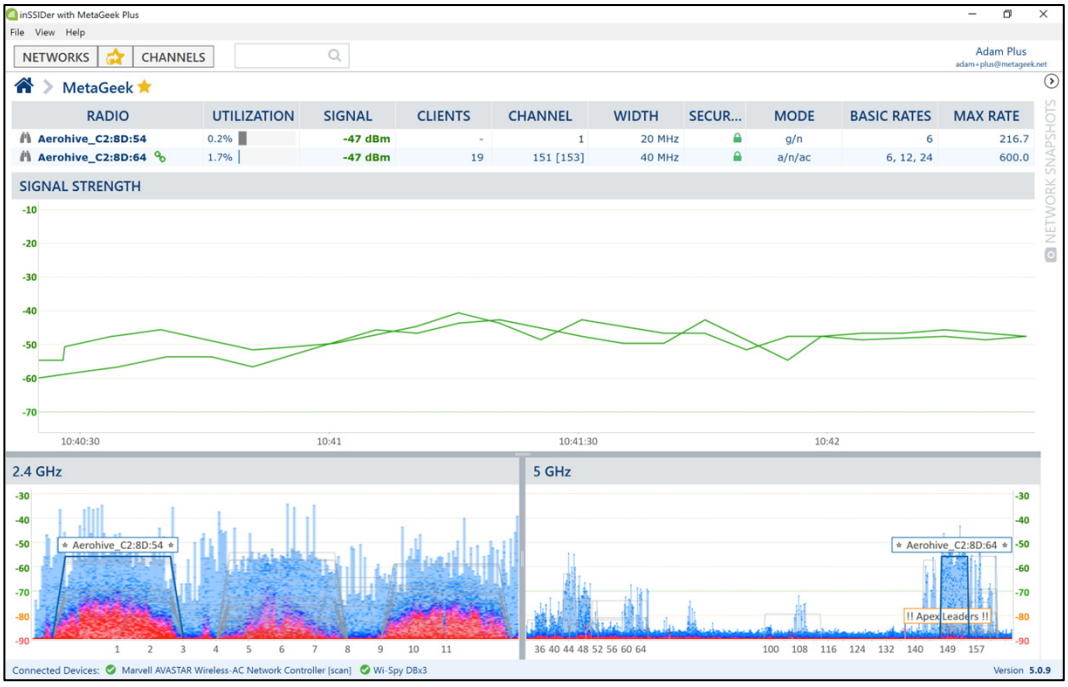
The following are some of the Metageek family of Wi-Fi tools:

- inSSIDer Office[1]
- Chanalyzer/Wi-Spy[2]
- Eye P.A.[3]

Metageek has also maintained updates for their inSSIDer software for Windows and macOS. The latest version of the Windows variant, inSSIDer 5, combines typical access point scanning with more in-depth frequency analysis features. This in-depth examination is invaluable for finding sources of interference but was previously reliant on expensive hardware. With the version 5, this feature set is available on much more accessible commodity hardware. While the macOS version is still at version 4, it still provides useful insight into the operating environment's 802.11 characteristics.

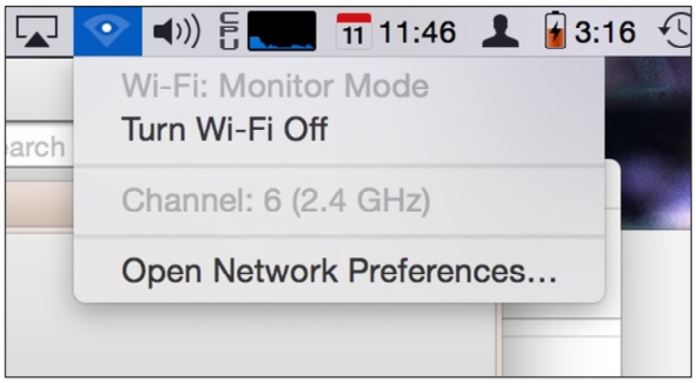
References:

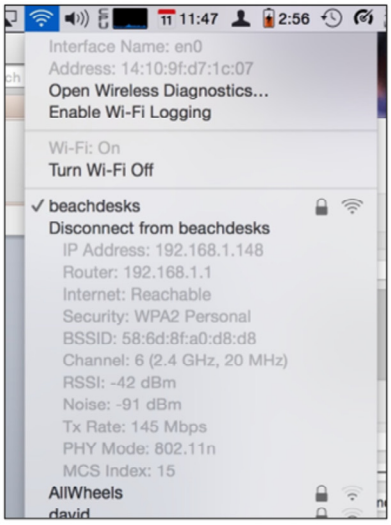
[1] https://for572.com/4i0t6
[2] https://for572.com/31w6d
[3] https://for572.com/4g89z



Built-In macOS Wi-Fi Tools

- Apple macOS has built-in monitor mode and easily accessible diagnostics





Apple's macOS provides a handy sniffer built into the `airport` tool (which is buried deep in the filesystem). The binary is located in the following location (for macOS Sierra – other versions may vary):

```
/System/Library/PrivateFrameworks/Apple80211.framework/Versions/⏎
    Current/Resources/airport
```

The syntax to use the sniffer is simply '`airport <device> sniff <channel>`'.

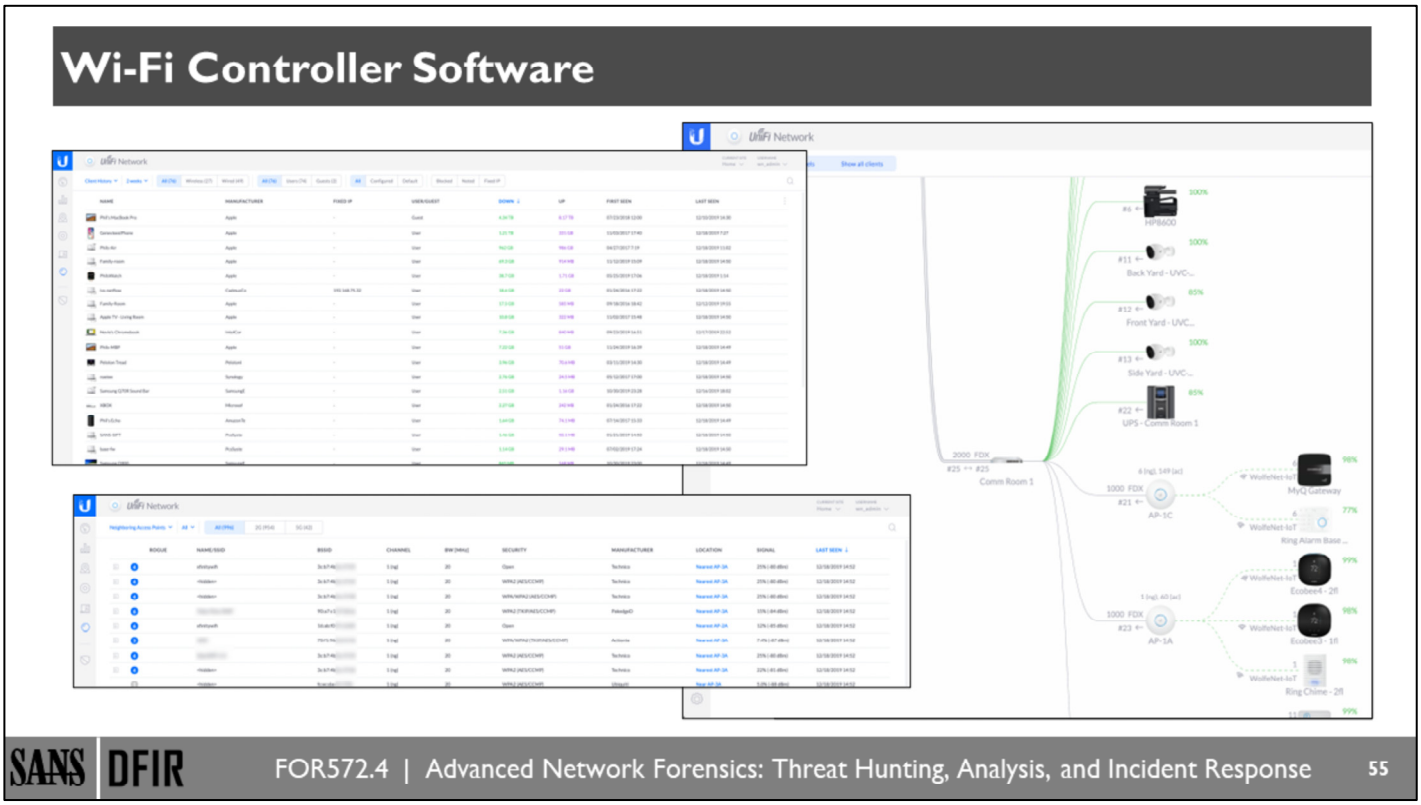
```
$ cd /System/Library/PrivateFrameworks/Apple80211.framework/Versions/⏎
    Current/Resources/
$ sudo ./airport en0 sniff 6
```

You will notice the regular wireless icon changes (see the left image on this slide) to what some refer to as "The Eye of Sauron" (in deference to the *Lord of the Rings* trilogy's all-seeing eye).

The output is only written to a file when the process exits. The output files are written to the `/tmp/` directory and have the following format:

```
/tmp/airportSniffXXXXXX.cap
```

One additional handy tip for Mac users: When connected to an access point, if you hold the "Option" key before clicking the Wi-Fi icon, you will get the technical details of the device to which you are connected. This "advanced settings" view is in the screenshot on the right.



Many wireless hardware vendors provide a means for centrally managing the infrastructure configuration, broadly classified as a wireless "Controller". While historically reserved for extremely large and complex networks, this capability has begun to permeate the mid-level market as well. One such example that has gained a large following in the past few years is the UniFi[1] product line from the Ubiquiti[2] company. The controller is provided free of any licensing and can manage anywhere from a single access point and router to an environment of hundreds of access points and switches across multiple clients and physical sites. This flexibility makes the UniFi product line an attractive one for network hobbyists, wireless MSPs, community ISPs, and large organizations.

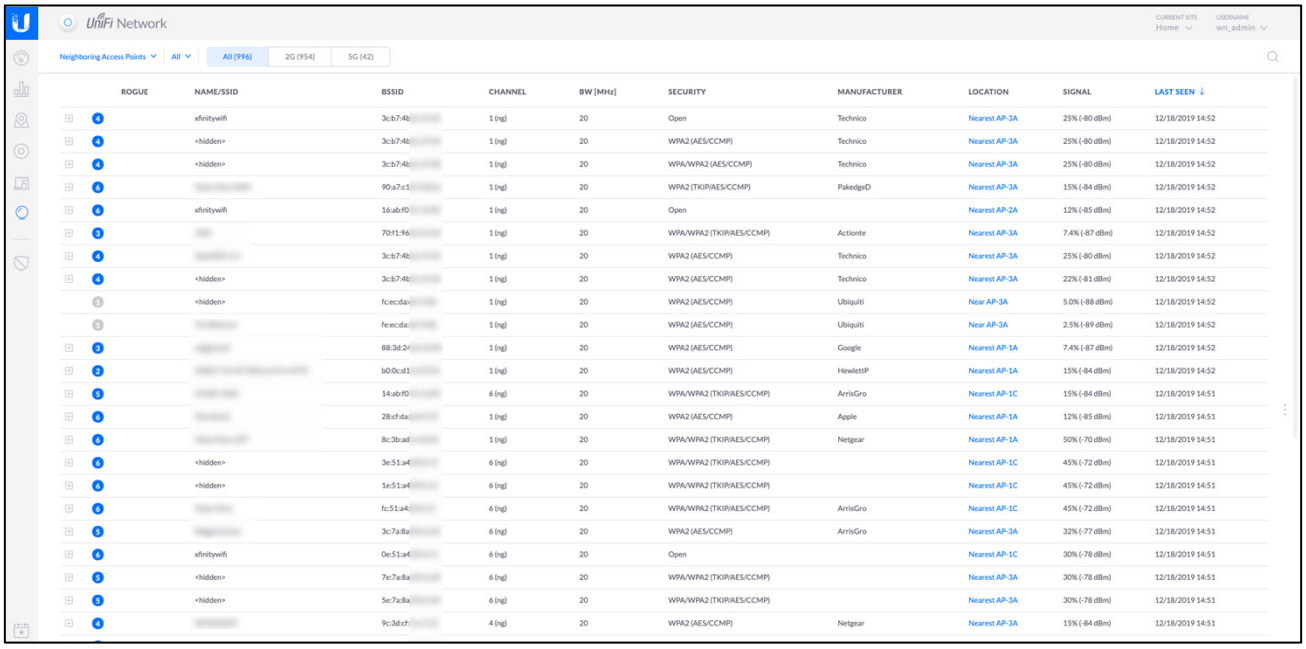
A controller-based network provides an investigator several forensic benefits. These may include:

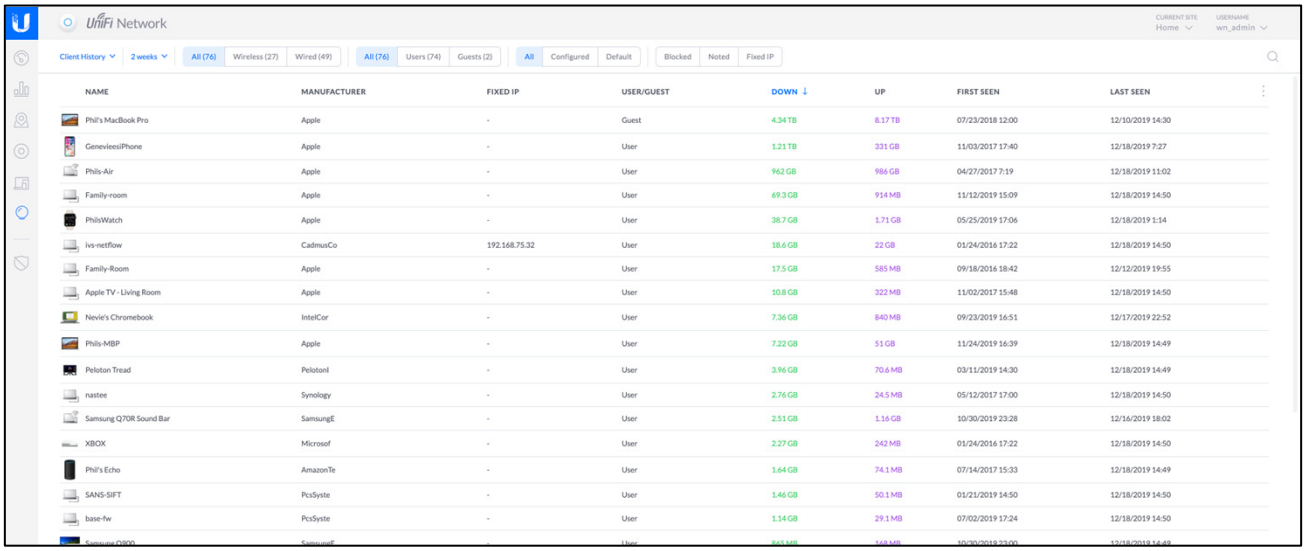
- Rogue access point detection
- Neighboring access points that can provide environmental baselines
- Client inventory and behavior profiling
- Signal strength logging per access point and client
- Channel changes due to congestion or interference
- Client movement based on access point association

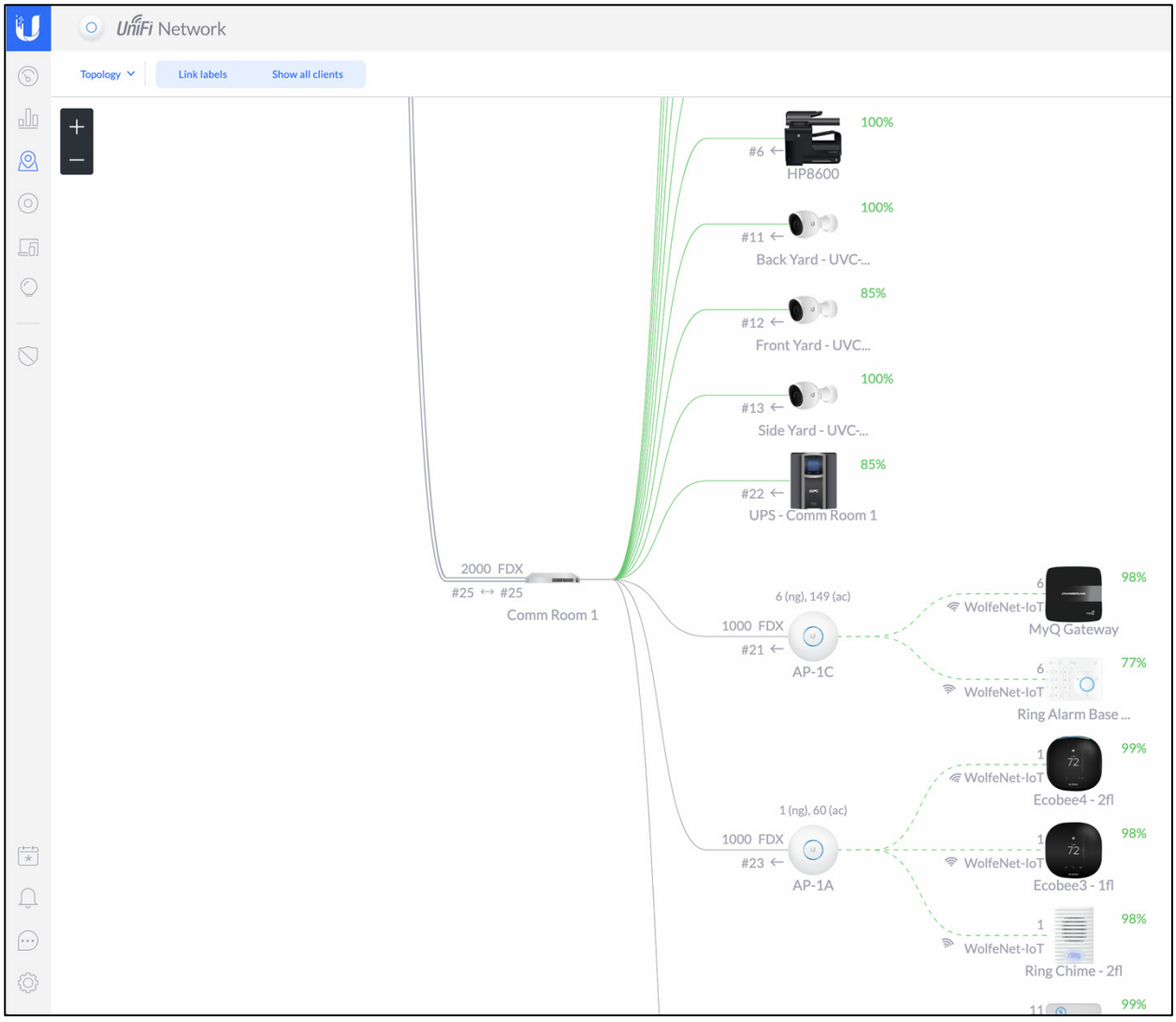
Since most controller software can also send log events via syslog, these can also be integrated to an organization's logging infrastructure for more streamlined analysis.

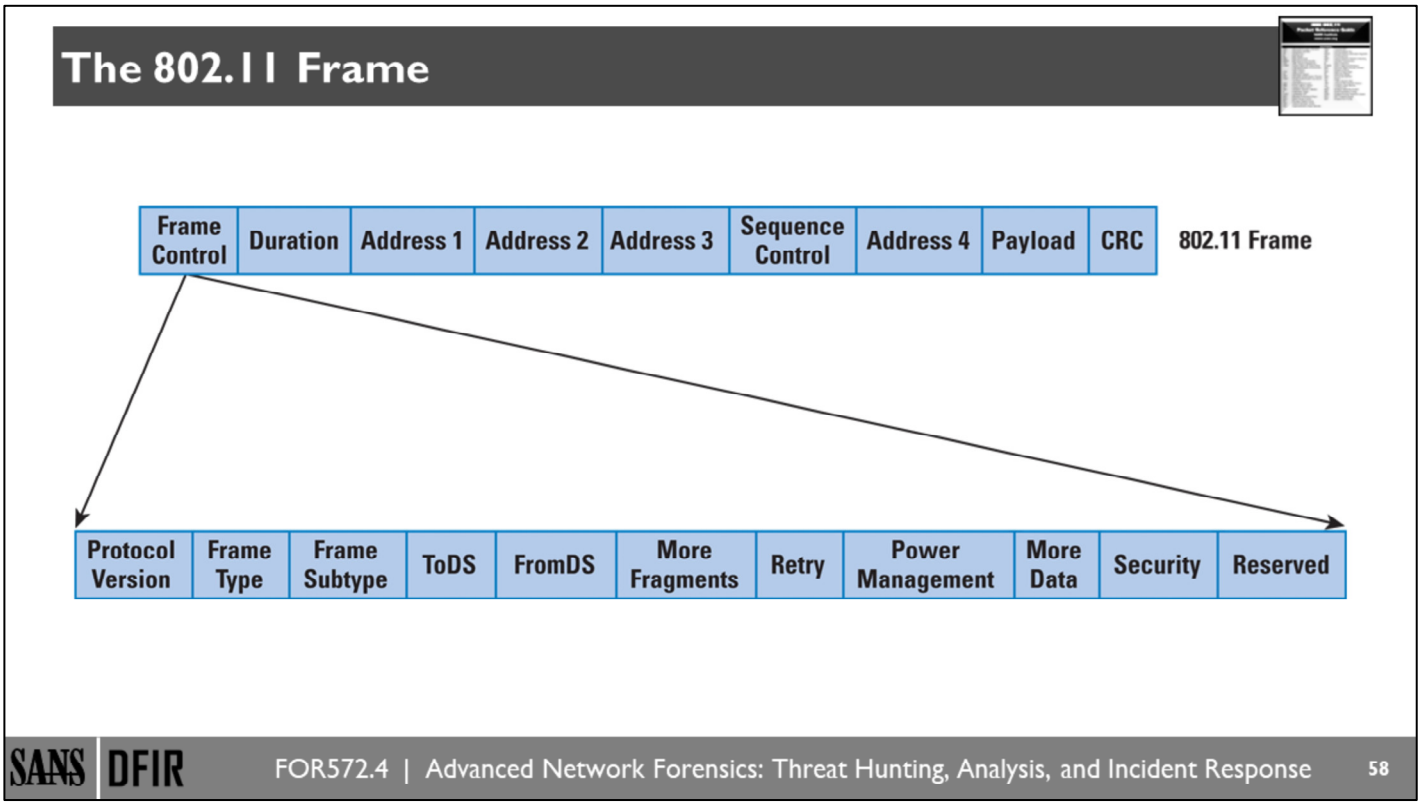
References:

[1] https://for572.com/xgod3
[2] https://for572.com/7ptj-









Note: When you see the icon shown at the top right corner of this slide, refer to the SANS 802.11 Pocket Reference Guide that was provided with your course materials. It can also be downloaded from Josh Wright's website.[1]

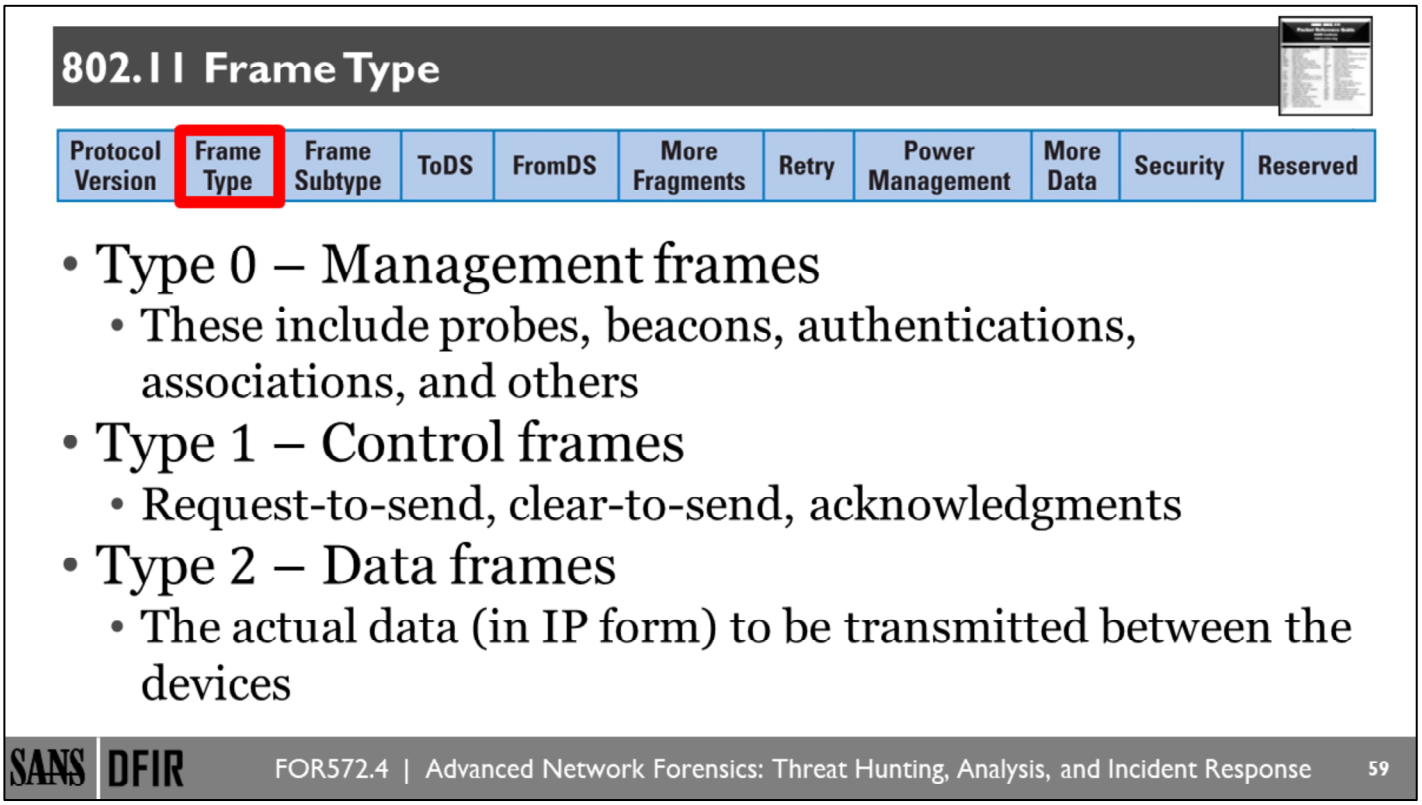
The 802.11 frame is like most OSI-based protocols in that it is layered and contains nested data. Of particular note is that 802.11 is a Layer 2 framing protocol, meaning it replaces the Ethernet header. This framing is performed at the wireless driver, so higher-layer tools may not have the knowledge that the packet was/will be sent across a wireless connection. This is important because it may change the analyst's understanding of the traffic, depending on what utility was used to perform the traffic capture.

The majority of the control information is contained in the first section called "Frame Control". In this section, the type and subtype of frame are defined allowing the receiving station to understand what it should do with the contents.

Over the next few slides, we will cover the key parts of 802.11 frames so that you as an analyst can move seamlessly from the wired to the wireless environment.

References:

[1] https://for572.com/6sto-



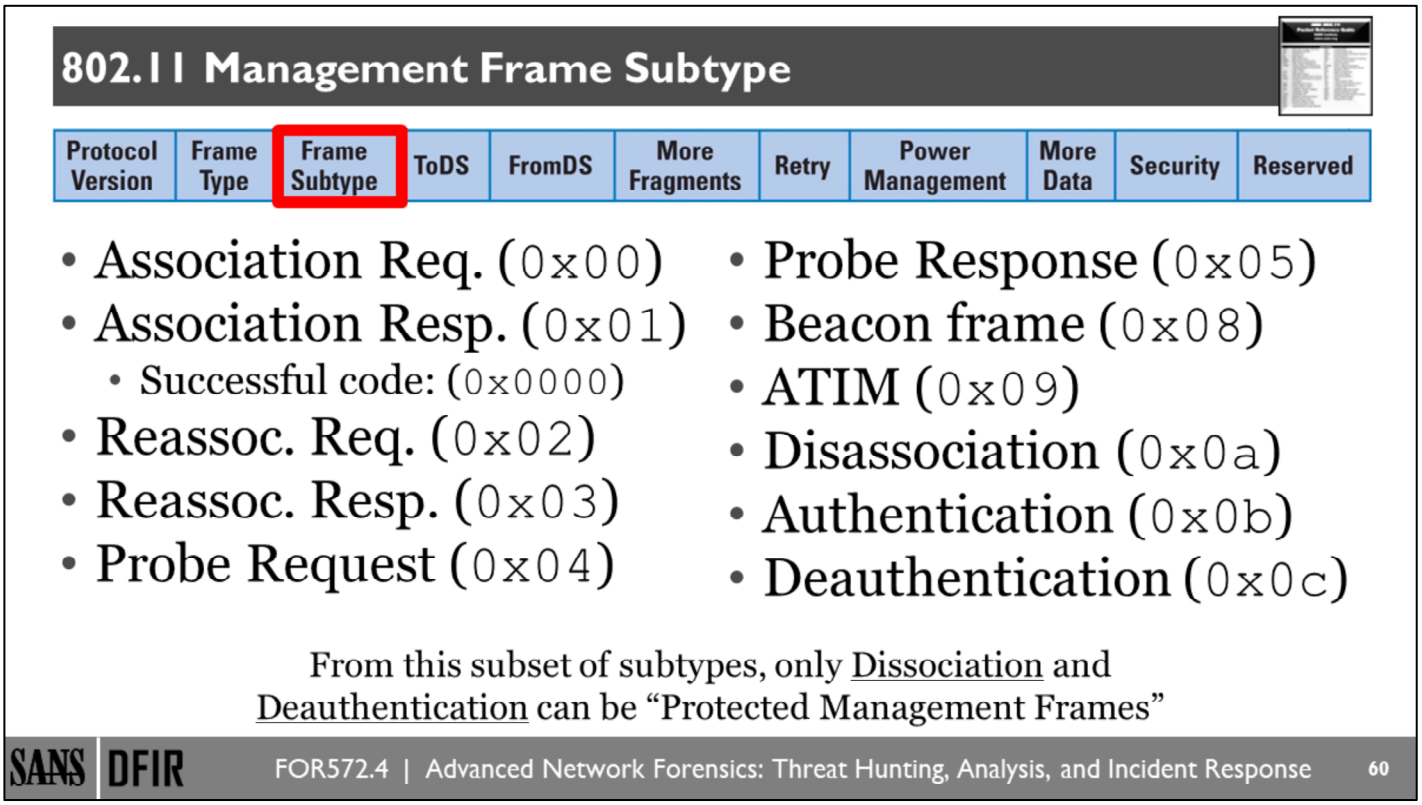
There are three types of 802.11 frames:

- Management frames
- Control frames
- Data frames

Unless you have a card that is able to go into Monitor mode, you will never see the management or control frames nor will you see the 802.11 headers of the data frames.

- **Management frames** are used to manage the BSSID's service area (i.e., its BSS). Management frames control who can connect to an AP and who can stay connected to the AP, thus they are limited to the associated (or attempting to associate) stations.
- **Control frames** are used to manage the medium (RF in this case). Control frames are transmitted to everyone regardless of the BSSID they are associated to as control frames manage who can communicate on the shared medium. Although the inherent forensic value of these frames is low, they often provide valuable insight into other activity on the wireless network, such as congestion and certain denial-of-service methodologies.
- **Data frames** are the actual data that the whole network is there to move from host A to host B. Data frames are addressed to individual nodes on the wireless network and have upper TCP/IP protocol data (i.e., the IP packets) in the payload.

References:
https://for572.com/f4oa9



Management frames, as their name suggests, are used to manage the communications on any WLAN, whether it's an AP beaconing out its supported speeds and SSID or a station sending out a probe request looking for an AP to which it was previously connected.

Management frames are important from a forensic point of view for several reasons:

- Management (and Control) frames are usually not encrypted
- They allow clients and APs to be located by triangulating their power or by manually plotting locations and relative power settings
- Attackers will look in these frames for information about the BSS's and ESS's capabilities
- Attackers manipulate APs and their clients by injecting or corrupting these frames

Therefore, knowing how to capture these frames and then decode and understand their contents are important skills when handling a wireless-based incident or investigation.

Newer Wi-Fi standards do provide the option for Protected Management Frames[1], but only a subset of subtypes can be protected. In particular, the frames that are used prior to and during the four-way handshake process must remain unencrypted. The IEEE 802.11w standard defines the functionality in total, but the following frames are NOT capable of protection under this standard:

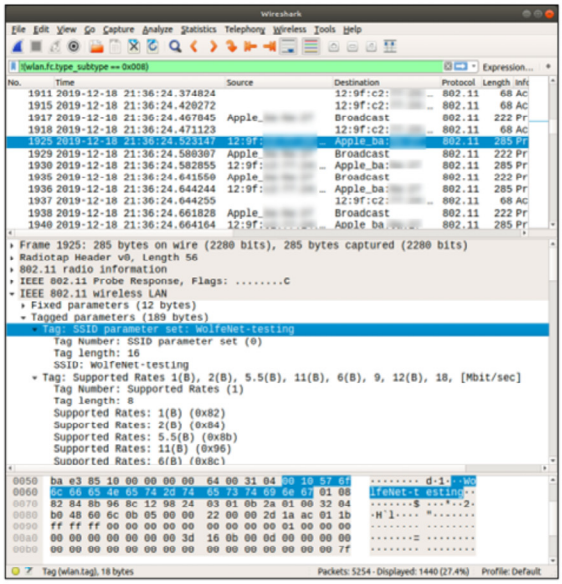
- Beacon
- Probe request
- Probe response
- Announcement Traffic Indication Message (ATIM)
- Authentication
- Association request
- Association response
- Spectrum Management Action

References:

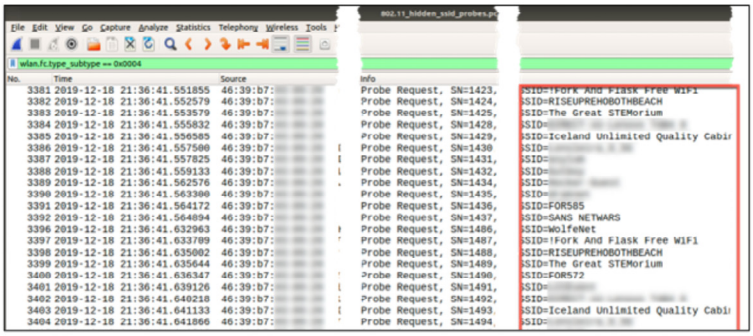
https://for572.com/4kdna

[1] https://for572.com/ob0xc

802.11 Fields: SSID and Other Tagged Parameters



- SSID: Service Set Identifier
  - Never truly hidden
  - Stored networks are broadcast

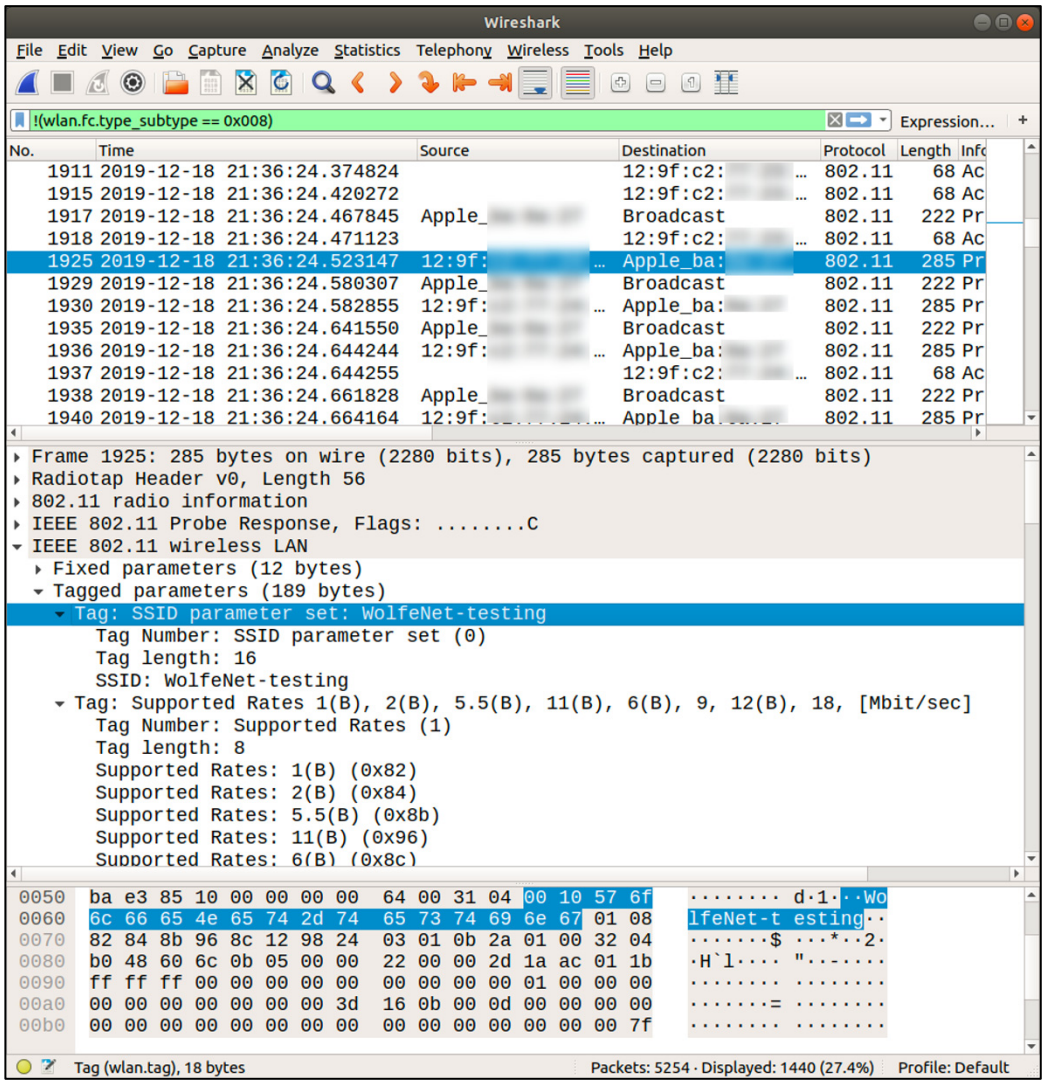


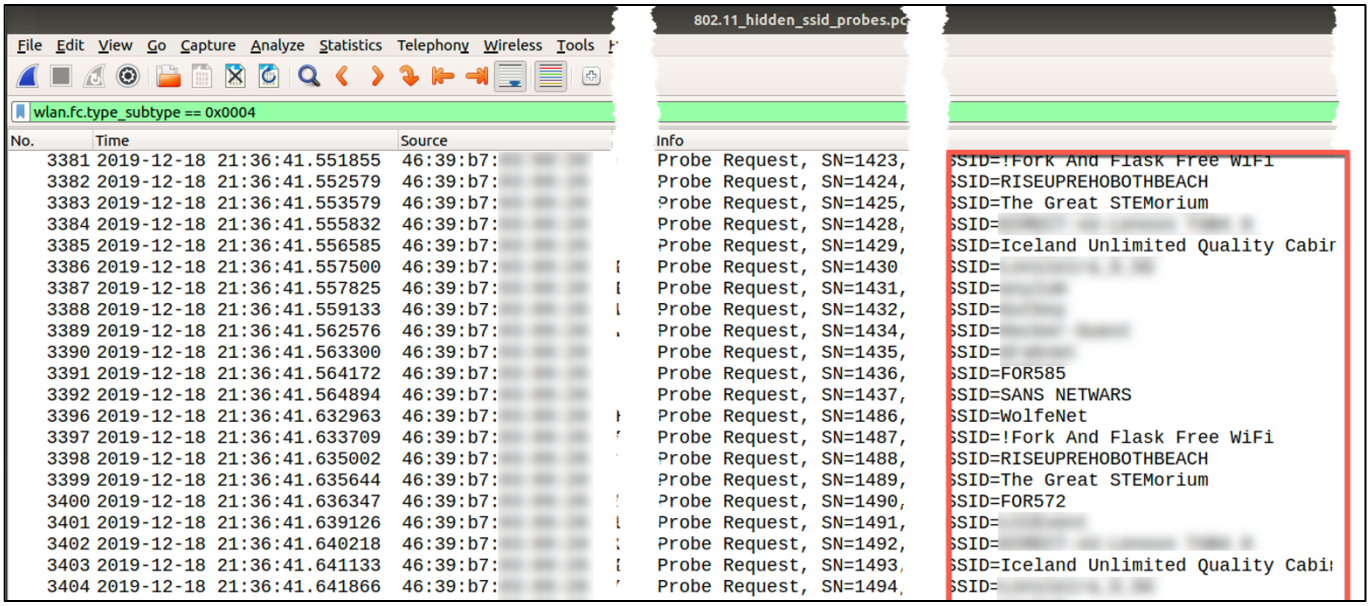
SANS DFIR FOR572.4 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response 61

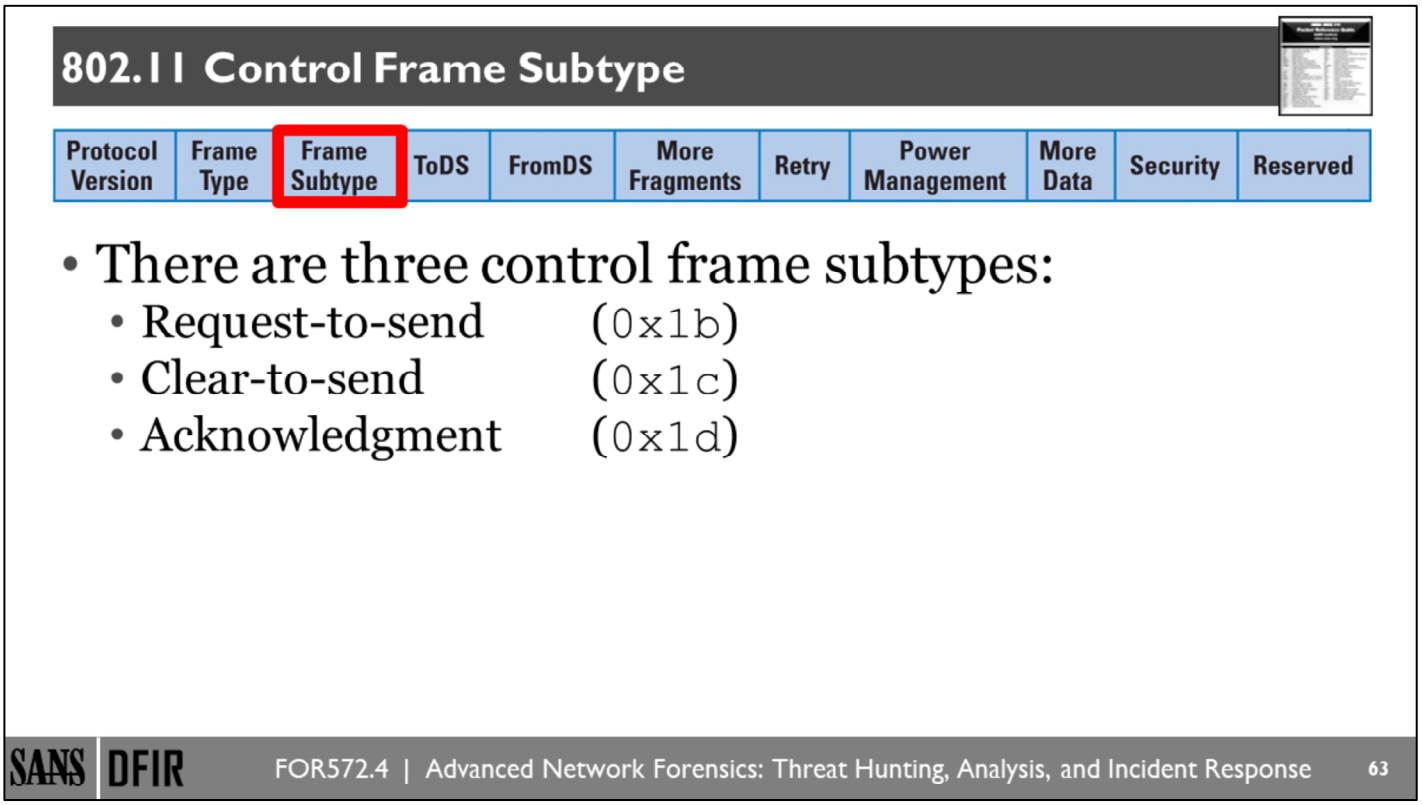
The Service Set Identifier (SSID) is a textual tag that identifies a wireless network to the users. It is transmitted in the AP's beacons, which are a type of management frame. Beacons are commonly transmitted between 10 and 20 times a second, making them one of the most common management frame subtypes in 802.11 traffic.

In the past, it was suggested that restricting beacon frames from being broadcast would increase the security of a network by making it a “Hidden Network” or “Cloaked Network.” However, this is a technical fallacy because the SSID is present—in plaintext format—in other management frames, including those sent during the wireless network association process. If an attacker wanted to learn the SSID of a targeted network, he could simply spoof a deauthentication packet to a station, and then monitor the reassociation process, which includes the SSID.

The SSID, along with numerous other values in the 802.11 frame header, is contained in a field of “tagged values”. A list of common tags is on your handout and shown above. Each tag consists of a single byte that tells what field follows, then a length value, and finally the tag itself. It is important to recognize that these tags often include critical information to the normal operation of the wireless network, including how it is protected, what authentication methods are allowed, etc.







As we identified earlier, the RF spectrum is a hub/broadcast medium and, as a result, there are transmission collisions.

With wired Ethernet, all stations access the same medium. Therefore, the standard "Carrier Sense, Multiple Access with Collision Detection" (CSMA/CD) is used. Because stations in this model can transmit and receive data at the same time, they can detect if they attempt to transmit at the same time another station is doing so, and delay a minuscule amount of time to avoid crosstalk.

However, in a wireless environment, clients are not always in range of each other and therefore cannot determine if another station in the BSS is utilizing the shared medium of the RF spectrum. This is referred to as the "hidden node" problem.[1] Further, once a station starts transmitting a data stream, it can no longer receive traffic on that frequency—including any control frames that would indicate another station needs to send traffic. Therefore, in the wireless realm, Carrier Sense, Multiple Access with Collision Avoidance (CSMA/CA) is used.

In CSMA/CA, control frames are used with specific subtypes to manage access to the frequency, which prevents rather than detects collisions. There are three Control Frame subtypes: "Request-to-send", "Clear-to-send", and "Acknowledgment". These frames are used by the in-RF-range stations and access points to manage the use of the RF spectrum.

These frames are also unencrypted as well as unique among 802.11 participants in that these frames are "per channel" and not tied to any specific SSID or BSSID. Rather, all stations and access points on a given frequency will exchange these messages to share the medium they all use.

References:

[1] https://for572.com/-wva2

802.11 Data Frames

Protocol Version	Frame Type	Frame Subtype	ToDS	FromDS	More Fragments	Retry	Power Management	More Data	Security	Reserved

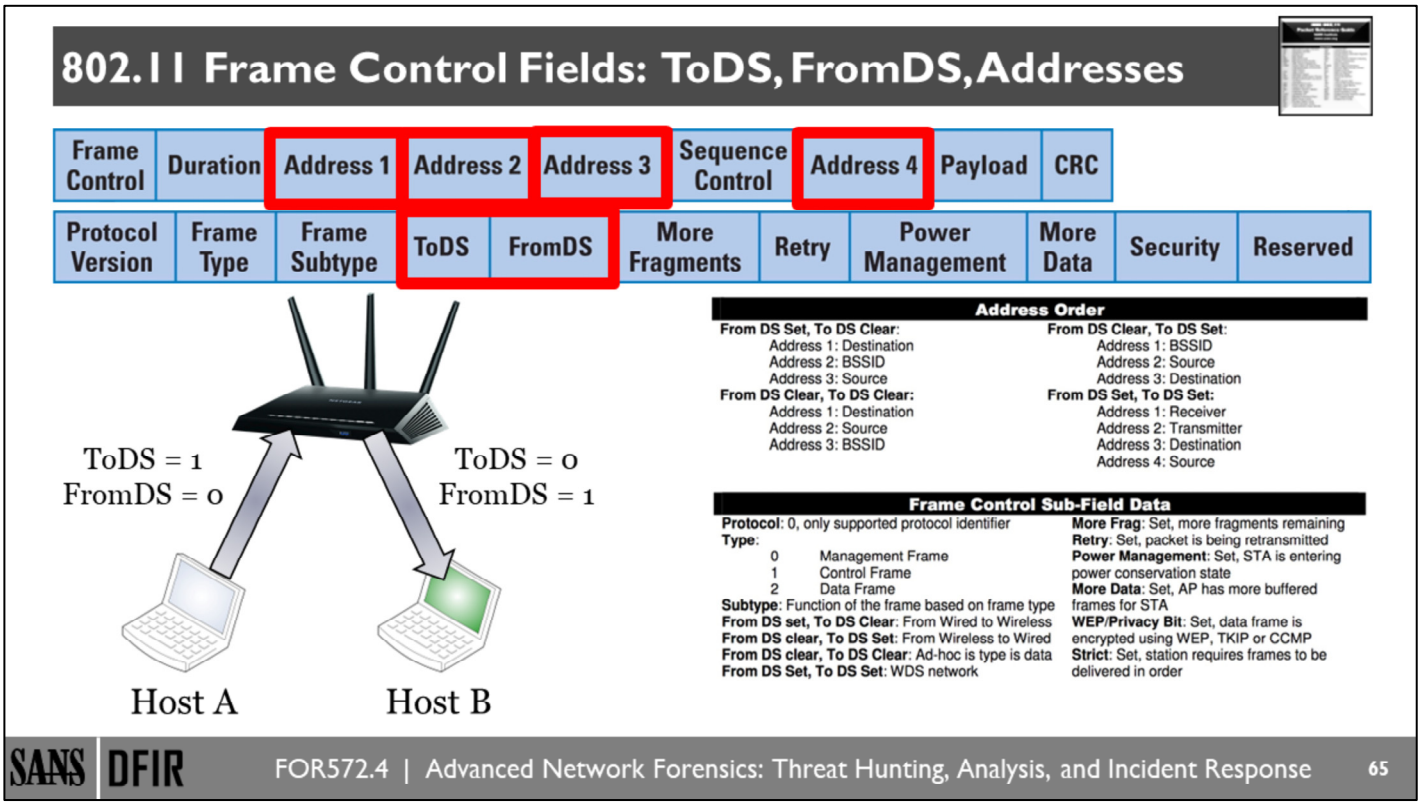
- Many different data frame subtypes, including the Null function (`0x24`), indicating no data
- Most interesting is likely to be those where the "Logical-Link Control Type" is IP (`0x0800`)
- Only frame type that can be encrypted, indicated by the single "Security" bit
  - No differentiation between WEP/WPA/WPA2/etc.
  - Old tools assume all encrypted traffic is WEP

Data frames carry the upper layer protocols (TCP/IP) within their payload section of the frame; thus, 802.11 data frames are the Layer 2 protocol within which IP is encapsulated. This is where most network forensic processes will focus, as we are typically looking at data payloads.

Data frames are the only 802.11 frame type that can be encrypted, but this is indicated in the frame header by just one single bit. Therefore, it is not possible to further differentiate WEP from WPA from WPA2 or any other encryption parameters. This differentiation is made using the tagged parameters discussed previously.

Old tools (such as the venerable but woefully outdated NetStumbler) assume that all frames with the security bit set are WEP because when they were written, WEP was the only security that was available at the 802.11 frame level. These tools should be replaced with something newer.

References:
https://for572.com/-95md (section 7.2.2)



The ToDS and FromDS bit flags allow the direction of the frame to be identified. When a frame is being sent from one station to another, it is actually ”reflected” by the access point, which also rewrites the header, effectively reframing the transmission. The directionality bits will indicate the leg of the transmission the frame was on at the time it was captured. This also means that typically, a passive capture including station-to-station traffic will have duplicated content.

Here, Host A sends a packet, say an ICMP Echo Request, to Host B. However, the stations use a BSS and as such, they are only communicating with the access point, not directly with each other. Therefore, the ICMP message is transmitted to the AP (not directly Host B) with the ToDS bit-flag set and the FromDS bit-flag clear.

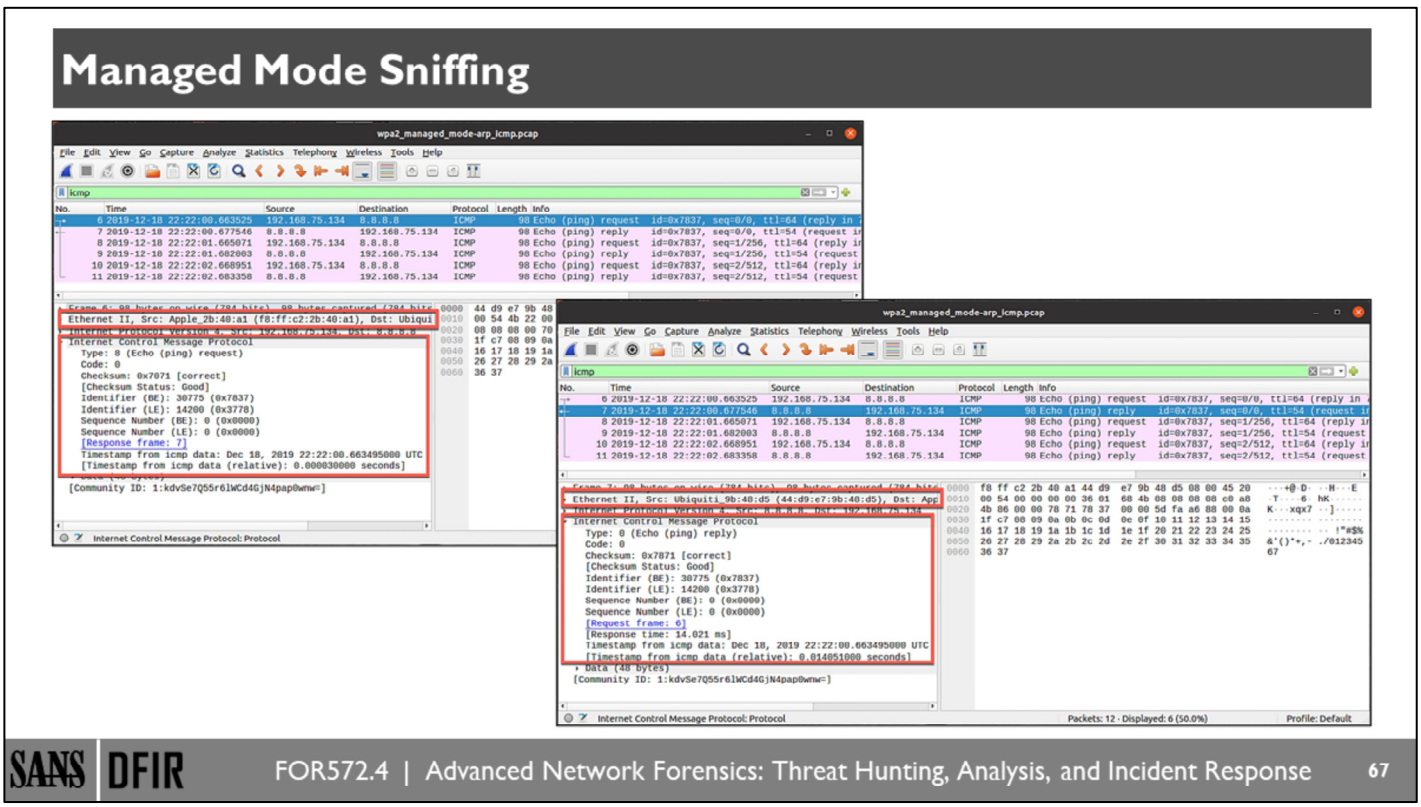
When the access point retransmits, the packet containing the ICMP Echo Request to the Host B, the FromDS bit-flag will be set, and the ToDS bit-flag will now be cleared.

If sniffing this in Managed mode, the ICMP Echo request will be observed normally, and from the pcap, the user may not even notice that Host B is a wireless client. However, if seen by a system in Monitor mode, both legs will be seen, appearing to be a duplicated packet unless carefully inspected to identify the directionality of each within the BSS.

The reframing process also rewrites the MAC addresses in the 802.11 frame header. Just like on wired networks, all adapters are required to have unique 6-byte hexadecimal Media Access Control (MAC) addresses. These values include both the BSSID and the MAC address of the station(s), depending on the leg on which the frame is currently being sent.

In the 802.11 header, there are always at least three valid address fields, and four in a WDS environment when the frames are forwarded between access points. The content of these fields is context-dependent. It is the ToDS and FromDS bits that define the context and thus the values that are placed in the fields. Note that in a non-WDS scenario, the field still exists in the frame header itself, but the contents are indeterminate.

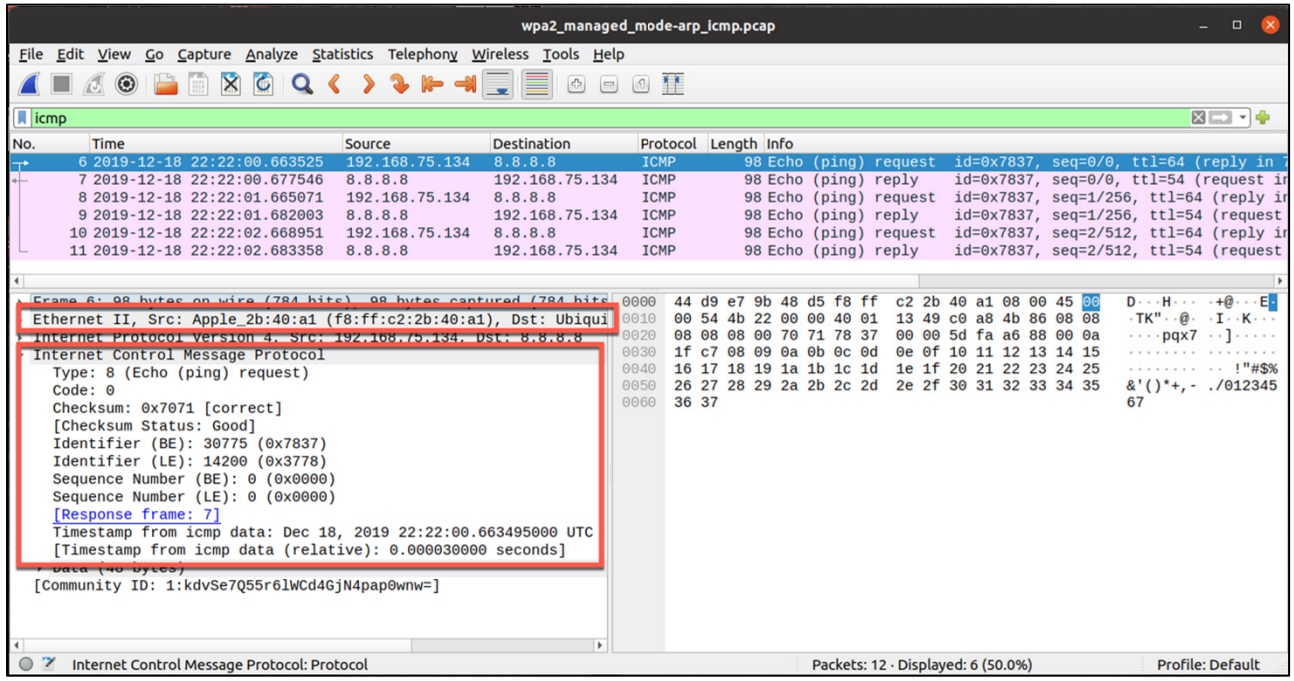
Although the pocket reference is concise, it does not mention that the same addresses can be in multiple fields within the same frame. If the access point is sending a frame to a Wi-Fi host (say the admin), the MAC address of the AP (also the BSSID) will be in Address location 2 and 3; similarly, the reply will go from the Wi-Fi host to the AP and the AP's MAC will be in field 1 and 3. This can cause confusion for new analysts as they are not used to seeing a MAC address in two places in a raw networking capture.

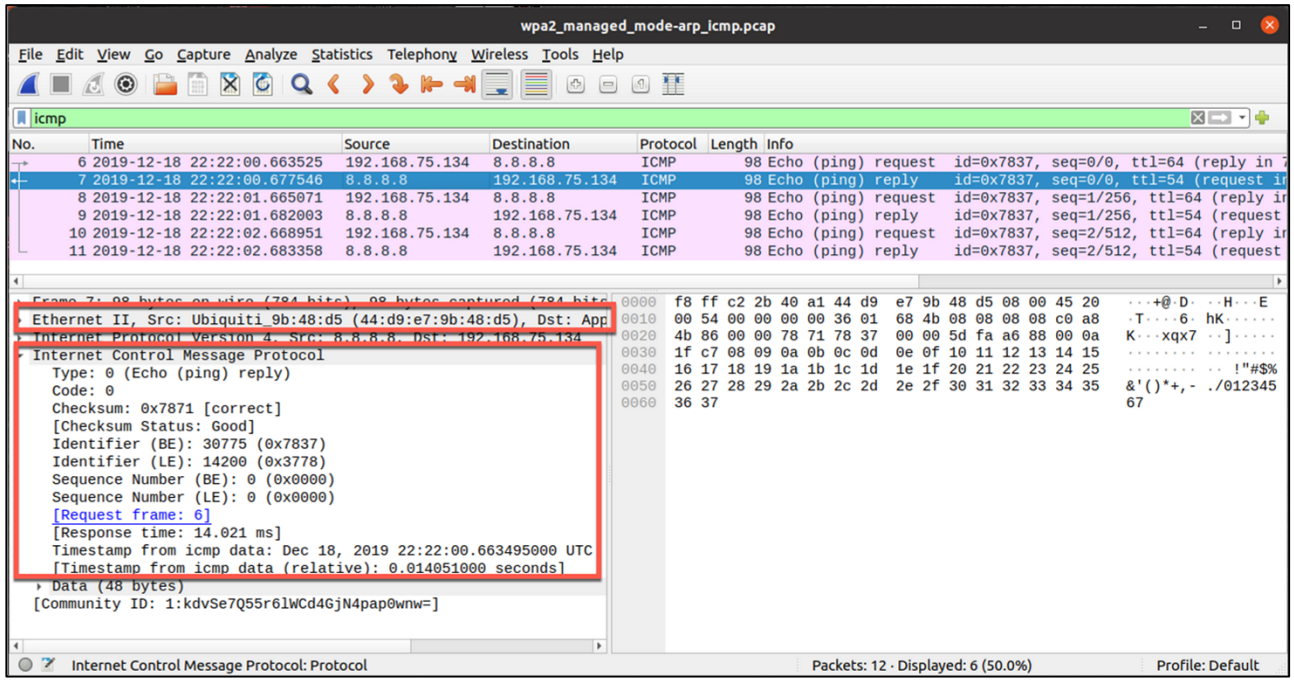


In Managed mode, the user's view of the captured traffic is akin to that of traditional wired sniffing, with none of the 802.11 underlying information being visible.

For this to be conducted, the user must be connected to an AP (including having the necessary WPA2 PSK or other credentials). As the user is connected to the AP, they will see the data free of any 802.11-based encryption.

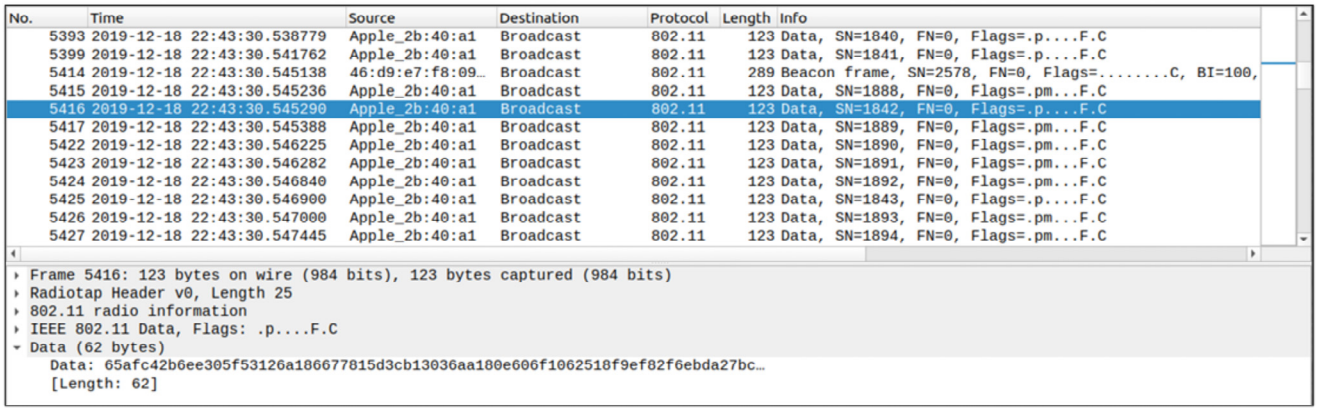
This is because the encryption functionality is handled by the wireless network card driver. The `tcpdump` process accesses the network data above that level, where the traffic is still abstracted as Ethernet. All of the 802.11 features and headers are added outside the capture process's level of visibility. Similarly, such a capture will not include the 802.11-specific management and control frames.





RFMON Sniffing – WPA2

- WPA2-protected data frames effectively opaque when captured in monitor mode
- Later acquisition of key material allows decryption



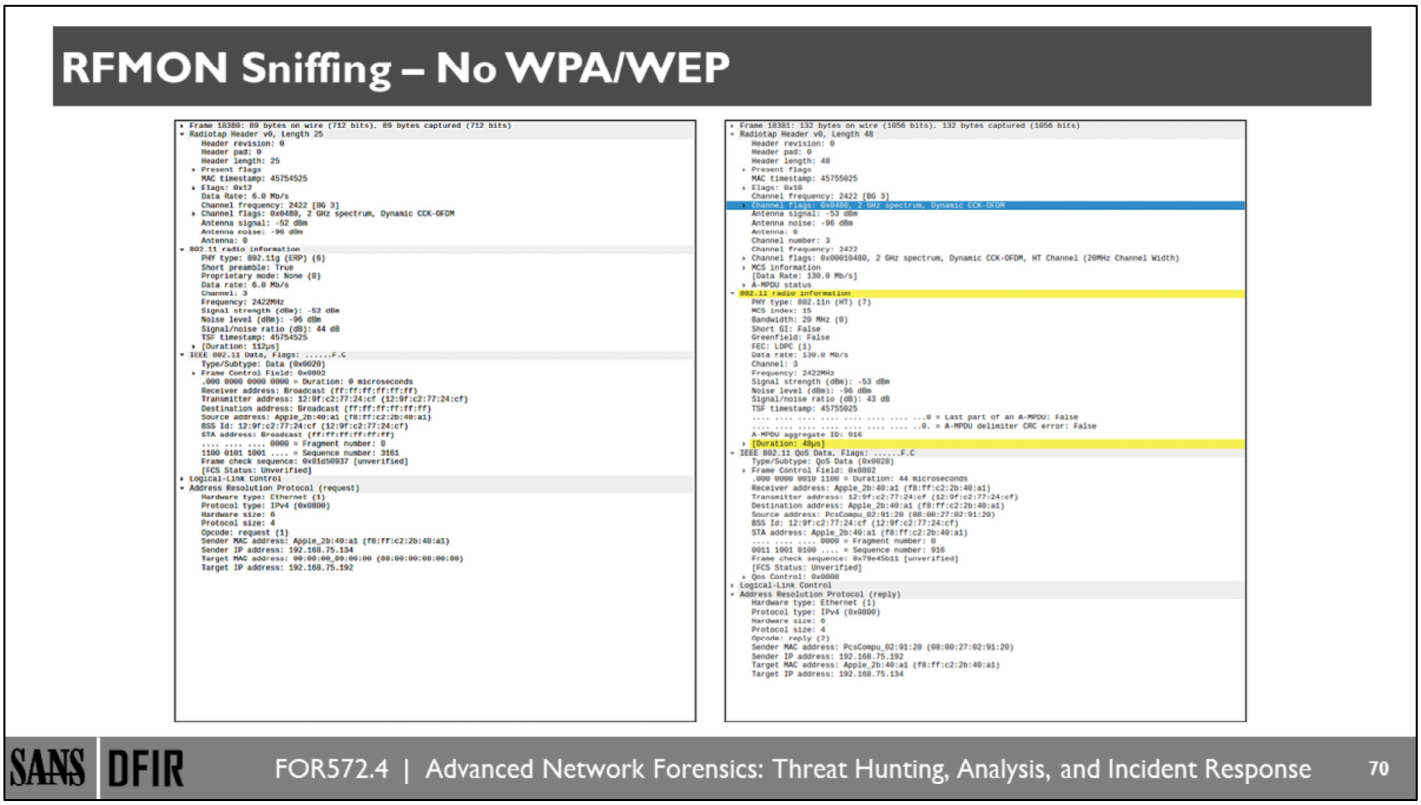
If the user is not connected to the AP, then the only sniffing that can be conducted is using RFMON and locking the radio in the Wi-Fi card to the frequency or channel of the targeted BSSID and SSID. However, without the Pre-Shared Key (PSK), any encrypted data frames will not be readable.

This screenshot shows that Wireshark reflects data frames (created by ARP scanning the wireless subnet) as arbitrary "Data" because the content is not accessible without the encryption key material. Although the attacker can sniff this data, no further decoding is possible without the PSK. The only information that can be used would be the Layer 2 addresses, timing, and volume of the traffic. (In this case, the screenshot was generated after ARP scan activity, which can loosely be inferred based on the rapid timing and broadcast destination.)

However, if the attacker retains this pcap file and later obtains the wireless network's authentication material (PSK or PairWise Master Key (PMK)[1] for the session), they may be able to decrypt the traffic. The encryption material could be gained through brute-forcing, social engineering, or other means.

References:

[1] The PMK is what is generated during the 4-way handshake between an AP and a station. From the PMK, the session keys are derived. Therefore, knowledge of the PMK (by knowing the PSK and seeing the 4-way handshake) will render that session's encryption compromised.



Here the sniffing was again with RFMON. However, the network is not encrypted at the 802.11 level (no WEP/WPA/WPA2).

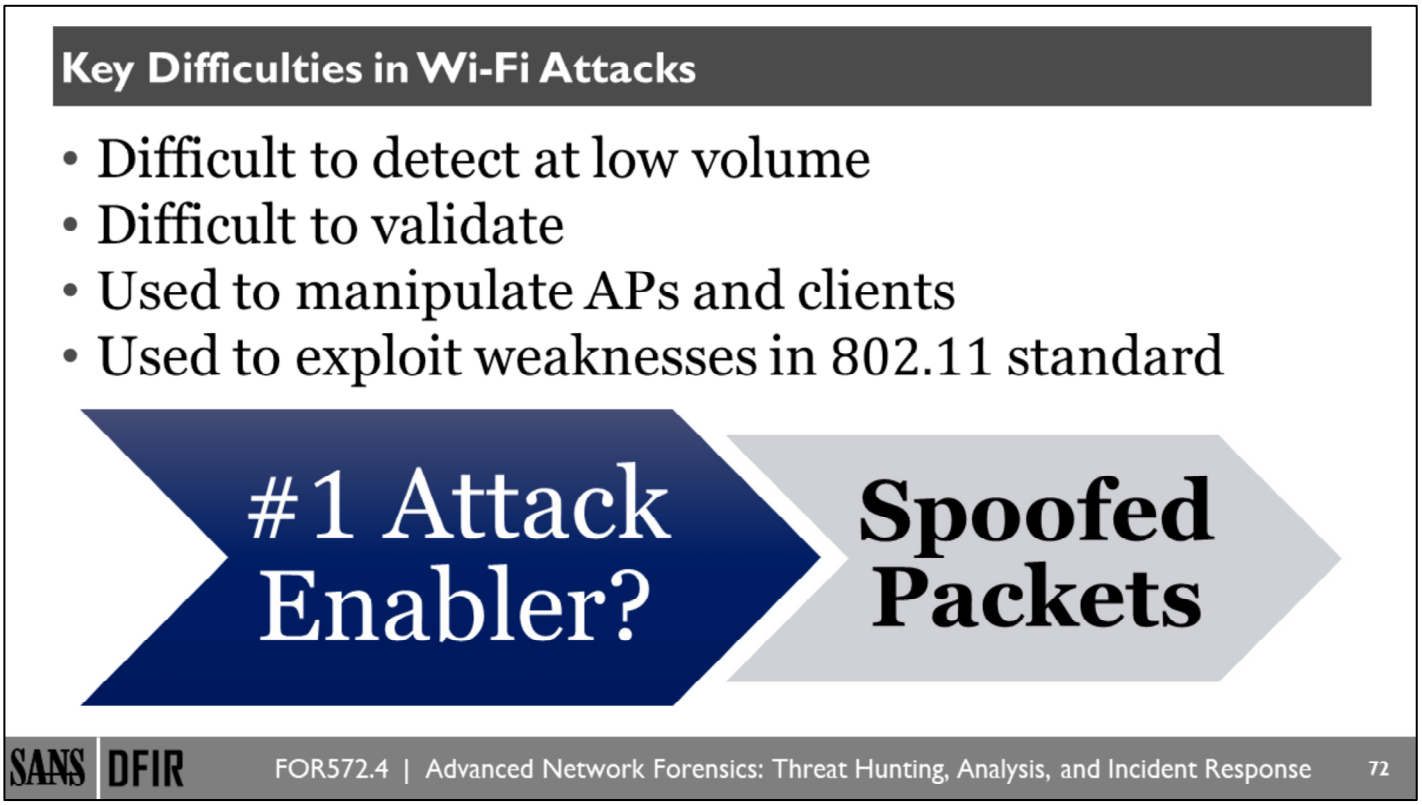
This provides the same endpoint, timing, and volumetric analysis, as well as content examination. Wireshark's encapsulation view model will provide visibility to both the framing and the content, with various field names at Layer 2 and up available for use.

No.	Time	Source	Destination	Protocol	Length	Info
5393	2019-12-18 22:43:30.538779	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1840, FN=0, Flags=.p....F.C
5399	2019-12-18 22:43:30.541762	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1841, FN=0, Flags=.p....F.C
5414	2019-12-18 22:43:30.545138	46:d9:e7:f8:09…	Broadcast	802.11	289	Beacon frame, SN=2578, FN=0, Flags=........C, BI=100,
5415	2019-12-18 22:43:30.545236	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1888, FN=0, Flags=.pm...F.C
5416	2019-12-18 22:43:30.545290	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1842, FN=0, Flags=.p....F.C
5417	2019-12-18 22:43:30.545388	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1889, FN=0, Flags=.pm...F.C
5422	2019-12-18 22:43:30.546225	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1890, FN=0, Flags=.pm...F.C
5423	2019-12-18 22:43:30.546282	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1891, FN=0, Flags=.pm...F.C
5424	2019-12-18 22:43:30.546840	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1892, FN=0, Flags=.pm...F.C
5425	2019-12-18 22:43:30.546900	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1843, FN=0, Flags=.p....F.C
5426	2019-12-18 22:43:30.547000	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1893, FN=0, Flags=.pm...F.C
5427	2019-12-18 22:43:30.547445	Apple_2b:40:a1	Broadcast	802.11	123	Data, SN=1894, FN=0, Flags=.pm...F.C

```
▸ Frame 5416: 123 bytes on wire (984 bits), 123 bytes captured (984 bits)
▸ Radiotap Header v0, Length 25
▸ 802.11 radio information
▸ IEEE 802.11 Data, Flags: .p....F.C
▾ Data (62 bytes)
    Data: 65afc42b6ee305f53126a186677815d3cb13036aa180e606f1062518f9ef82f6ebda27bc…
    [Length: 62]
```

```
▸ Frame 18380: 89 bytes on wire (712 bits), 89 bytes captured (712 bits)
▾ Radiotap Header v0, Length 25
    Header revision: 0
    Header pad: 0
    Header length: 25
  ▸ Present flags
    MAC timestamp: 45754525
  ▸ Flags: 0x12
    Data Rate: 6.0 Mb/s
    Channel frequency: 2422 [BG 3]
  ▸ Channel flags: 0x0480, 2 GHz spectrum, Dynamic CCK-OFDM
    Antenna signal: -52 dBm
    Antenna noise: -96 dBm
    Antenna: 0
▾ 802.11 radio information
    PHY type: 802.11g (ERP) (6)
    Short preamble: True
    Proprietary mode: None (0)
    Data rate: 6.0 Mb/s
    Channel: 3
    Frequency: 2422MHz
    Signal strength (dBm): -52 dBm
    Noise level (dBm): -96 dBm
    Signal/noise ratio (dB): 44 dB
    TSF timestamp: 45754525
  ▸ [Duration: 112µs]
▾ IEEE 802.11 Data, Flags: ......F.C
    Type/Subtype: Data (0x0020)
  ▸ Frame Control Field: 0x0802
    .000 0000 0000 0000 = Duration: 0 microseconds
    Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
    Transmitter address: 12:9f:c2:77:24:cf (12:9f:c2:77:24:cf)
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    Source address: Apple_2b:40:a1 (f8:ff:c2:2b:40:a1)
    BSS Id: 12:9f:c2:77:24:cf (12:9f:c2:77:24:cf)
    STA address: Broadcast (ff:ff:ff:ff:ff:ff)
    .... .... .... 0000 = Fragment number: 0
    1100 0101 1001 .... = Sequence number: 3161
    Frame check sequence: 0x01d50937 [unverified]
    [FCS Status: Unverified]
▸ Logical-Link Control
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Apple_2b:40:a1 (f8:ff:c2:2b:40:a1)
    Sender IP address: 192.168.75.134
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.75.192
```

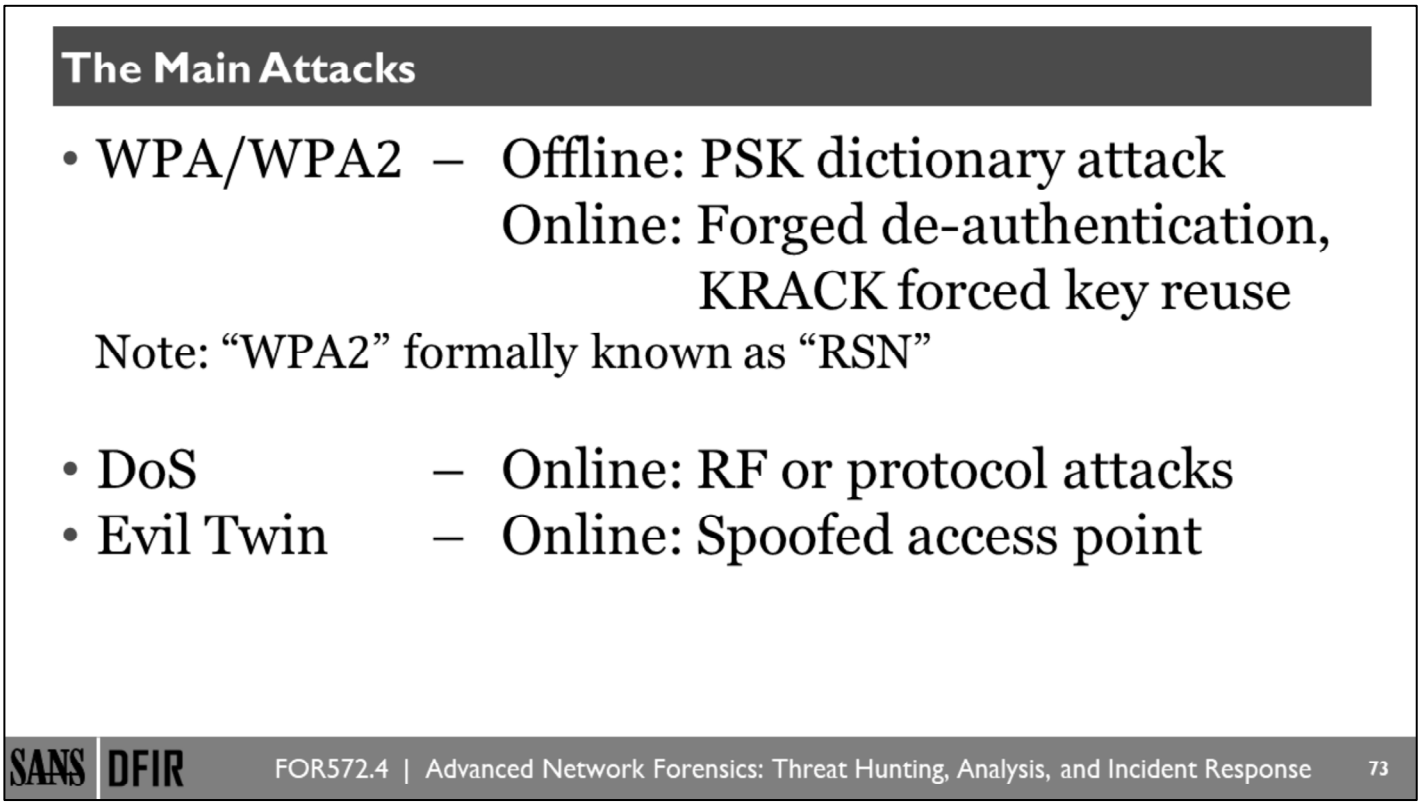
```
▸ Frame 18381: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits)
▾ Radiotap Header v0, Length 48
    Header revision: 0
    Header pad: 0
    Header length: 48
  ▸ Present flags
    MAC timestamp: 45755025
  ▸ Flags: 0x10
    Channel frequency: 2422 [BG 3]
  ▸ Channel flags: 0x0480, 2 GHz spectrum, Dynamic CCK-OFDM
    Antenna signal: -53 dBm
    Antenna noise: -96 dBm
    Antenna: 0
    Channel number: 3
    Channel frequency: 2422
  ▸ Channel flags: 0x00010480, 2 GHz spectrum, Dynamic CCK-OFDM, HT Channel (20MHz Channel Width)
  ▸ MCS information
    [Data Rate: 130.0 Mb/s]
  ▸ A-MPDU status
▾ 802.11 radio information
    PHY type: 802.11n (HT) (7)
    MCS index: 15
    Bandwidth: 20 MHz (0)
    Short GI: False
    Greenfield: False
    FEC: LDPC (1)
    Data rate: 130.0 Mb/s
    Channel: 3
    Frequency: 2422MHz
    Signal strength (dBm): -53 dBm
    Noise level (dBm): -96 dBm
    Signal/noise ratio (dB): 43 dB
    TSF timestamp: 45755025
    .... .... .... .... .... .... .... ...0 = Last part of an A-MPDU: False
    .... .... .... .... .... .... .... ..0. = A-MPDU delimiter CRC error: False
    A-MPDU aggregate ID: 916
  ▸ [Duration: 48µs]
▾ IEEE 802.11 QoS Data, Flags: ......F.C
    Type/Subtype: QoS Data (0x0028)
  ▸ Frame Control Field: 0x8802
    .000 0000 0010 1100 = Duration: 44 microseconds
    Receiver address: Apple_2b:40:a1 (f8:ff:c2:2b:40:a1)
    Transmitter address: 12:9f:c2:77:24:cf (12:9f:c2:77:24:cf)
    Destination address: Apple_2b:40:a1 (f8:ff:c2:2b:40:a1)
    Source address: PcsCompu_02:91:20 (08:00:27:02:91:20)
    BSS Id: 12:9f:c2:77:24:cf (12:9f:c2:77:24:cf)
    STA address: Apple_2b:40:a1 (f8:ff:c2:2b:40:a1)
    .... .... .... 0000 = Fragment number: 0
    0011 1001 0100 .... = Sequence number: 916
    Frame check sequence: 0x79e45b11 [unverified]
    [FCS Status: Unverified]
  ▸ Qos Control: 0x0000
▸ Logical-Link Control
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: PcsCompu_02:91:20 (08:00:27:02:91:20)
    Sender IP address: 192.168.75.192
    Target MAC address: Apple_2b:40:a1 (f8:ff:c2:2b:40:a1)
    Target IP address: 192.168.75.134
```



One significant problem with wireless networks is the fact that anyone can generate packets and place them in the medium. Additionally, an attacker doesn't need to be in the building/campus or on the organization's property—they can attack from the coffee shop across the street.

Therefore, data can easily be injected to manipulate both access points and client stations to behave in ways that benefit attackers by causing them to send sensitive/useful data back into the wireless environment, or by exploiting any weaknesses in the 802.11 standards themselves. This is compounded by the fact that 802.11 management frames are necessarily unencrypted, making it quite trivial for someone intent on causing harm to forge these packets.

With only basic wireless equipment, the analyst will struggle to identify injected packets and frames. Identification is possible, but without tools like Wireless IDS (WIDS), the manual checking for injected data does not scale much past the classroom.

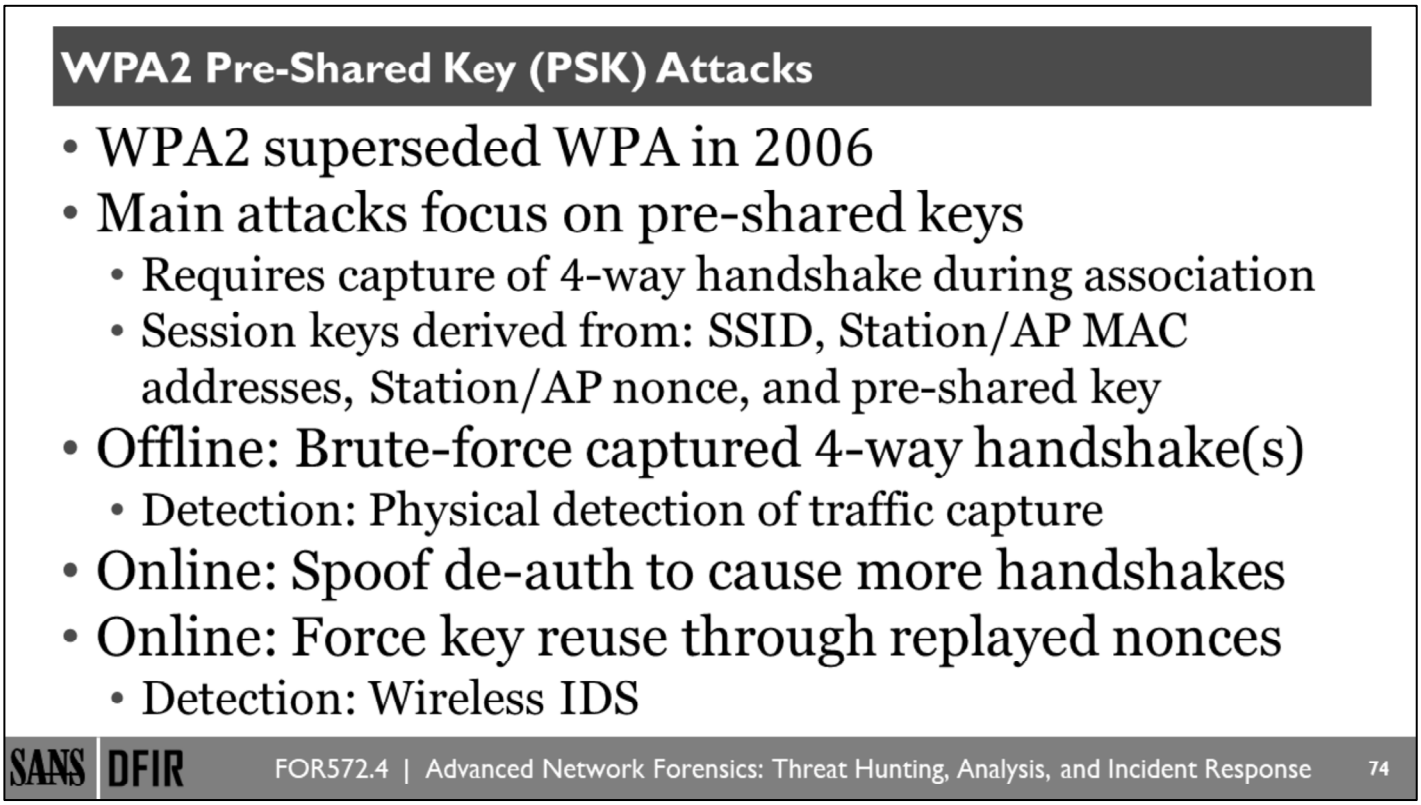


We'll briefly consider some of the types of attacks against networks protected by typical measures. In any case, consider that these attacks can be either online or offline. An "online" attack is one in which the attacker interacts with the environment, whereas an "offline" attack requires only RFMON collection.

- Wireless Protected Access (WPA/WPA2)
  - Offline attack: Brute-forcing Pre-Shared Key
  - Online attack: Force stations to leave the network, divulging authentication data upon automatic/immediate re-association, KRACK forced key reuse[1]
- Denial of Service (DoS)
  - Online attack: Can use inherent nature of RF to kill signal-to-noise ratio or attack protocol weaknesses
- Evil Twin
  - Rogue AP advertises known or expected SSIDs, exploiting stations' typical "automatic association to known SSID" behavior

One note to keep in mind while examining encrypted wireless traffic is that the standard commonly known as "WPA2" is actually the "Robust Security Network" or "RSN" standard. Wireshark uses the RSN designation in its parsing functions.

References:
[1] https://for572.com/xjt2n



The WPA2 cryptographic implementation is currently deemed mathematically secure, so attackers have focused on the values used to generate per-client session keys. These keys are part of the Pairwise Master Key (PMK) that is derived from various values that are transmitted in the clear.

This attack requires the interception of the following values:

- Network SSID
- MAC address of the station
- MAC address of the access point
- A NONCE transmitted by the station during the association process
- A NONCE transmitted by the access point during the association process

These are all transmitted when every client connects to the wireless network through what is called a 4-way handshake. The attacker lacks only the PSK (the encryption password) that is chosen by the admin establishing the network—the other five values are passed across the connection in the clear in management frames. The Aircrack Wiki has a detailed discussion of the entire authentication process including packet captures for successful and failed attempts.[1]

A completely passive attack (as accomplished by Josh Wright's **coWPAtty** tool[2]), requires a capture containing a 4-way handshake and a dictionary of words that might be the password.

However, an impatient attacker, or one attacking a relatively quiet environment, may not have time to wait for a "natural" 4-way handshake to occur. Instead, they could simply craft a packet that appears to come from the access point that directs a client station to de-associate itself from the network. The wireless driver on the client will seek to re-establish the connection as soon as possible for usability purposes. This reassociation process requires a new 4-way handshake—of course, the attacker will be listening for this exchange, capturing it for offline brute-forcing.

A more recent (and, arguably severe) attack against WPA2 implementations was publicized as the KRACK vulnerability in late 2017[3]. This affected all wireless access points and clients at the time, and worked by forcing a

re-authorization event to result in the same session key by reusing the nonce values from a prior session key negotiation. This is also an online attack, requiring the attacker to interact with the hardware from a place in the "nearby" physical environment.

Detecting a passive /WPA2 attack can be quite difficult—as with any passive attack. This would require spotting the attacker while collecting the traffic, or the so-called "shady character with a big antenna parked in the parking lot" method. Obviously, this is not ideal, but it is a harsh reality in the realm of wireless investigations.

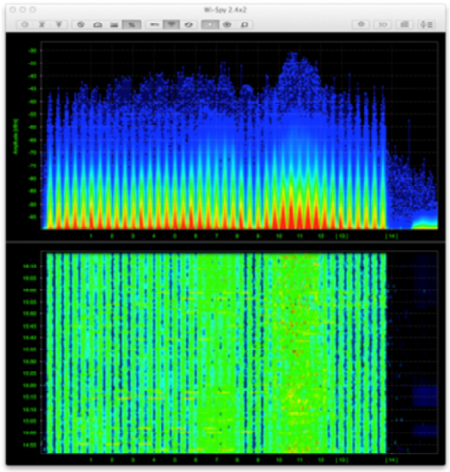
An online attack may be more easily detected with the use of a Wireless IDS (WIDS). These platforms can spot potentially spoofed traffic by keeping a baseline of typical behavior in the environment, and then alerting when traffic deviates from those baselines. In this case, knowing the normal rate of "deauthentication" packets would be a useful metric to track.

References:

[1] https://for572.com/ivujc
[2] https://for572.com/dr0b3
[3] https://for572.com/xjt2n

DoS Attack: RF Overload

- With a shared broadcast medium, DoS is easy
  - 2.4GHz and 5GHz spectrum bands can be polluted with a simple jammer
  - Unintentional DoS-by-overload



SANS DFIR FOR572.4 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response 76

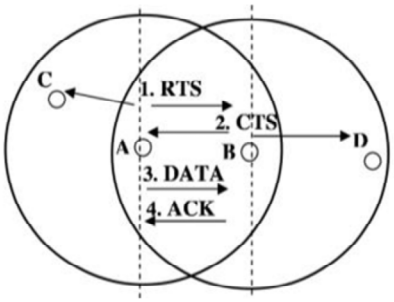
Aside from the 802.11 protocol itself, the fact that the RF medium is also vulnerable to DoS attacks should also be considered. By overloading the RF environment on a given frequency (say 2.4Ghz to 2.4835Ghz), all 802.11 traffic would effectively be blocked. IMPORTANT NOTE: Transmitting on a licensed frequency is illegal unless it is a frequency designated as "unlicensed" or you possess a license to transmit on that frequency. Conducting jamming activities is almost certainly illegal. SANS does not recommend, advise, or condone RF jamming or illegal transmission on licensed, or unlicensed frequencies. The SPEC5 device[1] shown jams 2.4GHz, 5GHz, and a variety of other frequencies. It is also almost certainly illegal to operate in most countries.

However, an RF DoS does not have to be intentional. The screenshot to the right of the slide was taken during a security audit of a video game authoring studio. The staff complained that it struggled to get traditional laptop wireless working in its building. Upon checking, it was discovered that the 300+ Microsoft Xbox consoles and 250+ Sony PlayStation consoles were generating sufficient 2.4Ghz RF to overload all 802.11 traffic operating in the same range.

References:
[1] https://for572.com/dlxu7

DoS Attack: 802.11 Management and Control Frames

- Spoofed deauthentication attack: Everybody off!
  - Uses forged management frames
  - Sends clients into de-auth/re-auth loop
  - Sent to client station: Forge AP-to-client de-auth
  - Sent to AP: Force AP-to-broadcast de-auth
- RTS/CTS control frame attack
  - If everyone is waiting for forged CTS to expire, no one can transmit



Deauthentication Attacks[1]
A deauthentication attack is conducted by an attacker spoofing deauthentication packets to control the behavior of the stations within their broadcast range. These use management frames, meaning they are unencrypted and unauthenticated. The attacker can target a single client station or the entire BSS of an access point. To target a single host, the spoofed packet is created with the BSSID as the source and the target as the destination. The wireless driver on the victim's system will interpret this packet directly, and dutifully remove itself from the network. Of course, the system still wants to communicate, so it needs to reassociate to the access point. This happens quickly, but if the attacker continues to flood the channel, the victim will never remain connected for any meaningful amount of time. To target the entire BSS, the attacker forges the same packet, but with a destination of the broadcast address—so all client stations in range will receive and process the deauthentication packet.

Deauthentication attacks can be devastating to the target and are surprisingly simple to implement by the attacker. Although the 802.11 standard does not provide a means to authenticate these packets, a WIDS would see the unusual spike in such packets and alert the administrator to the odd behavior.

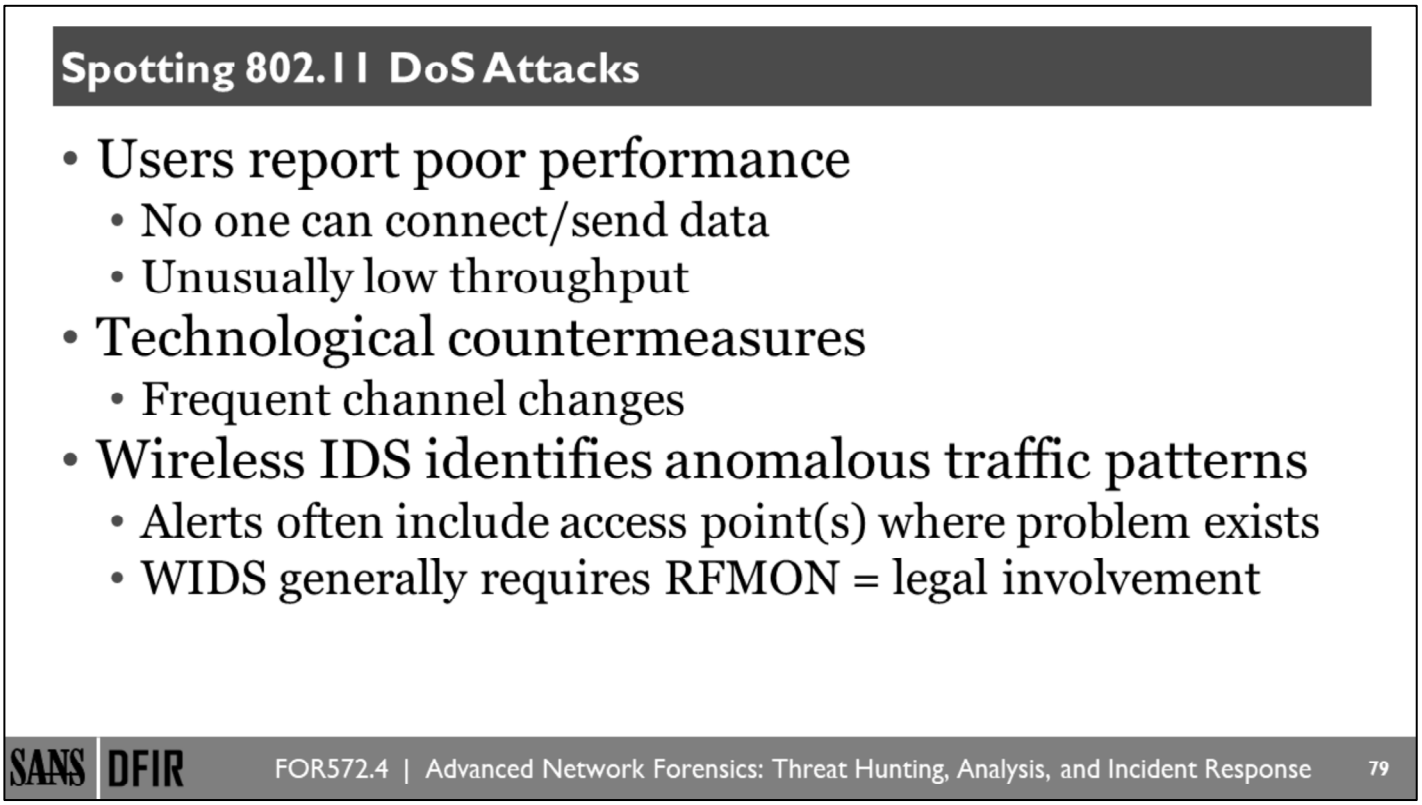
RTS/CTS Attacks
802.11 uses the airwaves for Layer 1, which are a shared medium much like that of a hub-built wired environment. However, one critical difference is that each station in the wireless environment may not have full visibility to all other hosts in the same environment. Therefore, at the 802.11 level, traffic collisions are avoided through the use of the CSMA/CA feature, as discussed previously.

When a client wishes to send traffic, it first sends a control frame called a "Request to Send" or "RTS". This includes a period of time for which the client station wishes to have exclusive access to the channel. When the client receives a "Clear to Send" (CTS) message, it can transmit for that duration of time. When any other client station sees a CTS destined for another party, it will go into a wait state and does not transmit. However, if those other stations also saw the original RTS, they can delay for the specified interval. If the original RTS was not seen, the other clients assume the original sender was of RF range and use an internally calculated random delay instead. This is absolutely necessary because client stations are frequently out of range without client stations in the same BSS, but it also opens client stations to a denial of service.

If an attacker forges CTS packets and sends them into the wireless environment, all clients who see the forged packets will assume they were not in range of the original RTS—an RTS that never existed in the first place. But the unknowing clients will enter the wait state until the CTS expires before attempting to communicate with a new RTS of their own. However, if the attacker floods the environment with these forged CTS packets, no amount of waiting would lead to the CTS expiring—the attacker would simply command the airwaves indefinitely. This CTS attack is explained in considerable depth in the paper "Smart Attacks based on Control Packets Vulnerabilities with IEEE 802.11 MAC."[2]

References:

[1] https://for572.com/2qnwz
[2] https://for572.com/ey46p



A DoS attack is usually easy to spot when it's successful. Generally, nobody has access to the network and user complaints are not too far behind. This often leads to a geographic area that is affected more than others, helping to localize the investigation.

However, most wireless platforms are built to mitigate such attacks by automatically changing channels when the selected one loses its quality or reliability. This change in channel is often logged, so if there is a rapid spike in channel changes or if it happens more frequently than expected, it may signify that the architecture is under attack but handling the situation so far. A proactive monitoring team should see these log entries and consider the possibility of a DoS attack.

From a protocol perspective, a quality WIDS platform will detect malformed traffic or a high volume of atypical traffic. This would likely include the access point(s) that are affected by the anomalies, which would also be a helpful detail for investigators because it would geolocate their response actions.

Evil Twin Attack

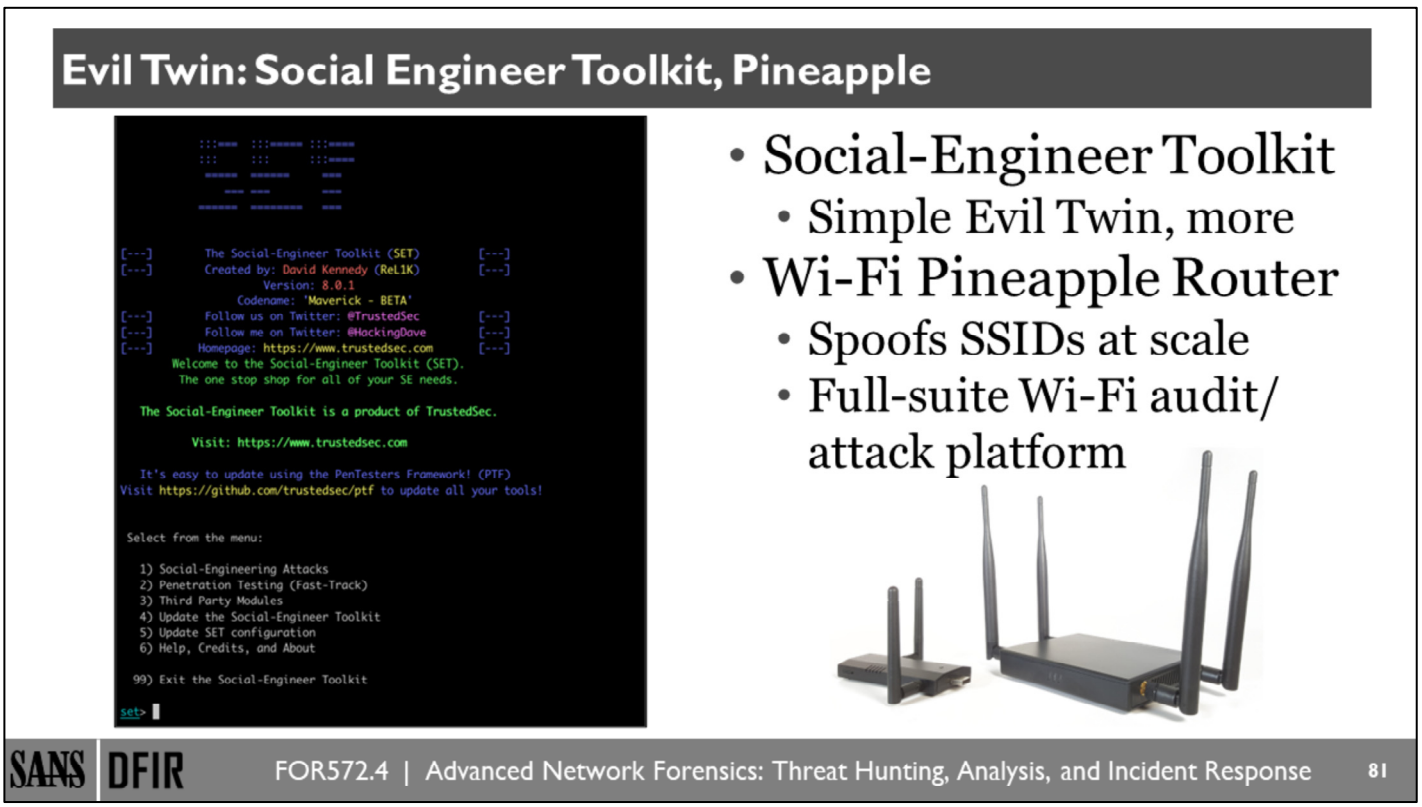
- Attacker hosts an access point with expected SSID
- Client stations attach to SSID regardless of BSSID
  - Highest priority on list at highest power usually wins
- Look for unexpected BSSIDs on known SSID
  - Also look for unusual spike in SSID count

In an Evil Twin attack, the victim is tricked into connecting to an SSID he believes to be safe and trusted. However, the victim does not realize that the BSSID providing the familiar network name is under the control of an attacker.

The attacker has used the expected SSID (and likely cloned any captive portal in use). To maximize the likelihood that a victim client station will associate to the Evil Twin and not the real BSSID, the attacker uses a card with a very strong signal. The attack can be conducted in or close to the organization's offices if powerful wireless cards and amplifying antennae are used. (For example, the Alfa adapter discussed earlier can be obtained in 2000mW versions, whereas a typical AP provides only 600–800mW. The legality of these amplified APs is dubious and varies greatly by country—you have been warned!)

When the victim connects to the attacker's Evil Twin BSSID, he uses the connection as normal—even providing credentials if requested. With those credentials, the attacker will be able to impersonate the user, intercept and manipulate traffic, and more. At this point, the attacker simply becomes the victim's upstream infrastructure.

Preventing Evil Twin access points is close to impossible because anyone can set up a device that acts like an access point, and the management frames are unencrypted and unauthenticated. Instead, incident responders should seek to keep a reliable inventory of trusted BSSIDs that provide service on SSIDs for which the organization is responsible. Whenever a new BSSID advertises that network, there is a chance a rogue access point has been activated. Another good metric to keep updated is the list of SSIDs typical for a given location. If a large number of new network names suddenly appears, this may be a special category of super-efficient Evil Twin attack, where a device dynamically impersonates dozens of SSIDs on demand.

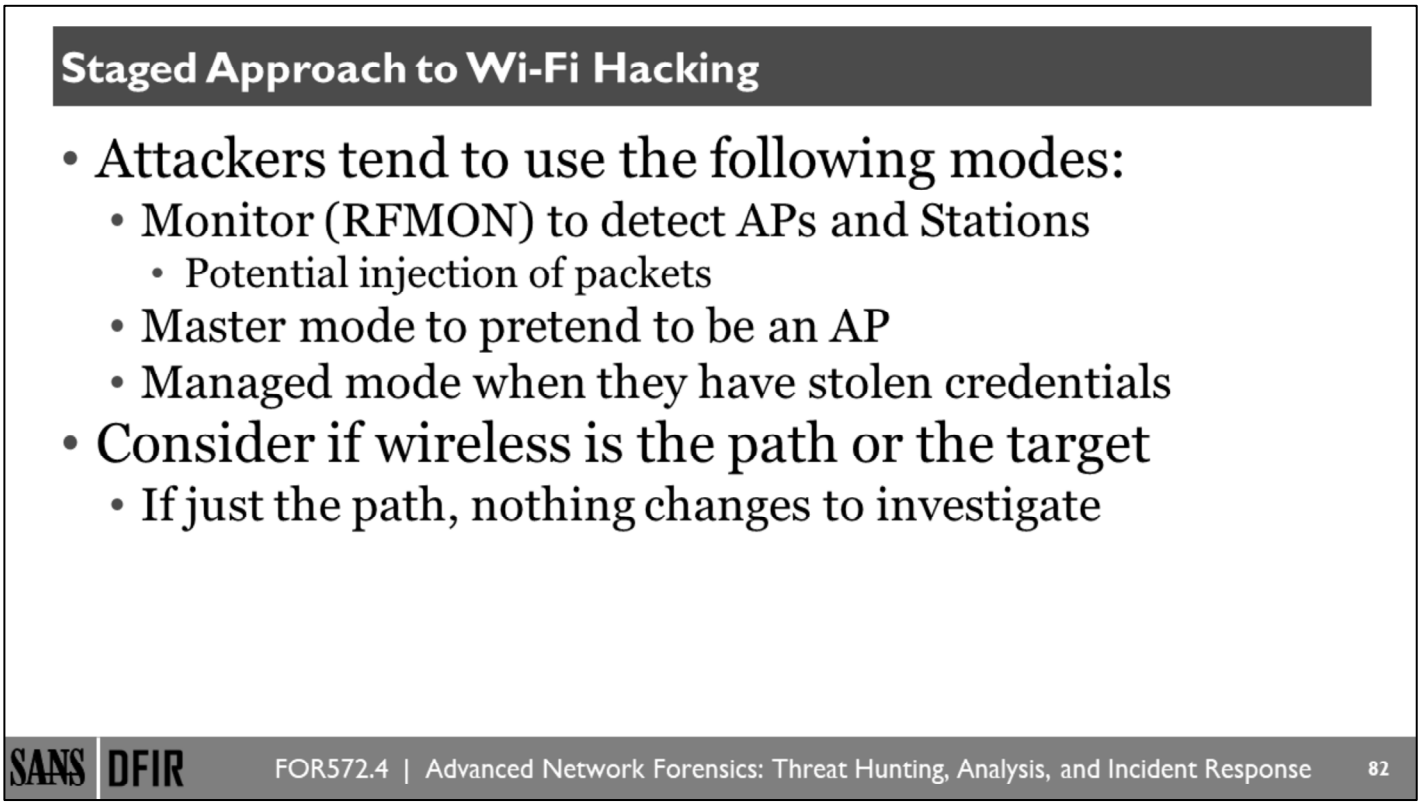


The Social-Engineer Toolkit [1] is a software tool that allows a malicious user or attacker to quickly set up a soft access point—a turnkey Evil Twin if using an existing SSID. Any victim systems that attach to the SET-based access point instead of the real infrastructure would then be subject to any traffic blocking, manipulation, or interception that the attacker wants to perform. As a robust toolkit intended for penetration testers, it provides added functionality such as automated credential theft, browser exploits, and more.

A hardware approach that can perform a similar function at scale is the Wi-Fi Pineapple Router[2] from Hak5. (Nano and Tetra models, shown above, respectively.) This hardware device listens for Wi-Fi probe requests, and then dynamically creates wireless networks with matching SSIDs. Victim systems then attach to these ”trusted” network names, allowing the attacker to perform the same Evil Twin functions against a wider base of victims. The Pineapple builds also provide extensive audit (therefore also attack) features.

Whether an ordinary computer running specialized software or dedicated hardware designed to support penetration testers, the miniaturization of hardware and growing availability of platforms that can provide a malicious access point means detecting rogue actors in the wireless environment will become an increasingly challenging aspect for incident responders for quite a long time.

References:
[1] https://for572.com/n83-4
[2] https://for572.com/sxg5u



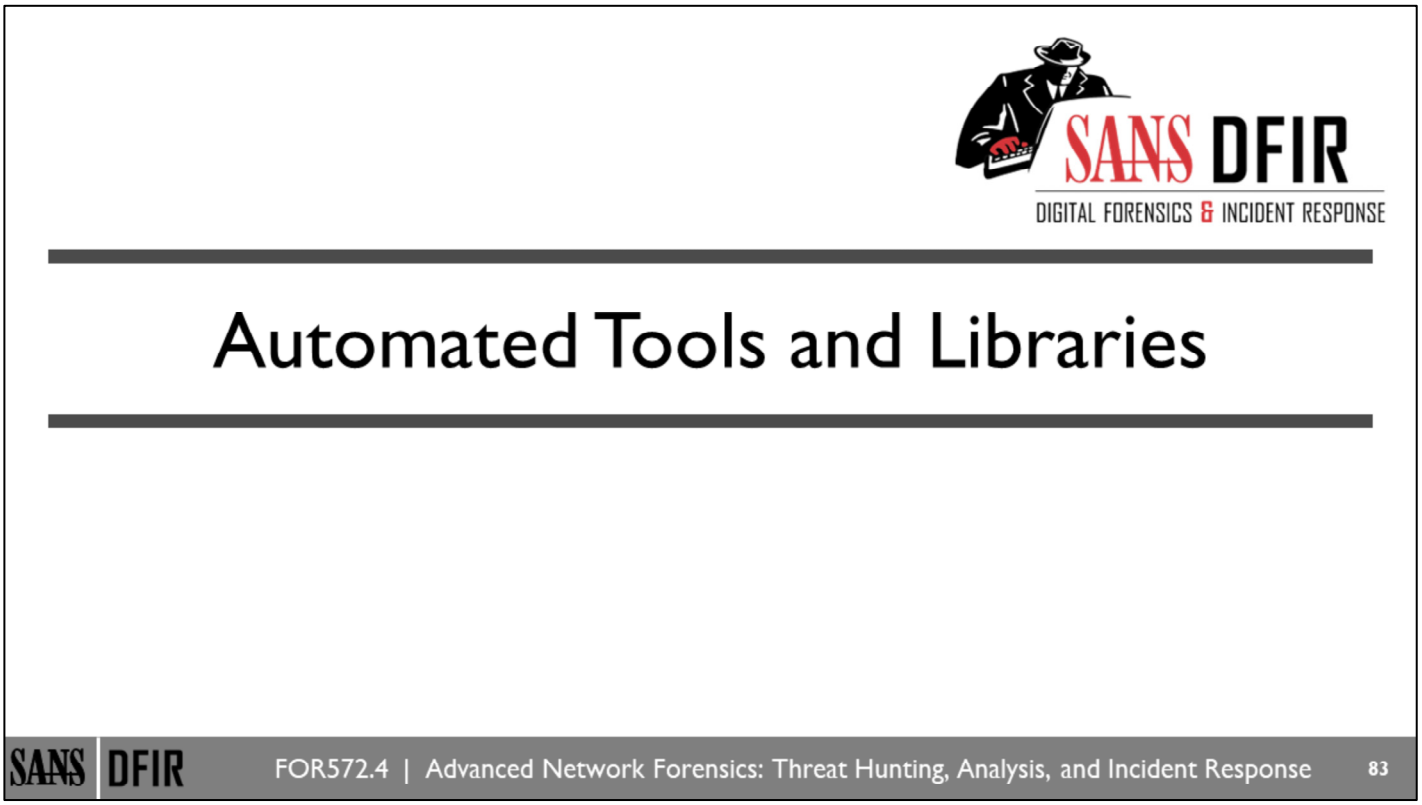
Although there is no single attack pattern in any environment—wired or wireless—the unique attack surface that a wireless network provides often means attackers tend to follow a known pattern of behavior.

First, the attacker will use monitor mode to gain intelligence about the victim's environment—SSIDs, protection mechanisms, volume of client activity, etc. This is undetectable and may occur over days or weeks, ensuring the attacker has enough data collected to continue the attack.

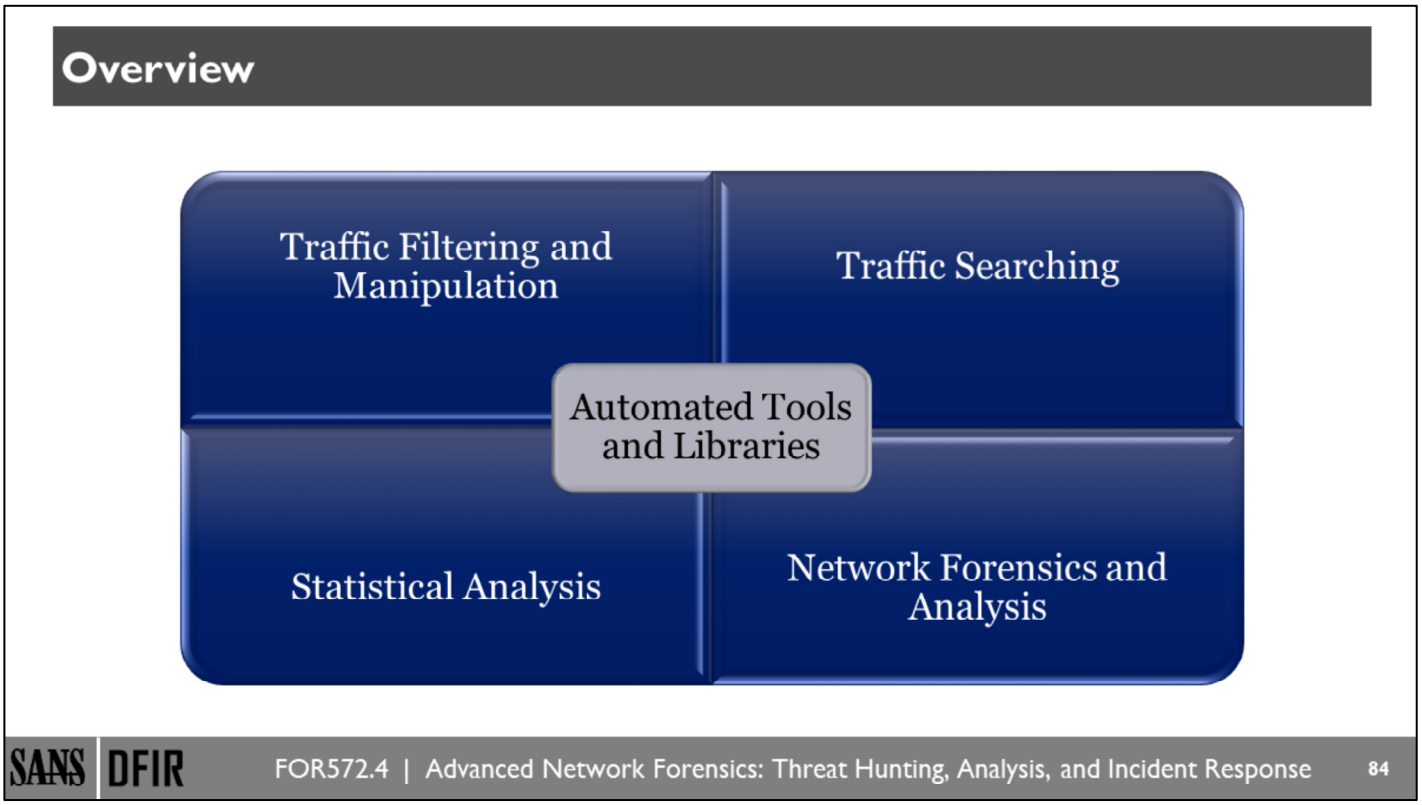
If the attacker sees the victim's wireless network is protected by WPA or WPA2, they can conduct offline attacks against the data collected previously, or prod network members to reassociate so they divulge the 4-way handshake, giving the attacker sufficient source material for an offline attack. In either case, the attacker seeks to derive the PSK used to associate with the network.

Depending on their goals, this may be when the attacker simply uses the PSK to access the victim's network and start conducting an attack from the inside. However, if they wish to conduct further reconnaissance or damage to the victim's users, they may instead take the Evil Twin route, setting up a rogue access point and intercepting the traffic of the unwitting users that attach to the Evil Twin instead of the legitimate infrastructure. Of course, at this point, the attacker has virtually limitless options to wreak havoc at their disposal.

The means to investigate this sequence of events center largely around log data and historic norms of the victim organization's traffic patterns. However, this notional sequence of events centers around the idea of a wireless attack, which should not be confused with an attack that simply uses wireless as its delivery mechanism. In the latter case, our methods do not significantly change from those used in any other network investigation. It is only when the attack leverages inherent weaknesses in wireless protocols that our investigative methodologies significantly change.

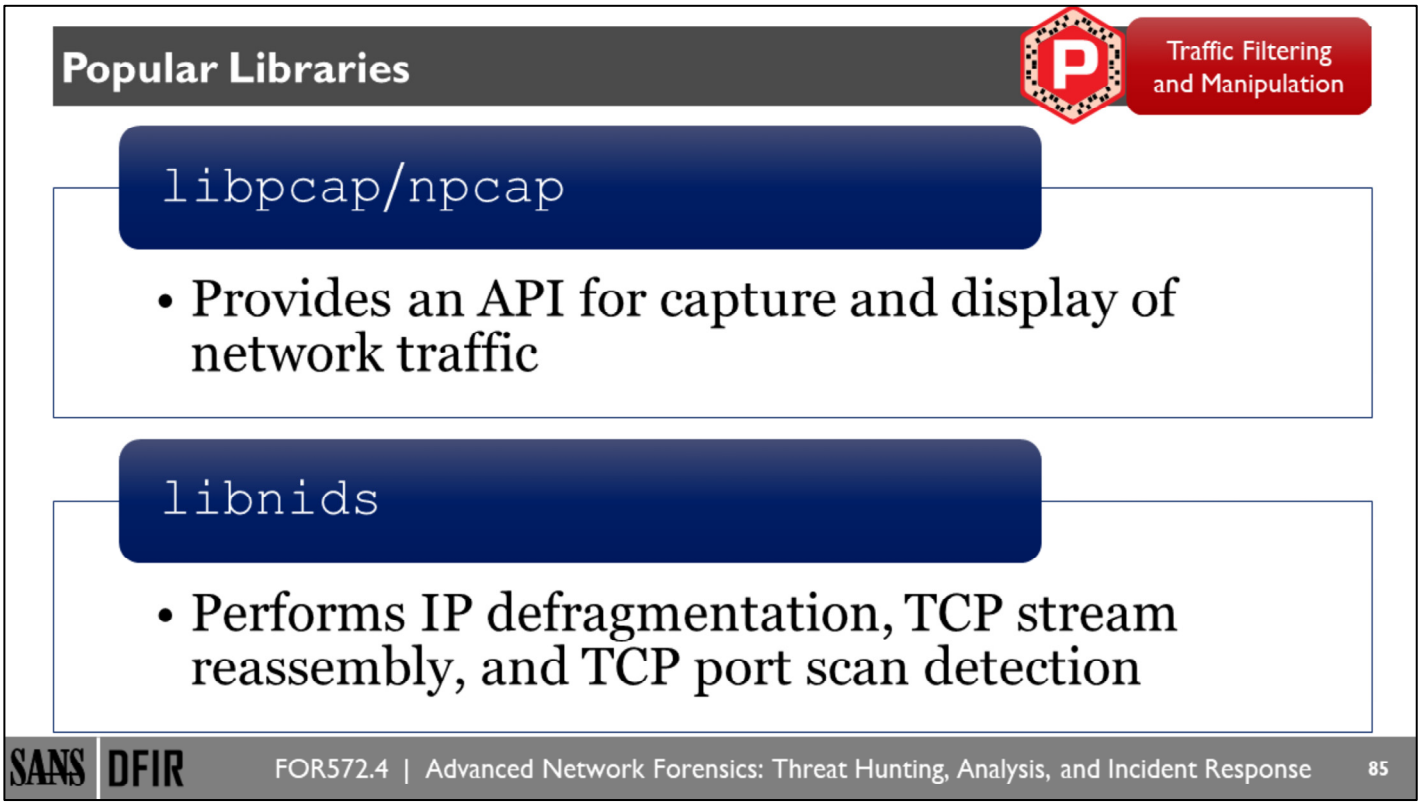


This page intentionally left blank.



As with most technical tasks, there are a number of tools that an individual may use to accomplish a specific task. Each tool provides the user a specific set of capabilities and leaves the user to determine whether it may or may not be the best tool for the job. Network forensics and analysis are no different. Many of the early open-source tools were developed to provide solutions to specific problems (such as `tcpdump` and `tcpflow`), while other higher-level tools take ideas from the early applications and extend them (Wireshark). For our purposes, we will use the following four broad categories to describe the tools' functionalities:

- Traffic Filtering and Manipulation — These tools provide users the basic ability to slice and dice network traffic.
- Traffic Searching — These tools provide users the ability to quickly search through traffic based on unique strings or byte sequences to determine whether the traffic of interest contains what the user is looking for.
- Statistical Analysis — These tools provide users the ability to obtain information to detect anomalies or provide some ability to quantify metrics associated with network traffic.
- Network Forensics and Analysis Tools — These tools typically provide users the ability to extract additional information from the application layer of network protocols to include images, files, and other items beyond just metadata.



There are several libraries that abstract the lower level mechanics of manipulating raw traffic collected from a network interface to perform some action. Two of the most popular libraries used for development of applications include `libpcap` and `libnids`.

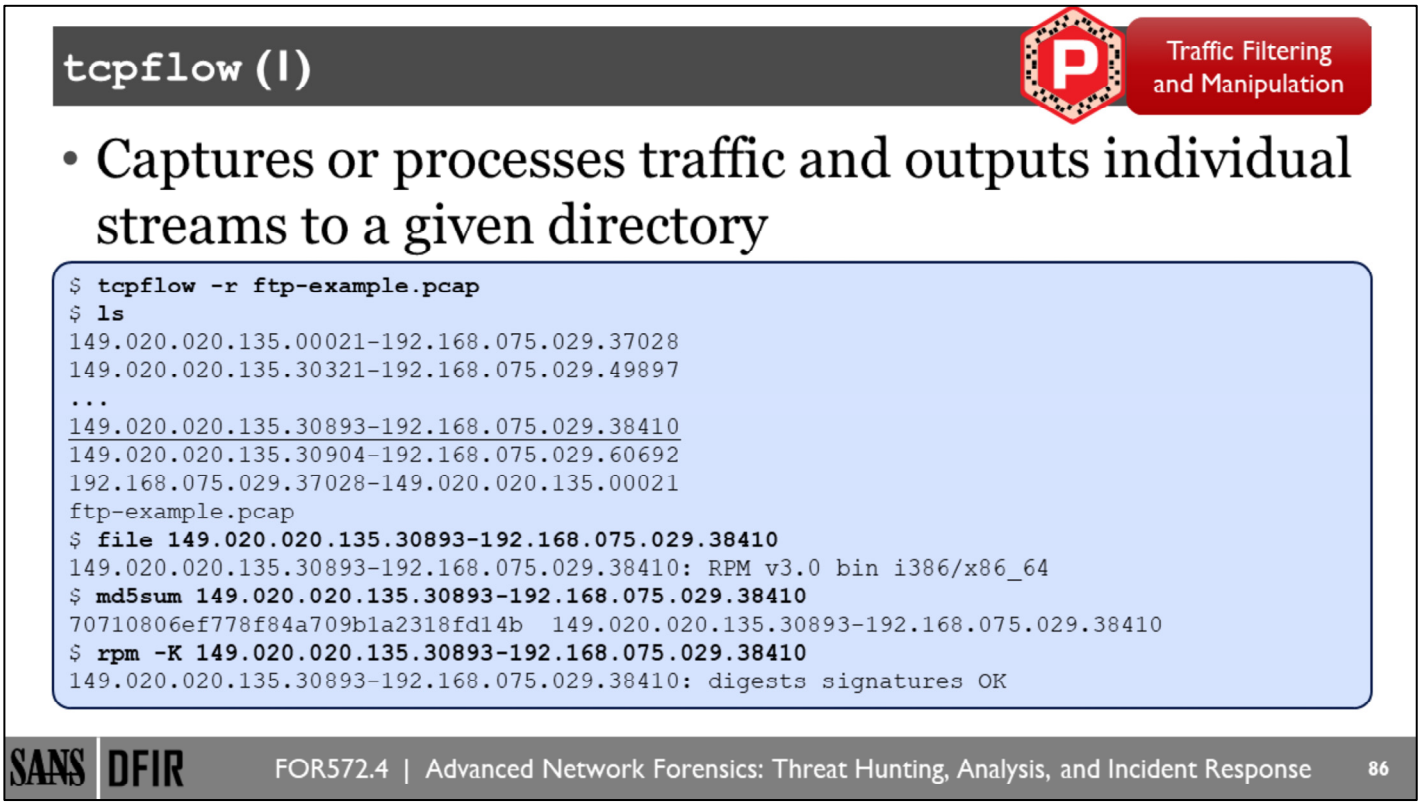
The `libpcap` libraries were originally developed by the Network Research Group at Lawrence Berkeley Laboratory (LBL). In the late 1980s, three researchers from LBL developed the tcpdump application to allow them to capture and display network traffic from a system connected to the LBL network. The `libpcap` library came about through stripping the packet capture code from the `tcpdump` application into a consolidated library and API that could then be used with other applications. Today, many open-source and commercial applications use the `libpcap` library as a foundation to handle the traffic capture and filtering portions of the application.

The developer of the venerable Nmap network scanning tool has released `npcap`[1], which provides a `libpcap` equivalent for those running the Microsoft Windows operating system.

The popular `libnids` library was developed in response to a paper published in January 1998 by Thomas H. Ptacek and Timothy N. Newsham called "Eluding Network Intrusion Detection."[2] As part of the paper, Ptacek and Newsham explained a number of ways to bypass network intrusion detection systems (NIDS). `libnids` was developed in response to the paper as a way to mitigate ways to bypass NIDS devices, which included implementing the ability to perform IP defragmentation, TCP stream reassembly, and TCP port scan detection.

Both libpcap and libnids have a number of wrappers for use with various programming languages, including C/C++, Python, Perl, and Java, and they are available on both UNIX- and Windows-based systems.

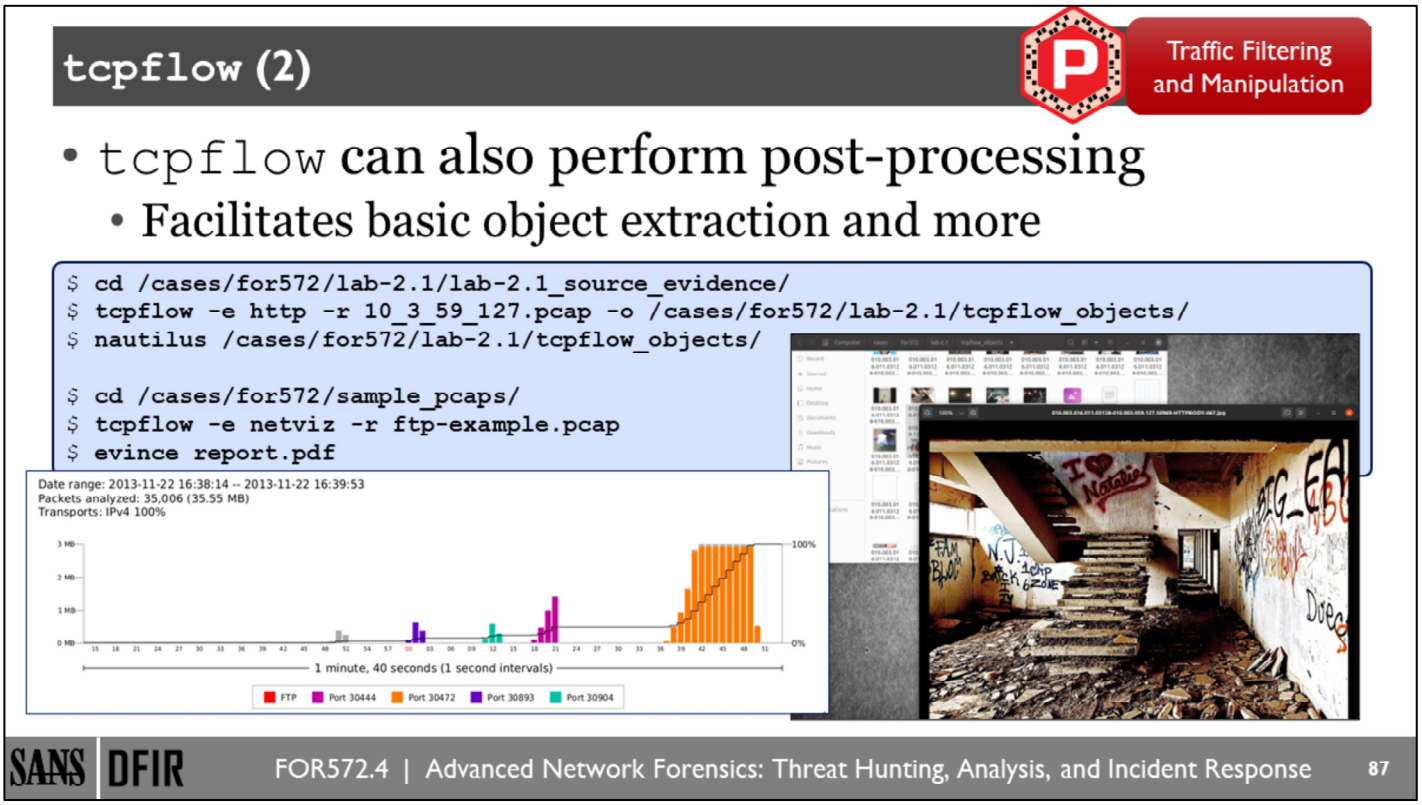
References:
[1] https://for572.com/gy8k7
[2] https://for572.com/5uf9t



Similar to `tcpdump`, `tcpflow` is a command-line, libpcap-based application that captures data transmitted across network interfaces or processes data from network captures. It can filter network traffic using the Berkeley Packet Filter (BPF) syntax. However, unlike `tcpdump`, `tcpflow` is knowledgeable of the sequence numbers associated with TCP connections, allowing it to perform reconstruction of the data streams and store each flow in its own file for later analysis. The output files contain the continuous data portion of the source, not new pcap files. `tcpflow` is commonly used for understanding network packet flows and performing network forensics, for example, reassembling the contents of HTTP sessions. Using `tcpflow`, a user can reconstruct web pages downloaded via HTTP or even extract malware or files transferred via HTTP or FTP. `tcpflow` is also commonly used during the analysis of undocumented network protocols aiding in reverse engineering of the protocols.

The preceding example shows `tcpflow` processing the `ftp-example.pcap` file from the `/cases/for572/sample_pcaps/` directory in your FOR572 SIFT VM. The `149.020.020.135.30893-192.168.075.029.38410` stream corresponds to a particular RPM file, which the subsequent commands validate.

References:
https://for572.com/odbi7
https://for572.com/wujhc

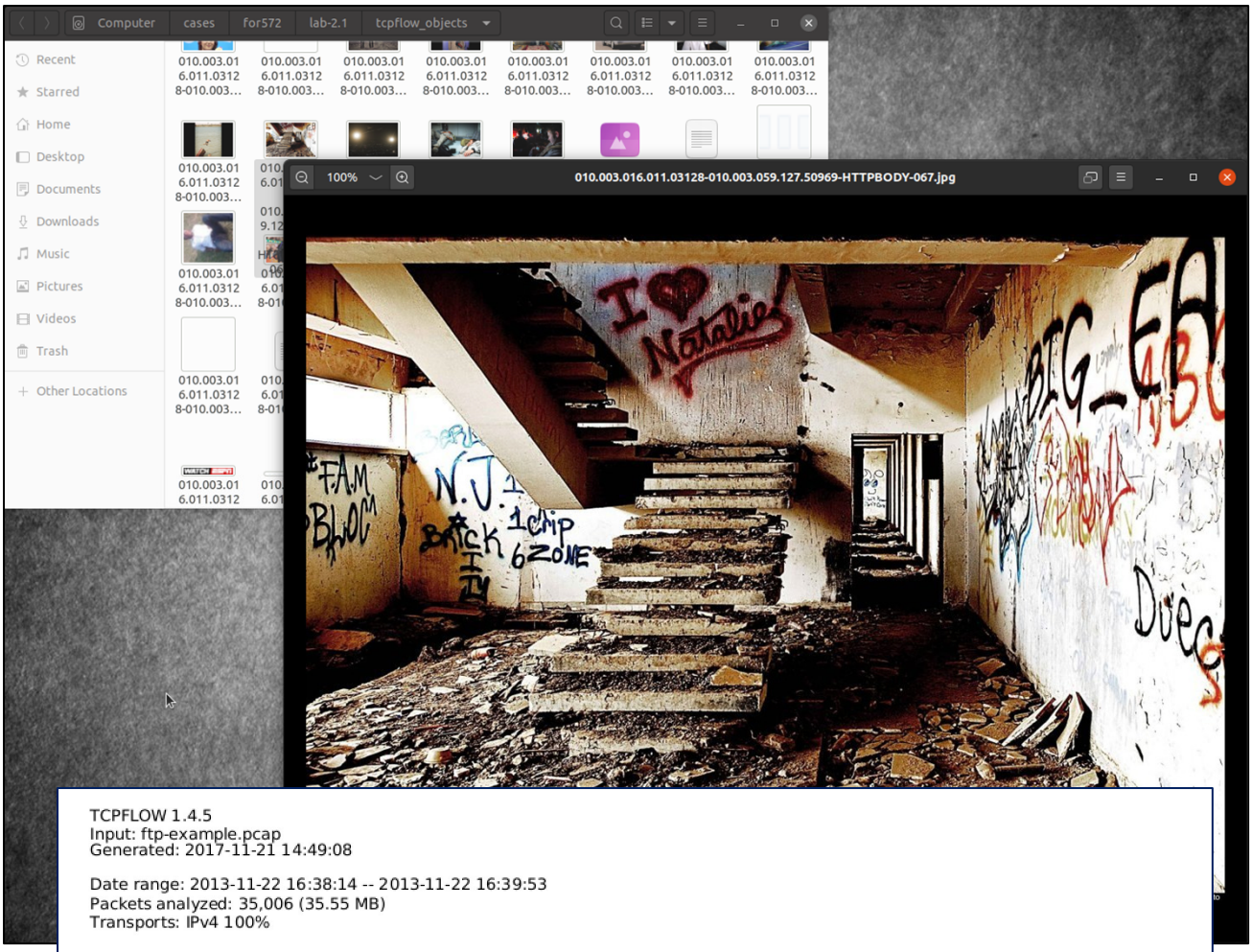


Additional `tcpflow` functionality that can be of value to the analyst includes automated post-processing for certain protocols and visualizations via the "`-e`" flag.

Using "`-e http`", for example, will not only output the reassembled per-flow data segments from the input data, but will also identify and reconstruct HTTP objects. This allows handling via the filesystem, for example. The slide above depicts reconstruction from the evidence provided in Lab 2.1.

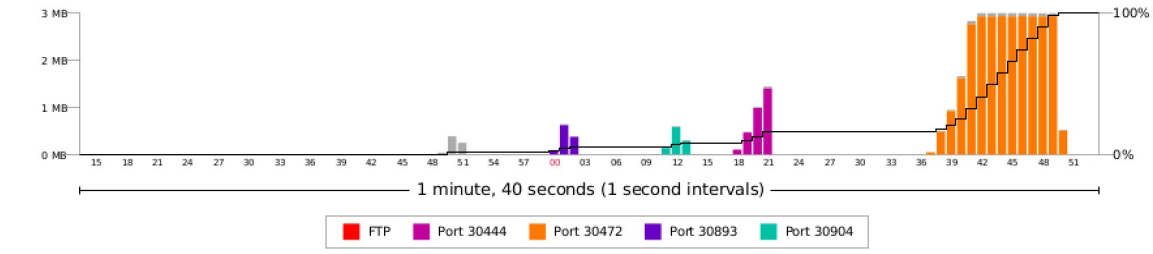
Visualization with the "`-e netviz`" option can also benefit the analyst by providing a quick overview of source data in a convenient PDF file. The slide above and the enlarged version on the next page shows the output generated from the "`ftp-example.pcap`" file, available in the "`/cases/for572/sample_pcaps/`" directory within your SIFT Workstation VM. The graphical results show port usage over time, as well as top source and destination IP addresses and ports.

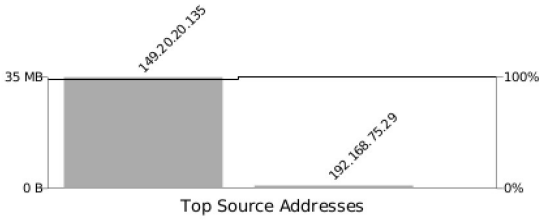
For both use cases, `tcpflow` provides quick, automated, and scalable parsing of input data that can be integrated to an overall network forensic workflow.



TCPFLOW 1.4.5
Input: ftp-example.pcap
Generated: 2017-11-21 14:49:08

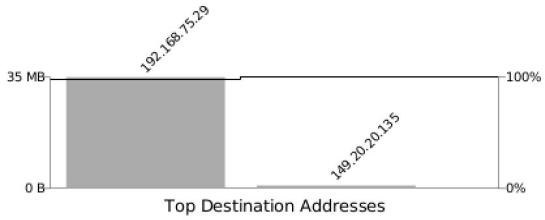
Date range: 2013-11-22 16:38:14 -- 2013-11-22 16:39:53
Packets analyzed: 35,006 (35.55 MB)
Transports: IPv4 100%





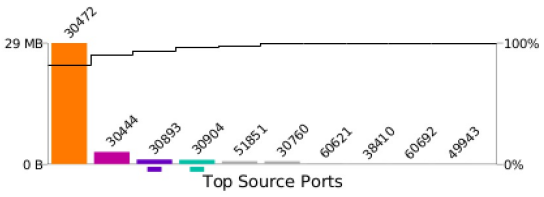
Top Source Addresses

1) 149.20.20.135 - 34.76 MB (97%)
2) 192.168.75.29 - 790.98 KB (2%)



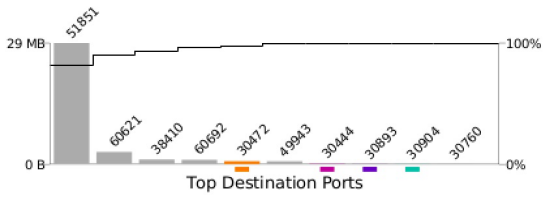
Top Destination Addresses

1) 192.168.75.29 - 34.76 MB (97%)
2) 149.20.20.135 - 790.98 KB (2%)



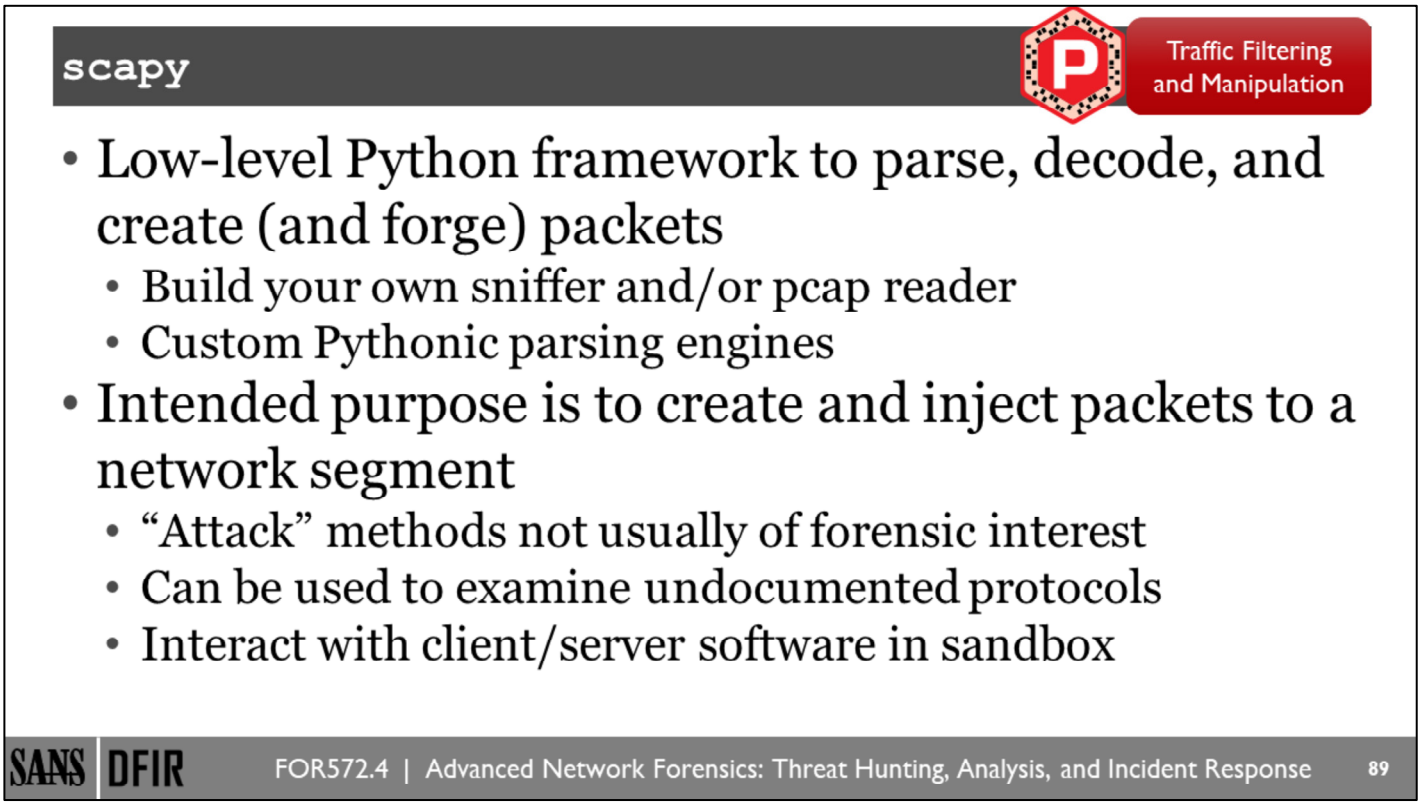
Top Source Ports

1) 30472 - 29.16 MB (82%)
2) 30444 - 2.89 MB (8%)
3) 30893 - 1.07 MB (2%)



Top Destination Ports

1) 51851 - 29.16 MB (82%)
2) 60621 - 2.89 MB (8%)
3) 38410 - 1.07 MB (2%)

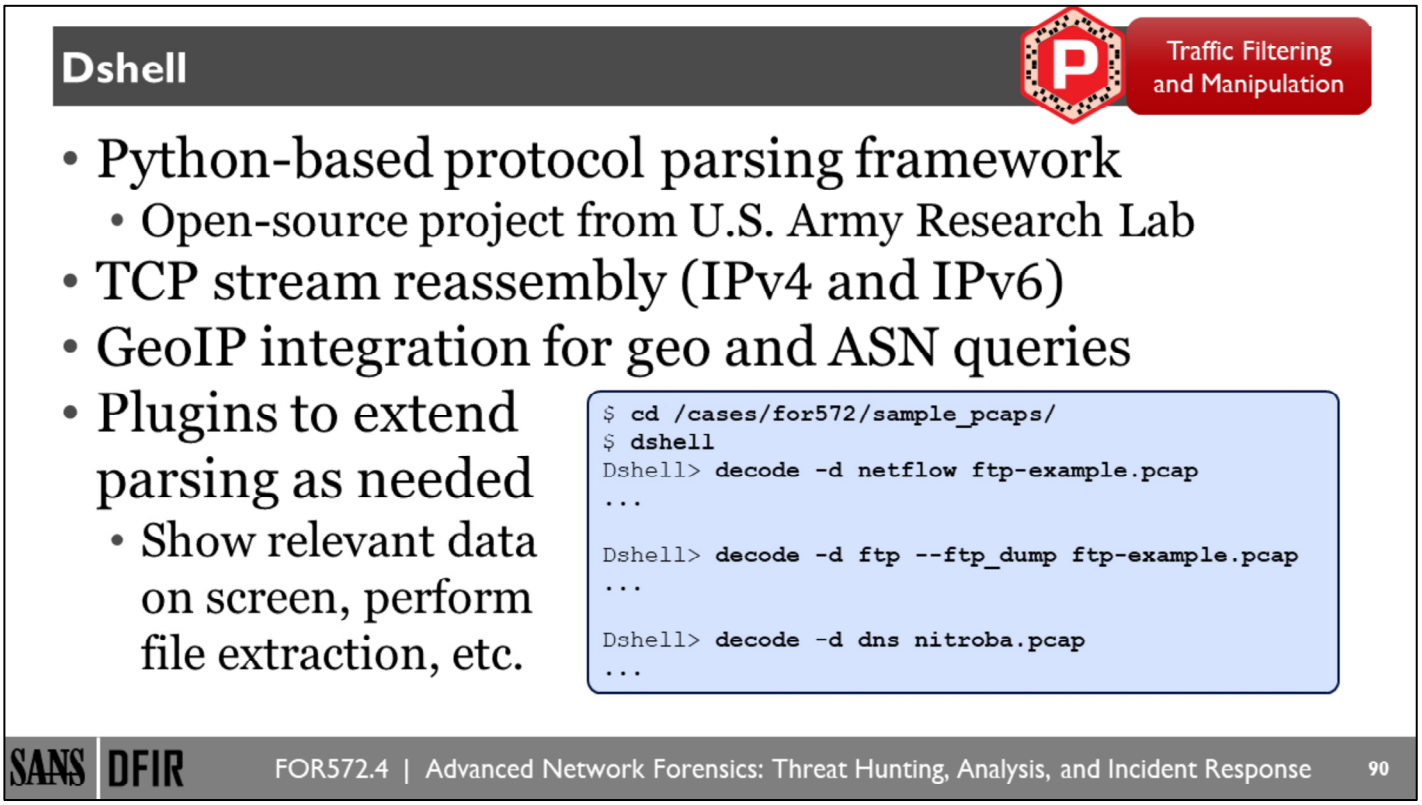


The `scapy`[1] framework is a Python tool commonly used by penetration testers and attackers alike. Although this is not a primary focus for the forensically focused investigator, it can still be a good tool to understand. If faced with an undocumented protocol, for example, a researcher could use `scapy` to build a custom parser for command and control traffic, use it to extract certain parameters from an HTTP stream, reconstruct files, etc. A reverse engineer could use it to interface with malware executing in a controlled sandbox environment.

`scapy` can also be used to sniff live packets from a network interface as well as read packet data from a pcap file. For example, a DNS sniffer can be created with just a handful of lines of Python code.[2] This type of functionality can be extended for new and existing protocols and could be a great benefit for tactical network monitoring during an incident response.

References:

[1] https://for572.com/6bvmx
[2] https://for572.com/ugwon



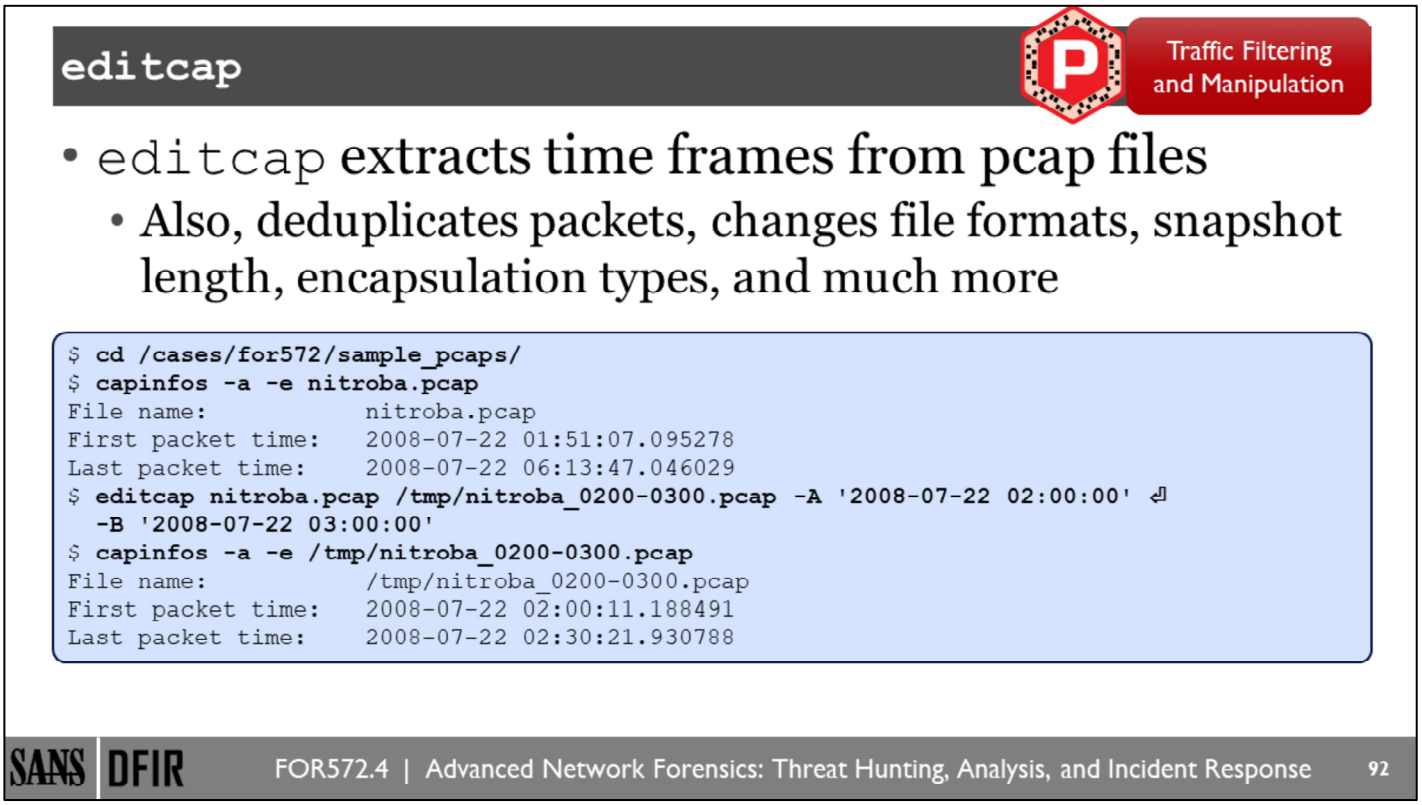
Another promising entrant into the field of traffic parsing utilities is Dshell,[1] a Python-based protocol parsing framework. It is fully free and open source, and anyone with a working knowledge of Python can write general or incident-specific parsers and analytic helper modules to aid in their work.

In the examples on the next page, you can see the results of running Dshell against the "`ftp-example.pcap`" file in the "`/cases/for572/sample_pcaps/`" directory within your SIFT Workstation VM, including a general NetFlow-like view showing the single command channel connection and multiple data channels, as well as a full object extractor, reconstructing the files retrieved via the data channels. The final command performs a DNS dump from the "`nitroba.pcap`" file in the same directory.

References:

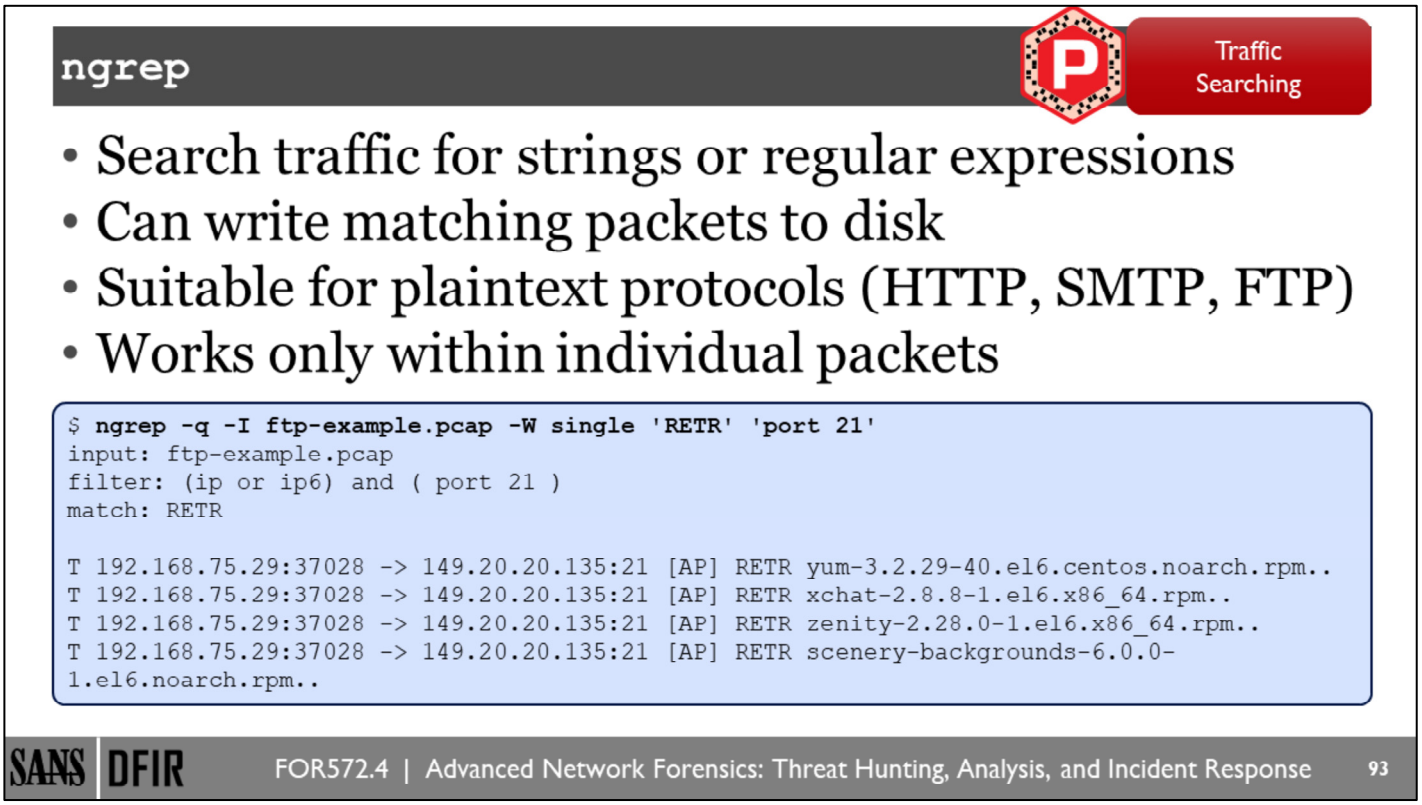
[1] https://for572.com/cmdlh

```
$ cd /cases/for572/sample_pcaps/
$ dshell
Dshell> decode -d netflow ftp-example.pcap
2013-11-22 16:38:30.883332     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   60090   30351     0      2        0     2896  0.1906s
2013-11-22 16:38:33.889745     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   49897   30321     0      0        0        0  0.2079s
2013-11-22 16:38:38.452075     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   37110   30487     0      0        0        0  0.1772s
2013-11-22 16:38:42.970871     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   37583   30669     0      0        0        0  0.1780s
2013-11-22 16:38:49.561496     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   49943   30760     0    424        0   612916  2.2349s
2013-11-22 16:39:00.453416     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   38410   30893     0    705        0  1019540  1.8620s
2013-11-22 16:39:11.299795     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   60692   30904     0    654        0   945952  1.9983s
2013-11-22 16:39:18.431439     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   60621   30444     0   1912        0  2767424  3.3674s
2013-11-22 16:39:37.628648     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   51851   30472     0  19260        0  27888036  12.5452s
2013-11-22 16:38:14.266541     192.168.75.29 ->    149.20.20.135  (-- -> US)  TCP   37028      21    28     43      359     3660  99.3606s

Dshell> decode -d ftp --ftp_dump ftp-example.pcap
ftp 2013-11-22 16:38:14    192.168.75.29:37028 ->    149.20.20.135:21    ** User: ftp, Pass: for572@lewestech.com, RETR File: centos/LIST
(2896 bytes written to RETR_centos_LIST) **
ftp 2013-11-22 16:38:14    192.168.75.29:37028 ->    149.20.20.135:21    ** User: ftp, Pass: for572@lewestech.com, RETR File: centos/6.4/LIST
**
ftp 2013-11-22 16:38:14    192.168.75.29:37028 ->    149.20.20.135:21    ** User: ftp, Pass: for572@lewestech.com, RETR File:
centos/6.4/os/LIST **
ftp 2013-11-22 16:38:14    192.168.75.29:37028 ->    149.20.20.135:21    ** User: ftp, Pass: for572@lewestech.com, RETR File:
centos/6.4/os/x86_64/LIST **
ftp 2013-11-22 16:38:14    192.168.75.29:37028 ->    149.20.20.135:21    ** User: ftp, Pass: for572@lewestech.com, RETR File:
centos/6.4/os/x86_64/Packages/LIST (611468 bytes written to RETR_centos_6.4_os_x86_64_Packages_LIST) **
ftp 2013-11-22 16:38:14    192.168.75.29:37028 ->    149.20.20.135:21    ** User: ftp, Pass: for572@lewestech.com, RETR File:
centos/6.4/os/x86_64/Packages/yum-3.2.29-40.el6.centos.noarch.rpm (1019540 bytes written to RETR_centos_6.4_os_x86_64_Packages_yum-3.2.29-
40.el6.centos.noarch.rpm) **
ftp 2013-11-22 16:38:14    192.168.75.29:37028 ->    149.20.20.135:21    ** User: ftp, Pass: for572@lewestech.com, RETR File:
centos/6.4/os/x86_64/Packages/xchat-2.8.8-1.el6.x86_64.rpm (945952 bytes written to RETR_centos_6.4_os_x86_64_Packages_xchat-2.8.8-
1.el6.x86_64.rpm) **
ftp 2013-11-22 16:38:14    192.168.75.29:37028 ->    149.20.20.135:21    ** User: ftp, Pass: for572@lewestech.com, RETR File:
centos/6.4/os/x86_64/Packages/zenity-2.28.0-1.el6.x86_64.rpm (2767424 bytes written to RETR_centos_6.4_os_x86_64_Packages_zenity-2.28.0-
1.el6.x86_64.rpm) **
ftp 2013-11-22 16:38:14    192.168.75.29:37028 ->    149.20.20.135:21    ** User: ftp, Pass: for572@lewestech.com, RETR File:
centos/6.4/os/x86_64/Packages/scenery-backgrounds-6.0.0-1.el6.noarch.rpm (27888036 bytes written to
RETR_centos_6.4_os_x86_64_Packages_scenery-backgrounds-6.0.0-1.el6.noarch.rpm) **
```



`editcap` is distributed as part of the Wireshark suite of utilities. It is a command-line, libpcap-based application that extracts portions of pcap files based on the user's date and time specifications. The `editcap` utility also performs a wide variety of pcap file manipulation options, which can help to derive pcap files for use in other pcap-aware utilities. For example, it can deduplicate packets in a merged capture file per a variety to characteristics, convert pcapng-formatted files to legacy pcap format, truncate packets per a specified snapshot length, and much more.

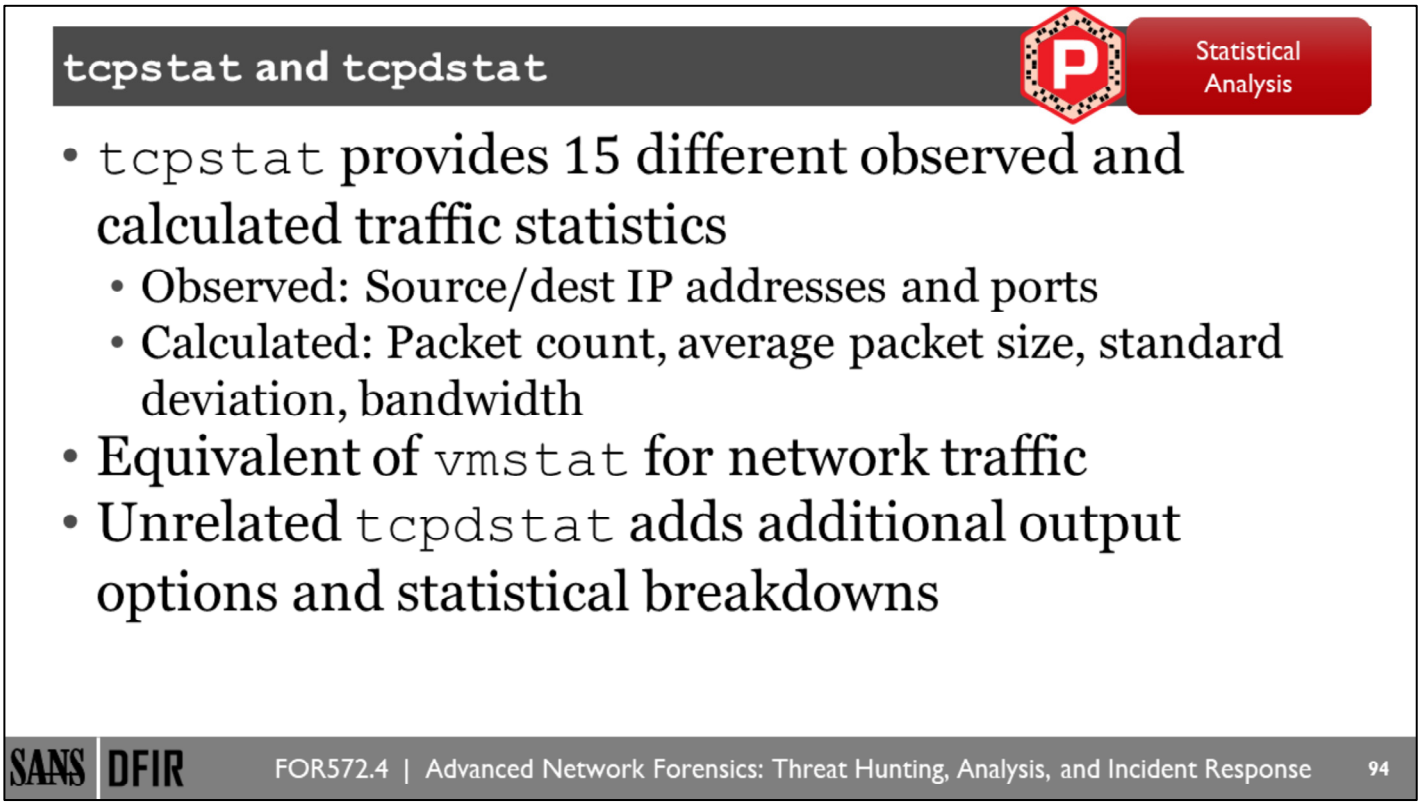
References:
https://for572.com/9zj-1



`ngrep`[1] (or network grep) is a `libpcap`-based application that allows users to specify extended regular or hexadecimal expressions to match against data payloads of packets. Similar to most other `libpcap`-based tools, it can filter network traffic using the Berkeley Packet Filter (BPF) syntax. `ngrep` is commonly used to quickly obtain information from plaintext protocols such as HTTP, SMTP, FTP, etc. Like `tcpdump`, `ngrep` can also read from a live network interface or capture file and can also capture data to a pcap file. However, because `ngrep` is packet-based, if the regular expression's target pattern crosses packet boundaries, `ngrep` will not match the pattern.

References:

[1] https://for572.com/1jey-

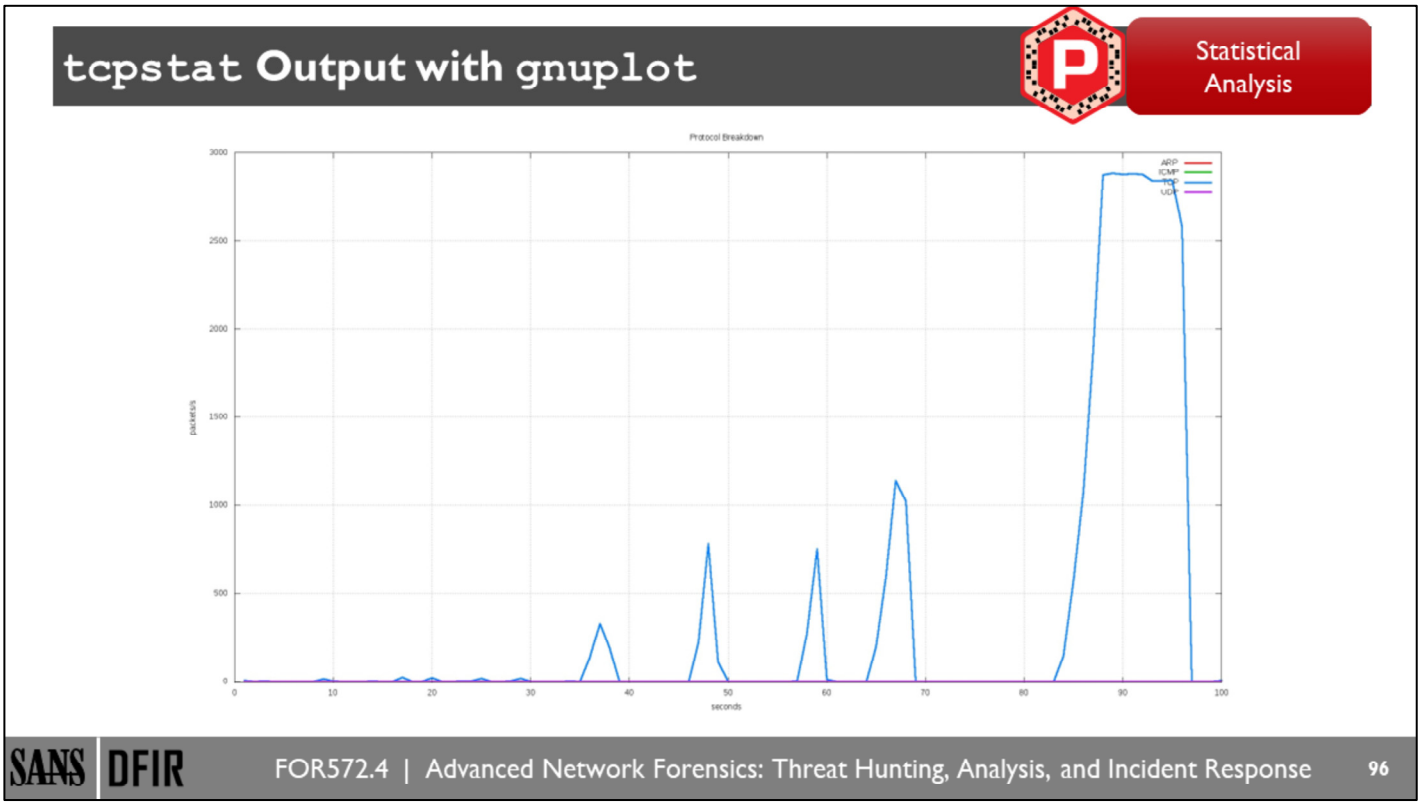


`tcpstat` is a libpcap-based application that provides network interface statistics. `tcpstat` acquires this information either by monitoring a specific interface or by reading an existing pcap file. It provides more than 15 different traffic statistics, including packet count, average packet size, standard deviation of packet size, and data rate in bits per second. `tcpstat` was designed to provide a network-focused cousin to the popular vmstat program that monitors host-based performance such as CPU, memory, and I/O performance.

A similar program, `tcpdstat`, was developed by Sony of Japan and also provides several output options including packet count, average data rate and standard deviation, the pairs of unique source and destination IP addresses, and information based on a protocol breakdown.

References:
https://for572.com/esydj
https://for572.com/p6zwq
https://for572.com/qql7p

```
$ cd /cases/for572/sample_pcaps/
$ tcpstat -r ftp-example.pcap
Time:1385138297        n=8         avg=64.50   stddev=13.74           bps=825.60
Time:1385138302        n=16        avg=185.19  stddev=346.60          bps=4740.80
Time:1385138307        n=3         avg=68.33   stddev=15.41           bps=328.00
Time:1385138312        n=41        avg=168.51  stddev=345.11          bps=11054.40
Time:1385138317        n=20        avg=68.25   stddev=29.21           bps=2184.00
Time:1385138322        n=20        avg=109.00  stddev=202.21          bps=3488.00
Time:1385138327        n=3         avg=69.00   stddev=15.25           bps=331.20
Time:1385138332        n=646       avg=1001.02 stddev=687.42          bps=1034659.20
Time:1385138337        n=0         avg=0.00    stddev=0.00 bps=0.00
Time:1385138342        n=1110      avg=970.75  stddev=696.94          bps=1724059.20
Time:1385138347        n=0         avg=0.00    stddev=0.00 bps=0.00
Time:1385138352        n=1026      avg=974.20  stddev=695.83          bps=1599243.20
Time:1385138357        n=197       avg=913.02  stddev=709.91          bps=287782.40
Time:1385138362        n=2749      avg=997.08  stddev=689.32          bps=4385568.00
Time:1385138367        n=0         avg=0.00    stddev=0.00 bps=0.00
Time:1385138372        n=0         avg=0.00    stddev=0.00 bps=0.00
Time:1385138377        n=725       avg=971.04  stddev=696.92          bps=1126411.20
Time:1385138382        n=11580     avg=1003.58 stddev=687.30          bps=18594304.00
Time:1385138387        n=14274     avg=1012.77 stddev=684.19          bps=23130009.60
Time:1385138392        n=2588      avg=1013.64 stddev=683.75          bps=4197260.80

$ tcpdstat ftp-example.pcap

DumpFile:  ftp-example.pcap
FileSize: 34.44MB
Id: 201311221638
StartTime: Fri Nov 22 16:38:14 2013
EndTime:   Fri Nov 22 16:39:53 2013
TotalTime: 99.36 seconds
TotalCapSize: 33.91MB  CapLen: 1514 bytes
# of packets: 35006 (33.91MB)
AvgRate: 3.14Mbps  stddev:6.84M   PeakRate: 23.46Mbps

### IP flow (unique src/dst pair) Information ###
# of flows: 2  (avg. 17503.00 pkts/flow)
Top 10 big flow size (bytes/total in %):
 97.8%  2.2%

### IP address Information ###
# of IPv4 addresses: 2
Top 10 bandwidth usage (bytes/total in %):
 100.0% 100.0%
### Packet Size Distribution (including MAC headers) ###
<<<<
 [   64-  127]:      12036
 [  128-  255]:          8
 [  256-  511]:          5
 [  512- 1023]:          3
 [ 1024- 2047]:      22954
>>>>


### Protocol Breakdown ###
<<<<
     protocol           packets                 bytes.          bytes/pkt
------------------------------------------------------------------------
[0] total            35006 (100.00%)         35553990 (100.00%)   1015.65
[1] ip               35006 (100.00%)         35553990 (100.00%)   1015.65
[2]  tcp             35006 (100.00%)         35553990 (100.00%)   1015.65
[3]   ftp              120 (  0.34%)            11959 (  0.03%)     99.66
[3]   other          34886 ( 99.66%)         35542031 ( 99.97%)   1018.80
>>>>
```



This graph depicts the output of `tcpstat` data when plotted by `gnuplot`. Contained within the graph are examples of network traffic breakdown by protocol. Through the combination of `tcpstat` with a `gnuplot` script, an analyst can produce graphs to identify spikes in activity indicative of potential data theft or file transfers.

```
$ cd /home/sansforensics/
$ cp -a /cases/for572/sample_pcaps/ftp-example.pcap ./
$ tcpstat -r ftp-example.pcap -o "%R\t%A\n" 1 > arp.data
$ tcpstat -r ftp-example.pcap -o "%R\t%C\n" 1 > icmp.data
$ tcpstat -r ftp-example.pcap -o "%R\t%T\n" 1 > tcp.data
$ tcpstat -r ftp-example.pcap -o "%R\t%U\n" 1 > udp.data
```

The "`-o`" option indicates the output format for traffic over the specified time interval (in this case, every one second). The output is then redirected into a specified data file. `gnuplot` uses a scripting language to generate the graphs. The following gnuplot script is available in your SIFT Workstation VM as "`/cases/for572/sample_pcaps/ftp-example.gnuplot`":

```
set output "proto_breakdown_over_time.png";
set term png size 1024, 768;
set grid;
set yrange [ -10 : 3000 ];
set title "Protocol Breakdown";
set xlabel "seconds";
set ylabel "packets/s";
plot "arp.data" using 1:($2) lw 3 smooth csplines title "ARP", \
     "icmp.data" using 1:($2) lw 3 smooth csplines title "ICMP", \
     "tcp.data" using 1:($2) lw 3 smooth csplines title "TCP", \
     "udp.data" using 1:($2) lw 3 smooth csplines title "UDP";
```

The following commands then generate and view the actual graph image file:

```
$ cd /home/sansforensics/
$ gnuplot /cases/for572/sample_pcaps/ftp-example.gnuplot
$ eog proto_breakdown_over_time.png
```

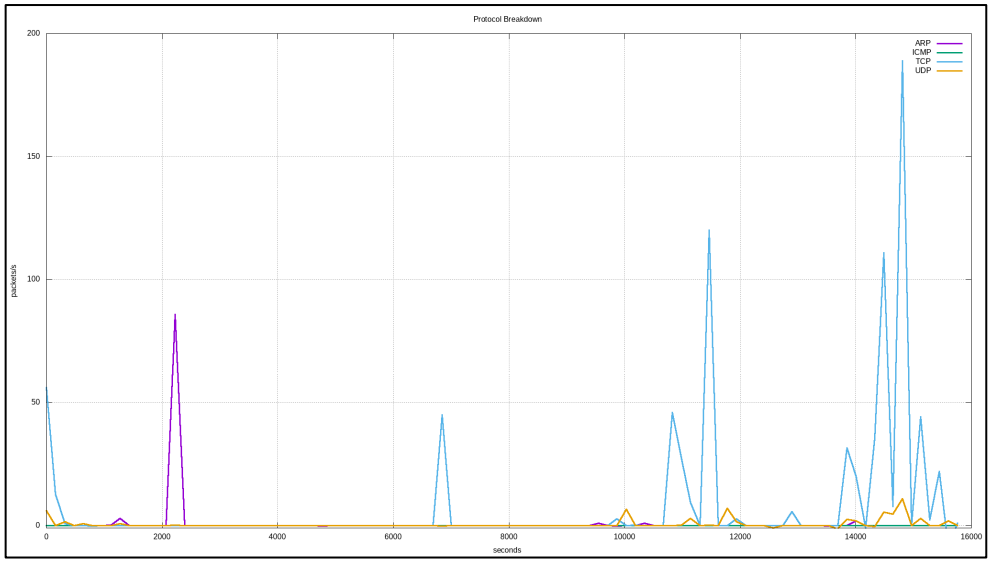
For another option, consider the much larger and more diverse collection of traffic in the "`nitroba.pcap`" file, also in the "`/cases/for572/sample_pcaps/`" directory.

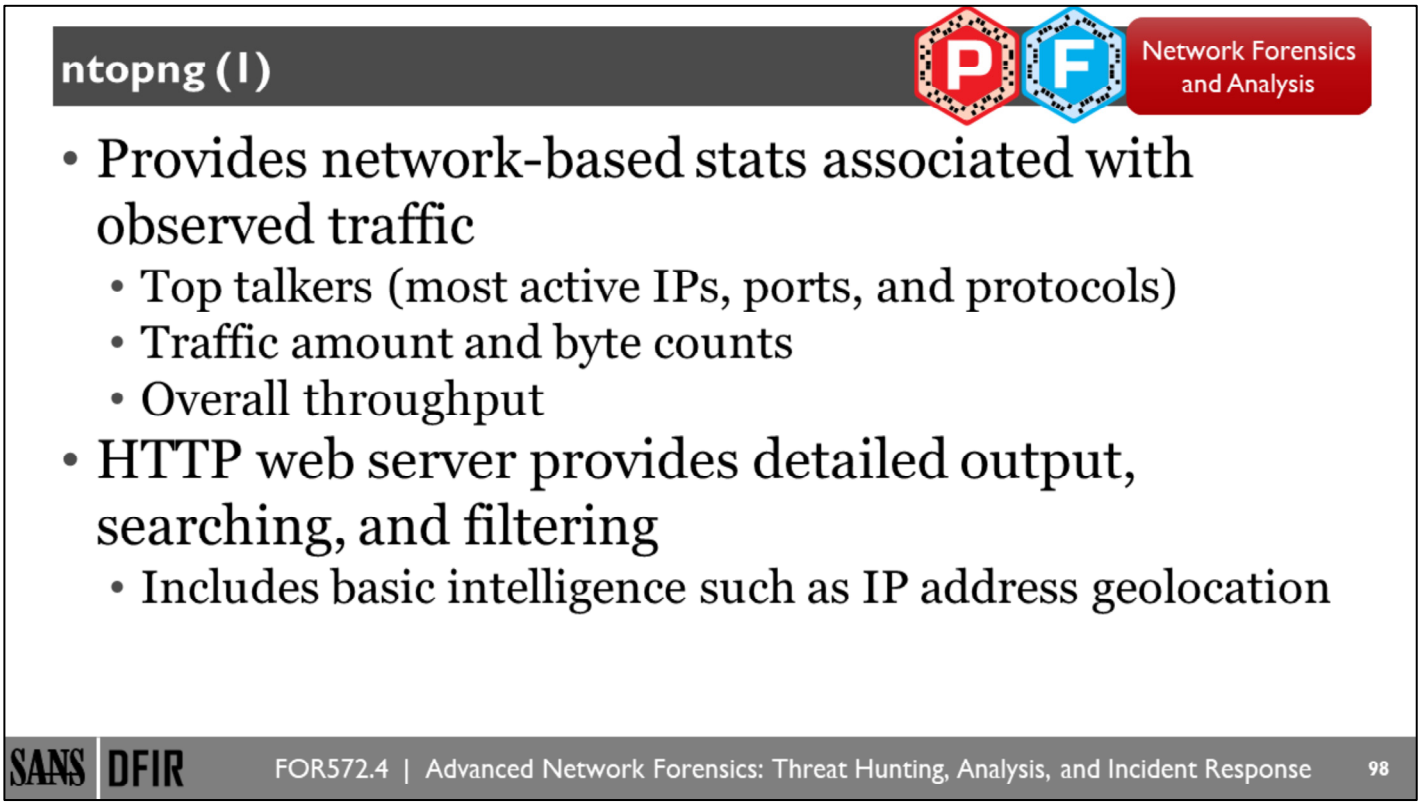
```
$ cd /home/sansforensics/
$ cp -a /cases/for572/sample_pcaps/nitroba.pcap ./
$ tcpstat -r nitroba.pcap -o "%R\t%A\n" 1 > nitroba_arp.data
$ tcpstat -r nitroba.pcap -o "%R\t%C\n" 1 > nitroba_icmp.data
$ tcpstat -r nitroba.pcap -o "%R\t%T\n" 1 > nitroba_tcp.data
$ tcpstat -r nitroba.pcap -o "%R\t%U\n" 1 > nitroba_udp.data
```

The following gnuplot script is available in your SIFT Workstation VM as "`/cases/for572/sample_pcaps/nitroba.gnuplot`":

```
set output "nitroba_proto_breakdown_over_time.png";
set term png size 1024, 768;
set grid;
set yrange [ -10 : 3000 ];
set title "Protocol Breakdown";
set xlabel "seconds";
set ylabel "packets/s";
plot "nitroba_arp.data" using 1:($2) lw 3 smooth csplines title "ARP", \
     "nitroba_icmp.data" using 1:($2) lw 3 smooth csplines title "ICMP", \
     "nitroba_tcp.data" using 1:($2) lw 3 smooth csplines title "TCP", \
     "nitroba_udp.data" using 1:($2) lw 3 smooth csplines title "UDP";
```

```
$ cd /home/sansforensics/
$ gnuplot /cases/for572/sample_pcaps/nitroba.gnuplot
$ eog nitroba_proto_breakdown_over_time.png
```

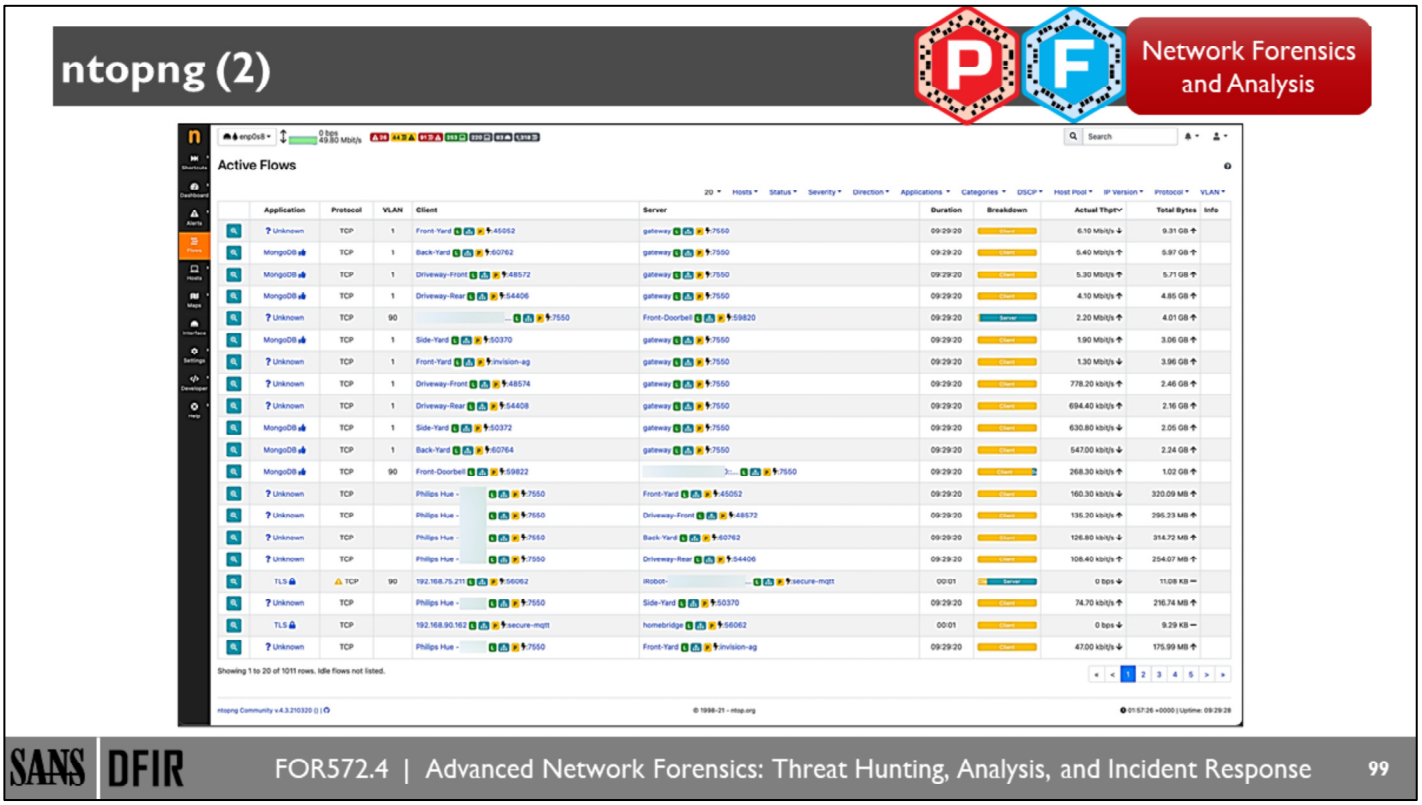




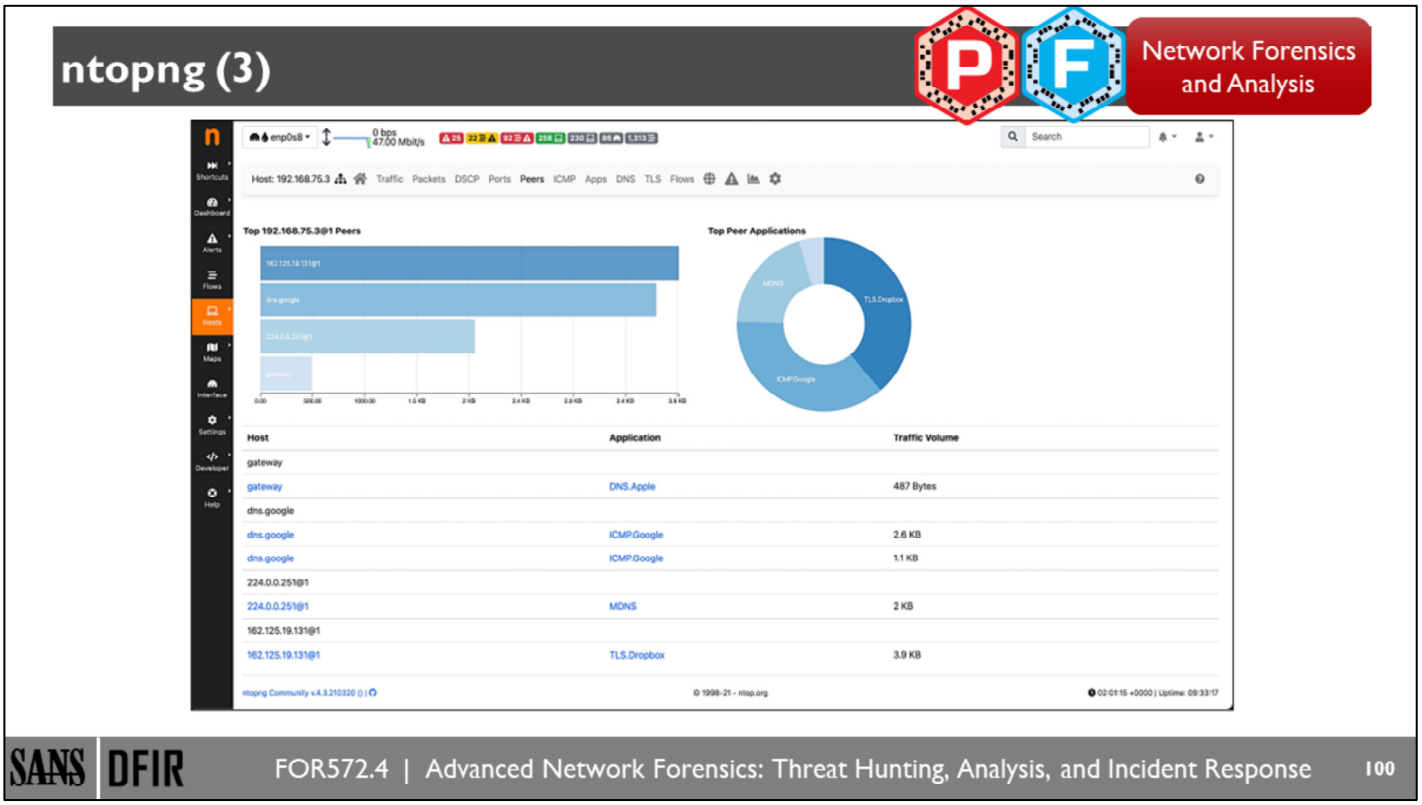
Similar to the venerable UNIX "top" command, ntopng provides real-time usage information associated with network usage. ntopng is a libpcap-based application that operates by observing active network interfaces in real-time to provide statistical information associated with the observed traffic. ntopng provides an interactive mode where information is displayed to the command-line interface and a web-based mode, which provides the statistical information via a web browser. It can also receive NetFlow data and perform targeted full-packet capture on a per-host basis.

However, it does not consume existing data, so it is limited to real-time usage and preincident deployment. In a tactical collection platform, however, ntopng provides solid performance in a freemium licensing model.

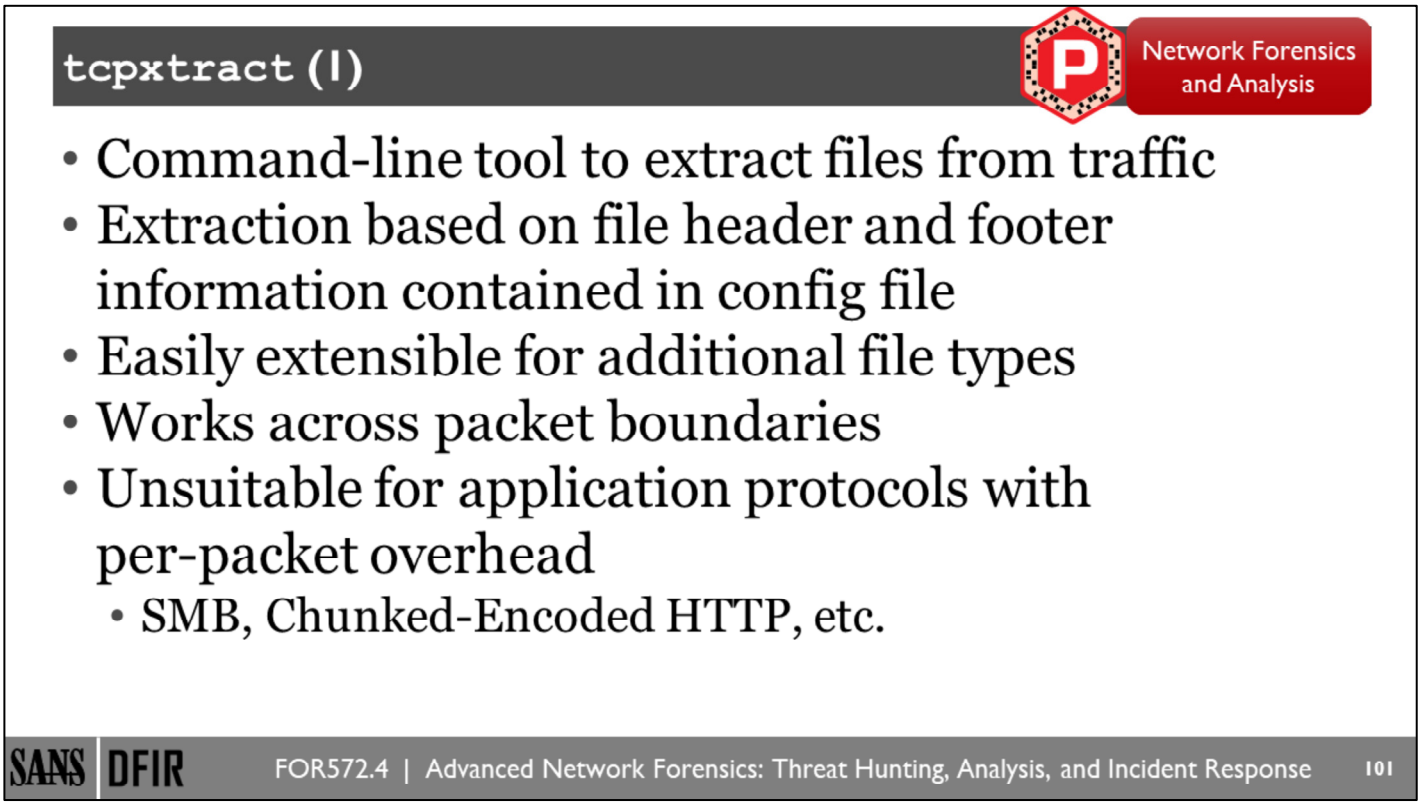
References:
https://for572.com/iysd4



From a user-interface perspective, ntopng provides a graphical interface including metadata associated with each flow. This metadata includes fields such as the client/server ports as determined by the TCP 3-way handshake and the amount of traffic transferred between the client and server over a given time period. You can also dive into individual flows to obtain more information or filter by client/server or dive deeper into traffic associated with a specific client or server address.



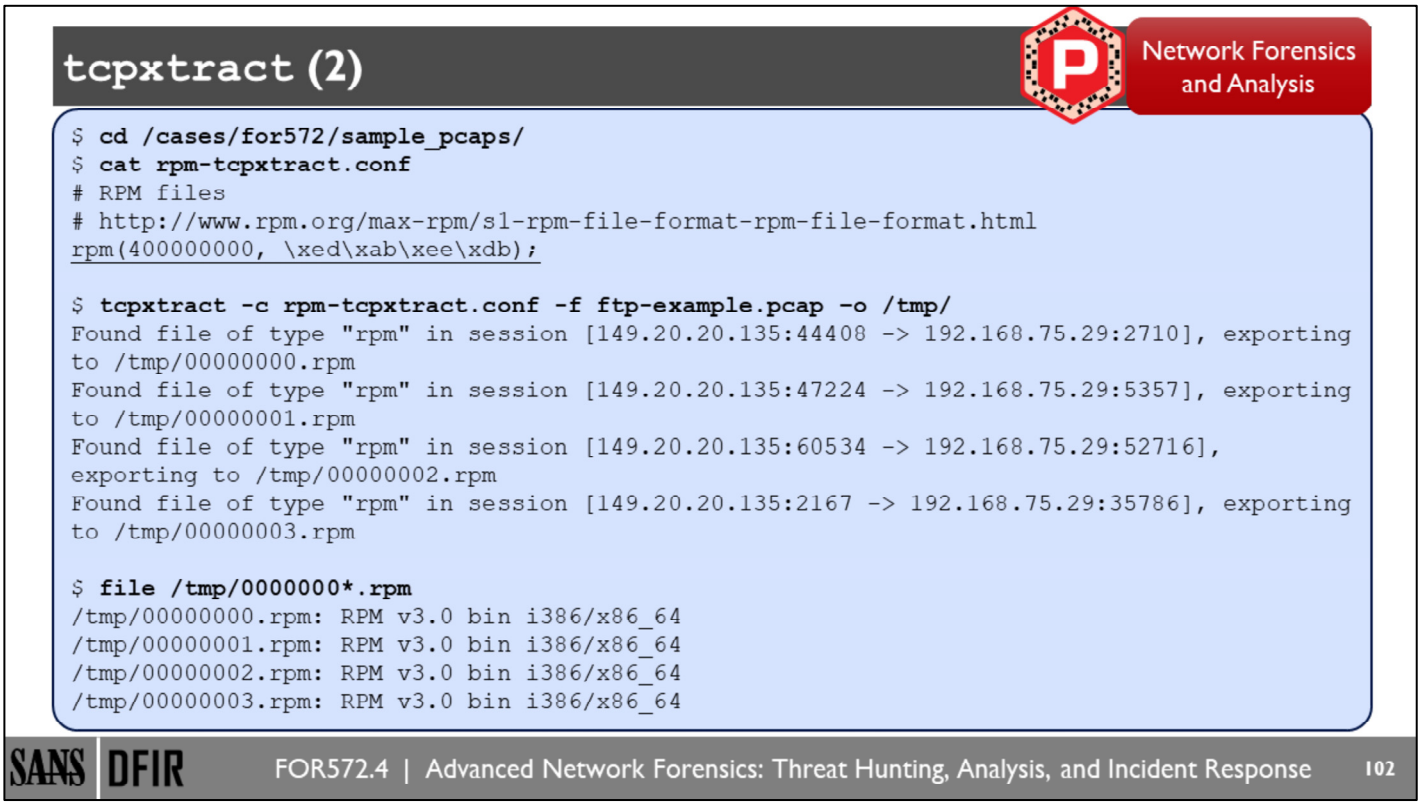
This screenshot indicates the type of information a user can obtain from ntopng when deep-diving into a specific host; however, similar information can be obtained from other metadata, such as ports. The upper-left portion of the screen contains information about what other peers (IPs or hostnames) the given IP address connected to and the amount of data transferred between the two peers. The upper-right graph contains a breakdown in the protocols used by the IP address. The bottom chart contains information on the individual flows, including the peer the original queried IP address connected to, the associated protocol, and the given amount of traffic. The graphic visualization aids users in making comparisons between different types of traffic, which may help users also find anomalies.



`tcpxtract`[1] is a command-line, `libpcap`-based application that extracts files from a live network stream or pcap file based on the "magic" file type headers and footers. If you're familiar with file carving from static media using tools like `foremost`[2] or `scalpel`,[3] `tcpxtract` does this against network data. It performs TCP stream reassembly and uses a configuration file to identify headers and footers of various file types. It easily extracts files such as images and office documents, and it conveniently handles data spread out across packet boundaries. Upon identifying a file, `tcpxtract` writes the file to disk. Note, however, that the filenames are not recovered. Instead, `tcpxtract` uses the frame number as the name for the carved file with the file extension defined in the configuration file.

Although `tcpxtract` is a handy tool to have, it is not aware of higher-layer protocols, such as SMB. Therefore, it is limited to protocols where file content is passed contiguously in a stream, without any Layer 5–7 headers in each packet.

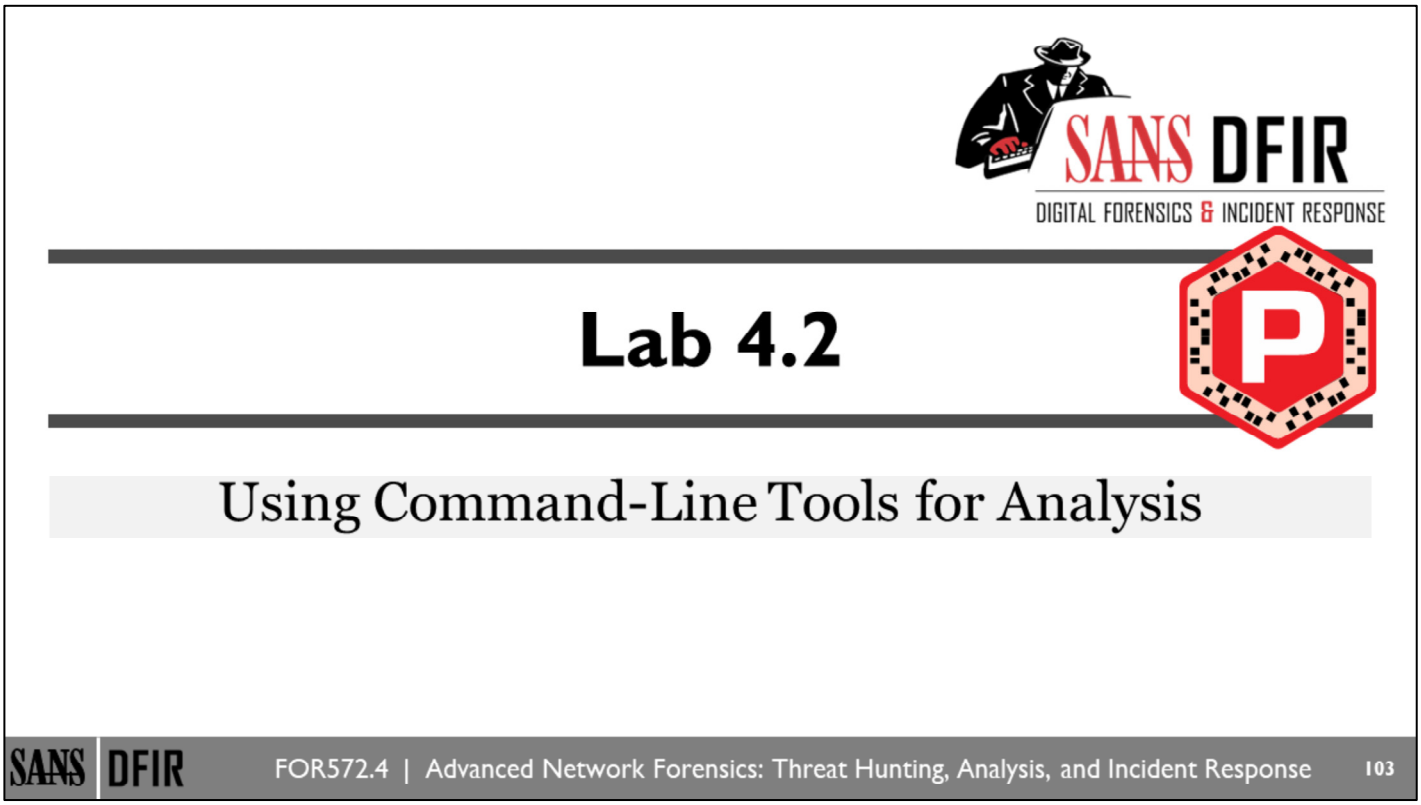
References:
[1] https://for572.com/uq1ok
[2] https://for572.com/5v4a0
[3] https://for572.com/yjeaw



The "`rpm-tcpxtract.conf`" file is provided within your SIFT Workstation VM in the "`/cases/for572/sample_pcaps/`" directory. It contains a single carving signature, which uses the magic values for the RPM file as defined by the file specification.[1] Running `tcpxtract` against the "`ftp-example.pcap`" file with this configuration results in four extracted files, which can be validated based on their contents. FTP is an ideal protocol to use to demonstrate this functionality because the file contents are transmitted in a contiguous byte stream. RPM files are also ideal to use while testing file reconstruction tools because their content can be cryptographically verified against what is expected.

References:

[1] `https://for572.com/20xat`



This page intentionally left blank.

Lab 4.2 Objectives: Using Command-Line Tools for Analysis

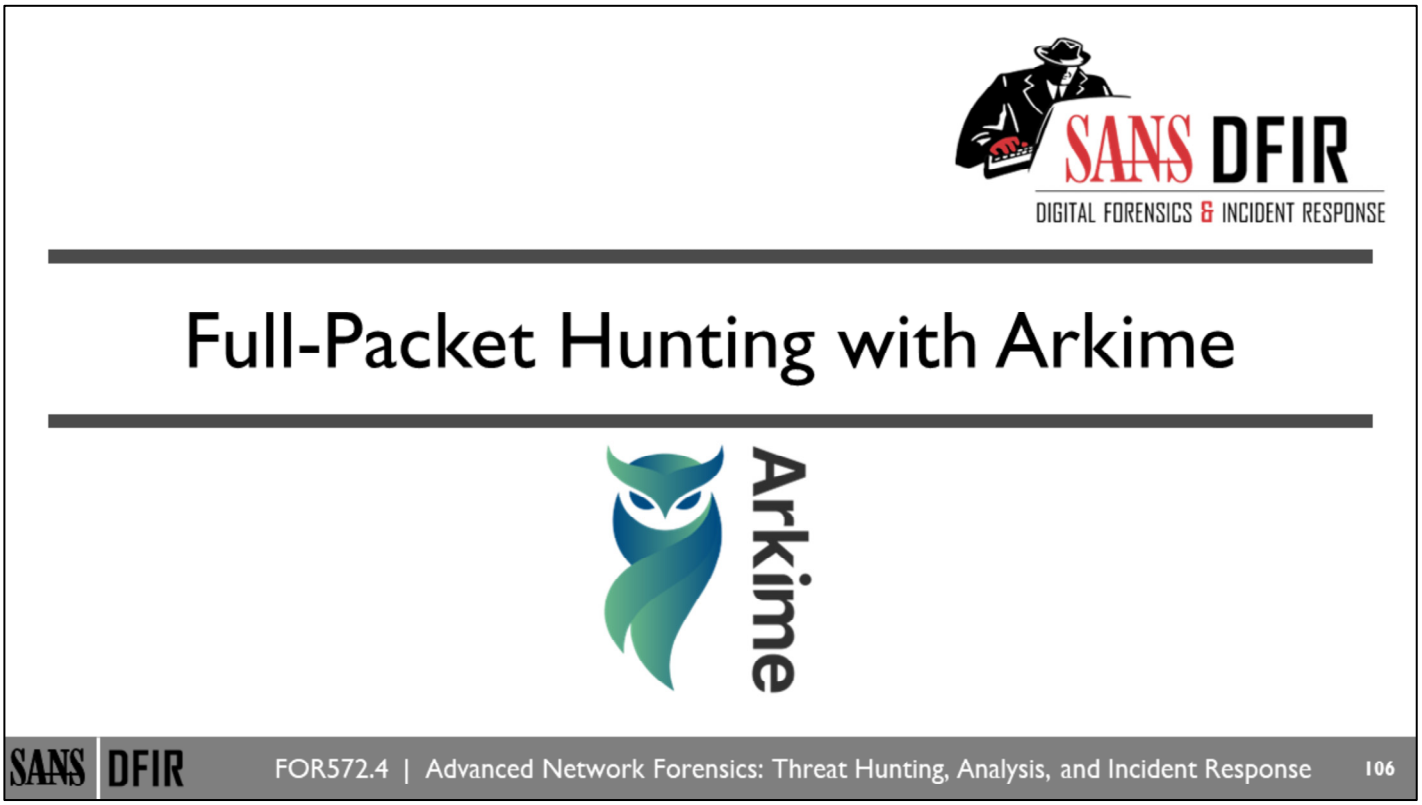
- Become familiar with command-line tools to perform common network forensic tasks
- Reconstruct transferred files at the command line
- Identify FTP transfers using `ngrep`
- Recover files using both `tcpflow` and `tcpxtract` based on network and file metadata and file content signatures

This page intentionally left blank.

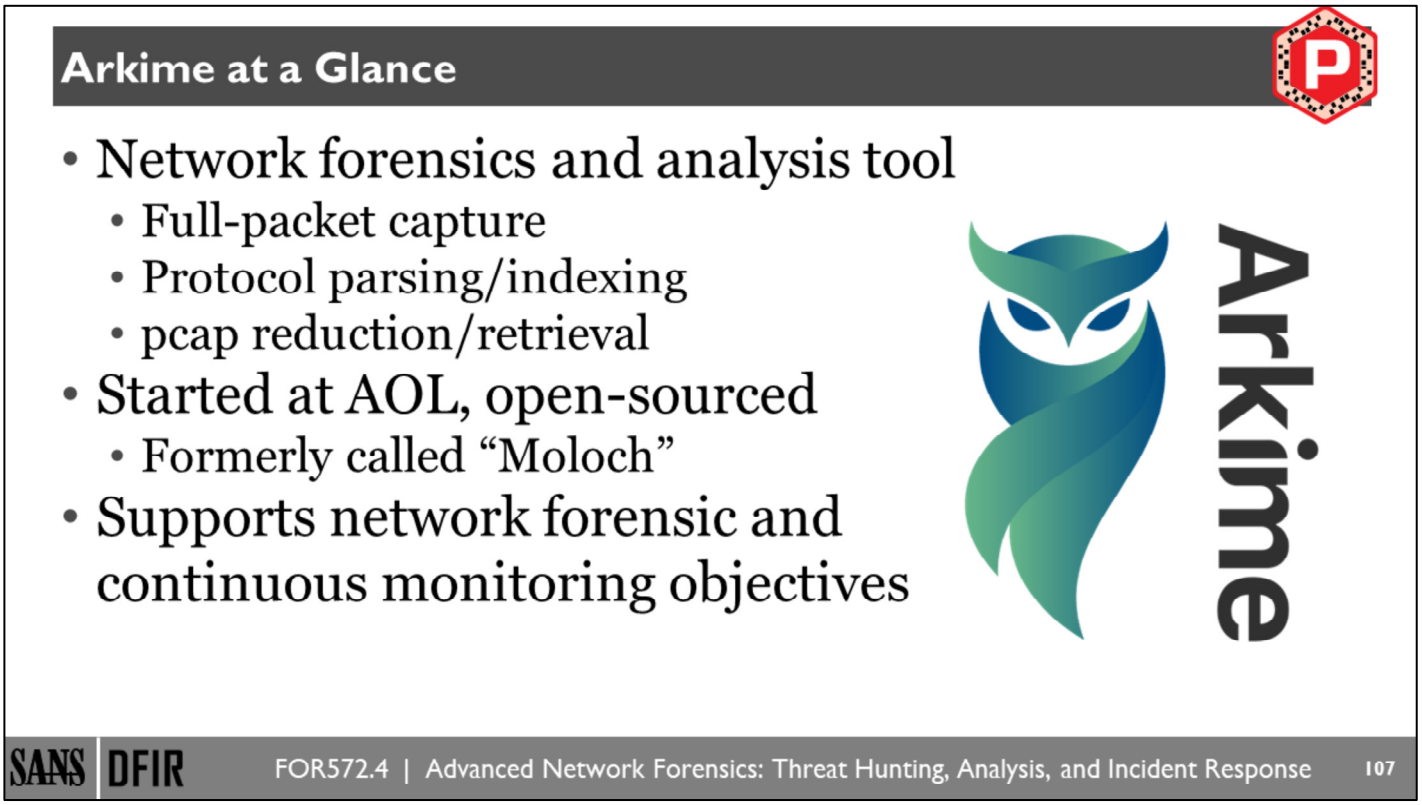
Lab 4.2 Takeaways: Using Command-Line Tools for Analysis

- Graphical tools often make analysis easier; they are sometimes infeasible
  - Not often suitable for remote analysis
  - Don't lend themselves to repeatable, scalable analysis
- Command-line tools tend to be built for a single purpose and are not all-in-one solutions
  - Analysts must often identify the best tool for each job
  - Combining such single-purpose tools can provide a very robust and efficient toolchain for the broader task
  - Scripting with these tools creates repeatable processes

This page intentionally left blank.



This page intentionally left blank.

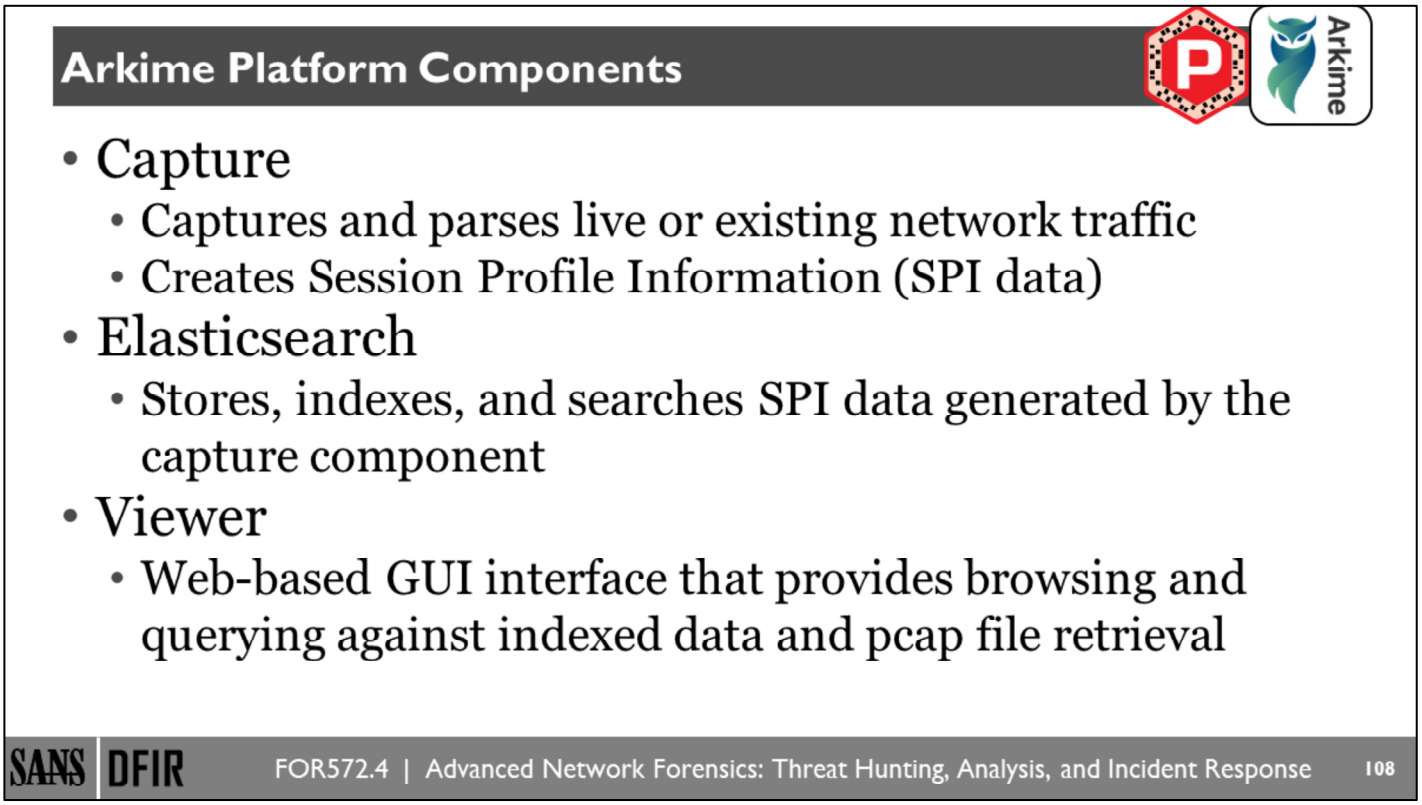


As you've seen, command-line tools can provide scalability in terms of processing large amounts of data. However, maintaining the custom scripts, linking tools and processes together, and reviewing the data can all be a bit cumbersome. If we want an easier way to search and review data, a tool like Wireshark may quickly come to mind, but it has scaling problems of its own, especially based on the volume of data being processed. There are countless commercial solutions that aim to perform efficient full content capture and indexing of network traffic, but these are often quite expensive. Fortunately, there is an excellent, scalable, free, open-source tool that rivals some commercial platforms in terms of features and usability.

Arkime is a large-scale packet-capturing, indexing, and database system that provides a simple web interface for browsing, searching, and exporting pcap files. Originally started by employees at AOL, Arkime works alongside other analytic processes to store and index all the network traffic in a standard pcap format while providing fast access and search capabilities. It works in standalone mode (as you will use during the corresponding lab) or in a fleet model with numerous systems operating together within a single architecture.

Note that Arkime was recently called "Moloch" and the developers are still in the process of updating the project naming. Be aware of this, as there are still a number of references to the Moloch name in the command-line tools and documentation. In the future, this naming will all coalesce on the "Arkime" name.

References:
https://for572.com/arkime

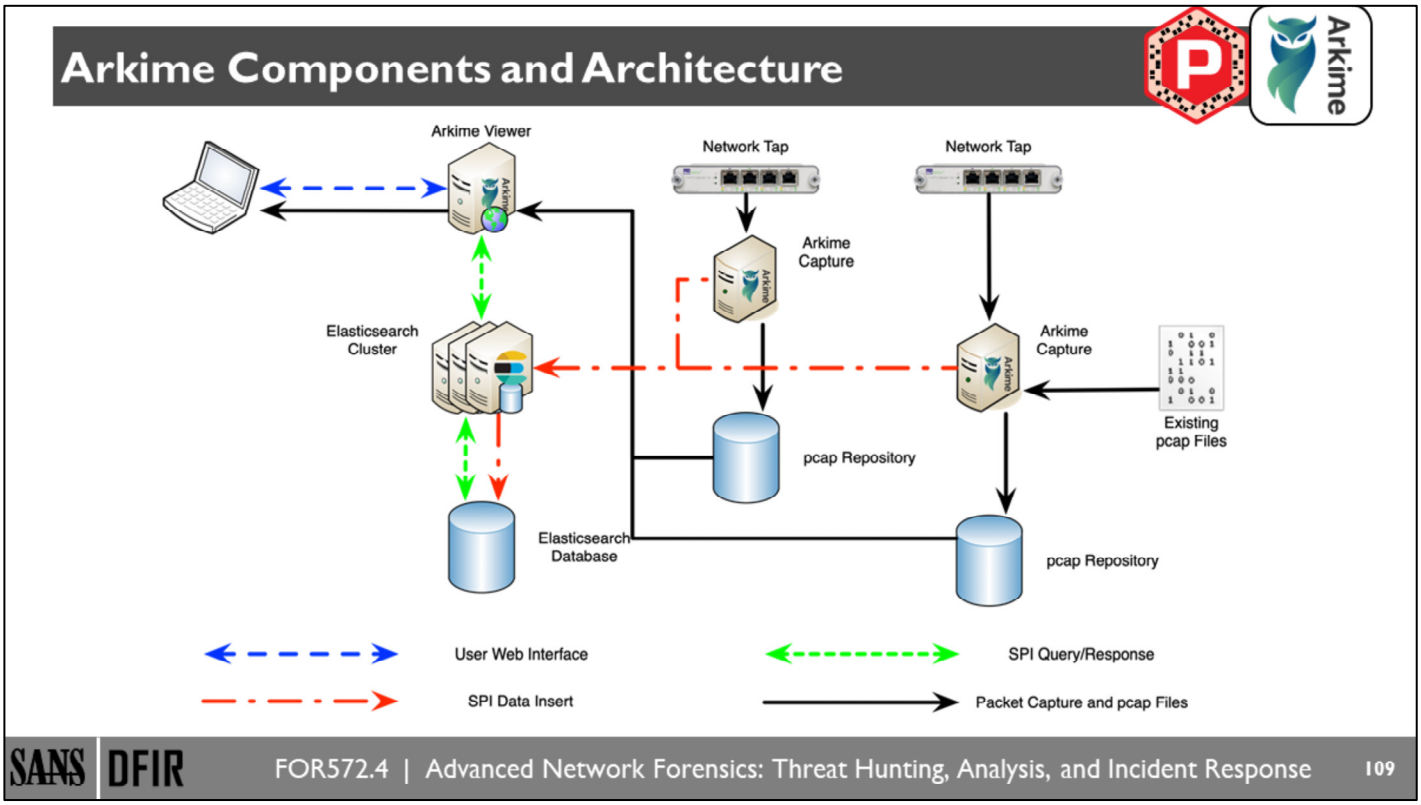


Arkime consists of three main components, which work in concert with each other to provide the platform's overall feature set.

The capture mechanism can passively listen on a network interface or read existing packet capture data. It parses the traffic, tracks Session Profile Information (SPI data), and writes the raw packets to disk for later use and extraction.

Arkime uses the Elasticsearch search and analytic engine to store SPI data records and expose them to the user through the viewer component. Because Elasticsearch is such a widely used and well-documented part of the overall platform, its use also allows third-party access and integration to the broader scope of Elasticsearch analytic capabilities.

The viewer component is Arkime's integrated web-based GUI that provides user-level query capabilities, session-level pcap data reduction and extraction, as well as an API that allows for programmatic and repeatable access to the underlying data stored in Elasticsearch.



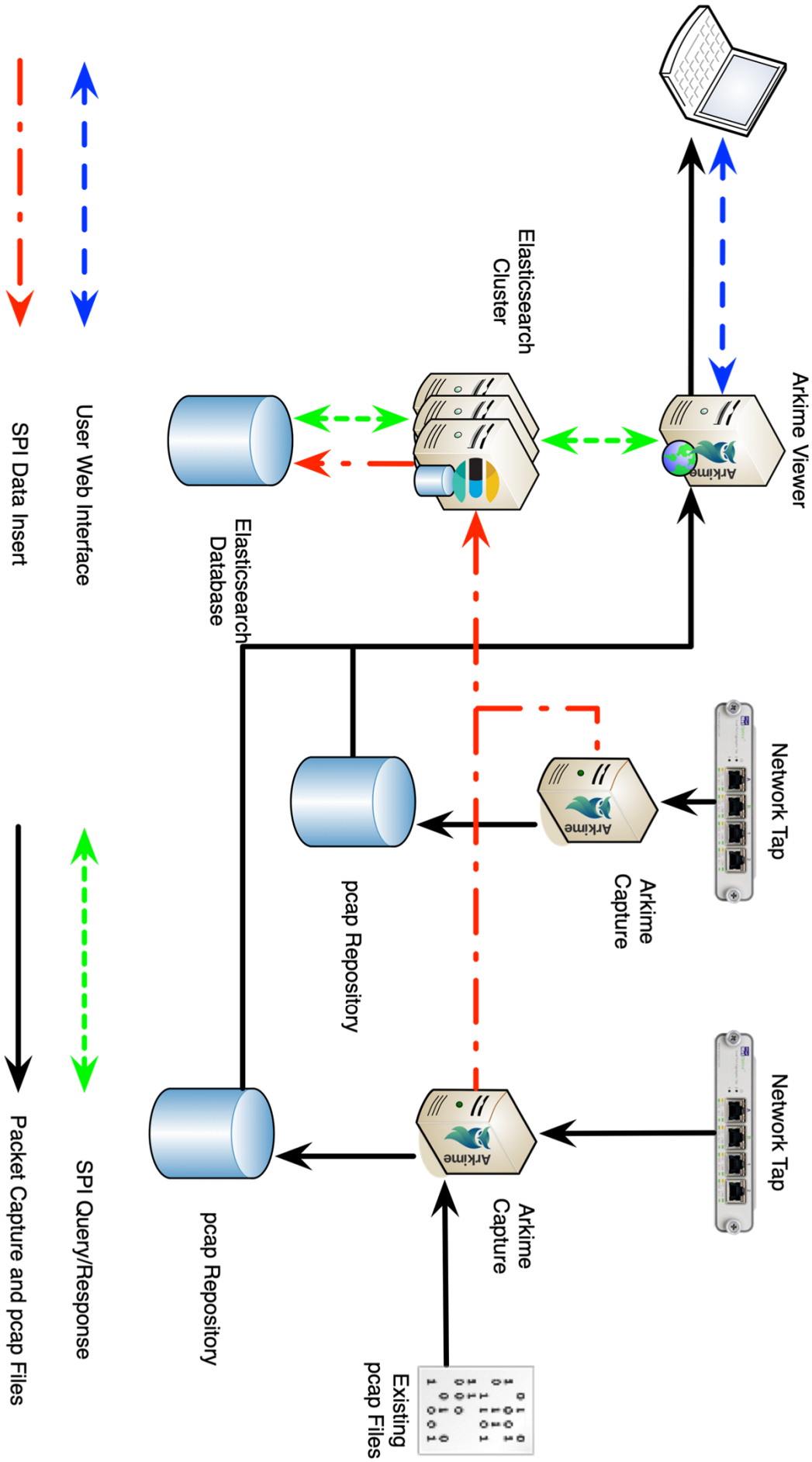
Architecturally, each component of Arkime can scale independently from the others. For smaller-scale live deployments and small-to-moderately sized pcap data loads, all components can run simultaneously on a single system. This is the model used in the FOR572 Arkime VM that you will use in the lab.

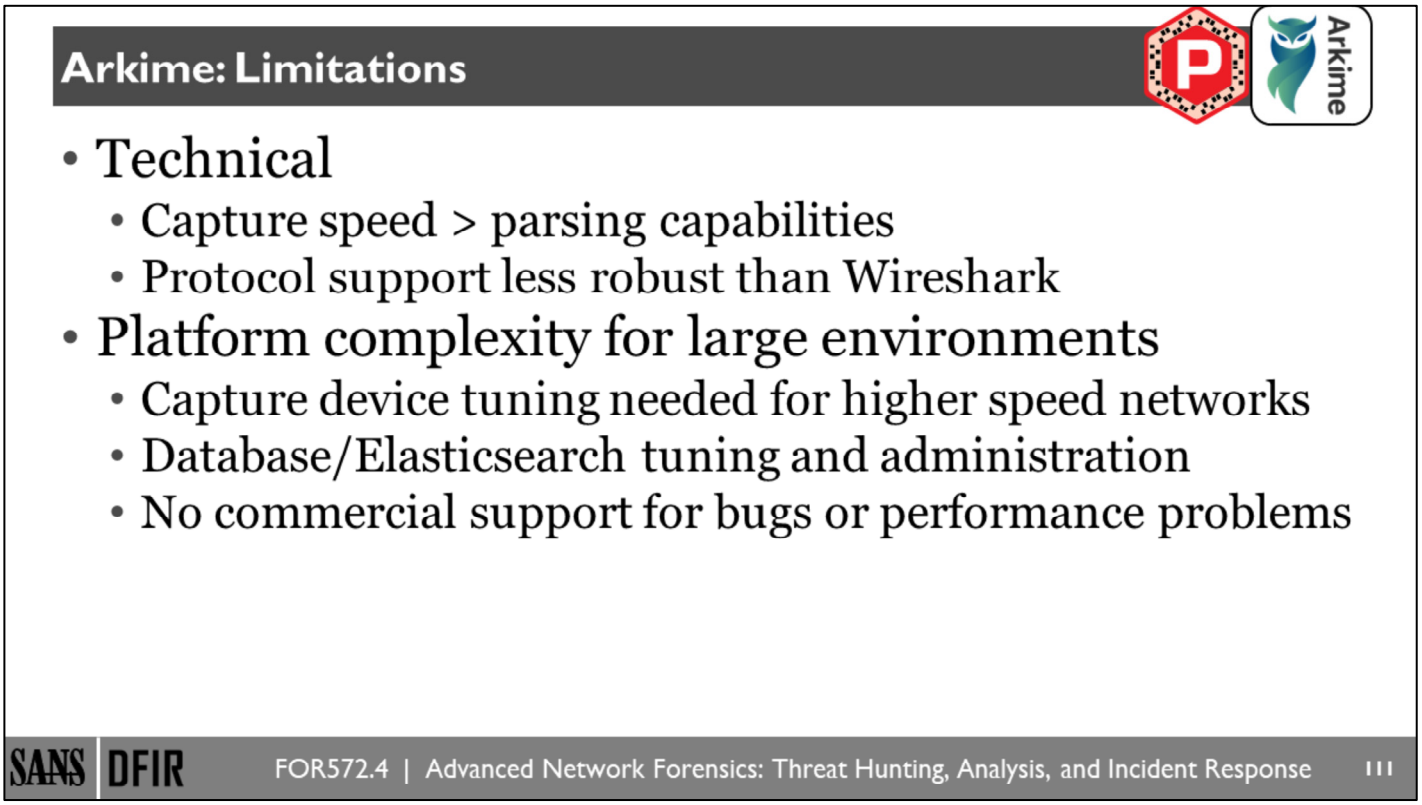
Again, for larger-scale use cases, Arkime can operate under a "fleet" model, with multiple capture platforms deployed throughout the environment, each storing full content and indexing network traffic under its own purview.

The resulting SPI data from the fleet of capture platforms would then be fed back to the Elasticsearch node or cluster for unified storage. This cluster could range from a single system to hundreds in a cluster if needed, given the horizontal scaling inherent in Elasticsearch. Since the storage backend instances can be different, Arkime's model allows for separate management of full-packet storage and SPI data – including different storage volumes, media types, data retention policies, and access controls.

Because the Arkime viewer process interacts directly with the Elasticsearch, the user can query all available SPI data from the entire capture platform fleet in one interface. This allows for more comprehensive and streamlined analysis, to include pcap session extraction for deeper examination with additional tools beyond Arkime itself.

References:
https://for572.com/8xc62





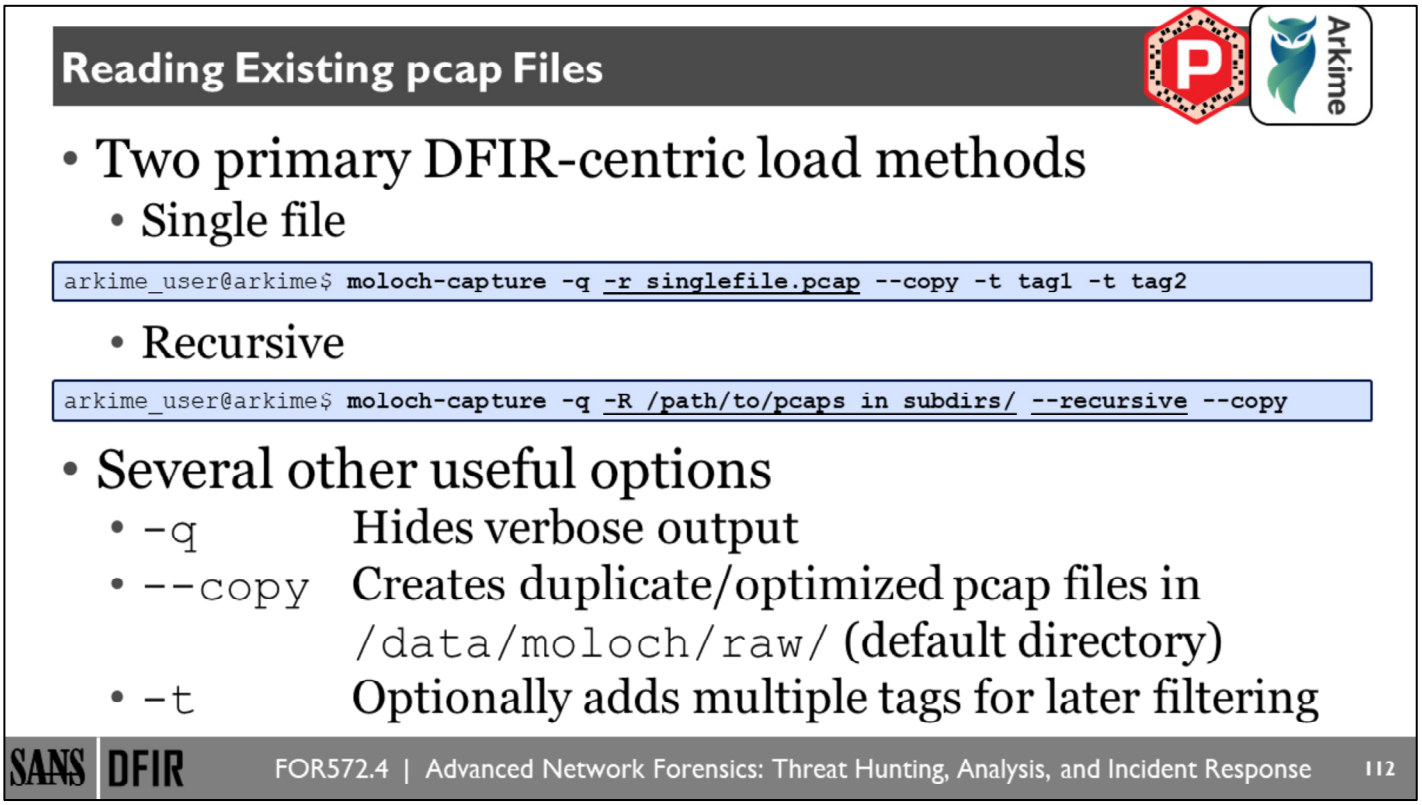
Although Arkime fills a need between expensive, out-of-the-box commercial solutions and more limited command-line tools or deep-dive utilities such as Wireshark, it is not without its shortcomings. When collecting live traffic, it is not difficult to overwhelm the protocol parsers, resulting in lost data unless the architecture is engineered to a scale that can accommodate high data rates. From the analyst's perspective, Arkime protocol support is notably sparser than that of Wireshark or many common commercial platforms. Obscure protocols are not natively supported and adding parsing support for them requires development experience.

In terms of operations, the complexity of a Arkime deployment parsing live network data increases significantly in larger and more diverse environments. Using Arkime in such an organization requires knowledge on how to tune both the capture application and the underlying Elasticsearch database. There are a number of resources available online regarding best practices for tuning Elasticsearch and troubleshooting common issues, including a free Slack team[1] where the developers and a vibrant and experienced community of users often provide unofficial support that rivals some commercial "support" offerings. However, any such support is not official and not subject to any SLA. At this time, there is no current or projected commercial support available for Arkime. Third-party consulting services from experienced administrators may be an option for some users, though.

Regardless, the value proposition for this platform is certainly noteworthy, and Arkime is a great addition to any network forensicator's toolkit. An active user community and responsive developers means new features such as these continue to be integrated with each software release. Of course, as an open-source project hosted on GitHub, community pull requests are always appreciated.

References:

[1] https://for572.com/d768b



Loading existing pcap files to Arkime can be done using several methods. For most DFIR purposes, one of the two detailed will usually suffice. The primary difference to consider is whether you need to load a single pcap file or a directory containing a large collection of them.

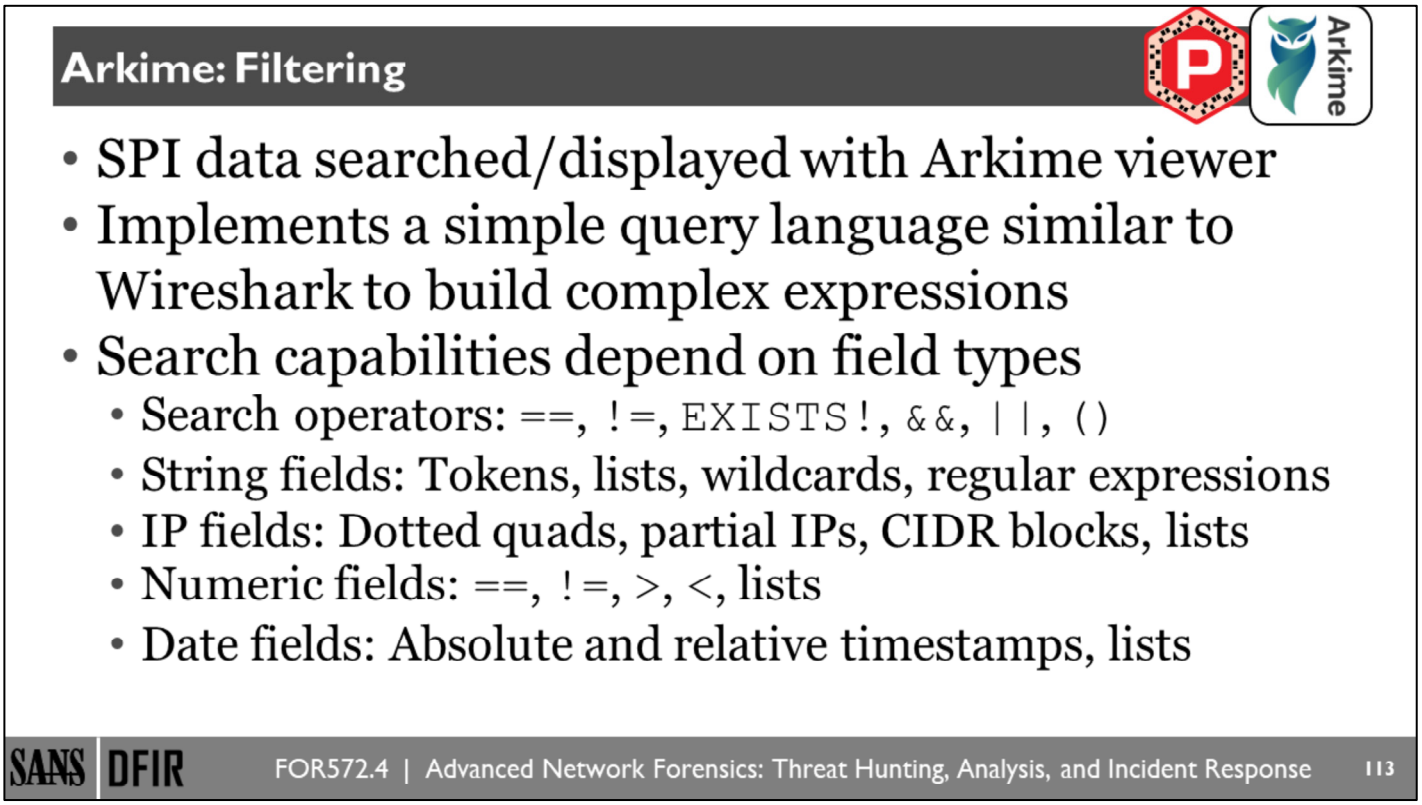
For single files, the "`moloch-capture`" command syntax uses the "`-r`" flag followed by the file to be loaded. Multiple files can be loaded with the "`-R`" flag in its place, followed by the directory to load from. However, there are a few additional considerations to keep in mind with the latter case. All files must end with a "`.pcap`" extension. Additionally, if you have a series of subdirectories containing the `*.pcap` files, you must also specify the "`--recursive`" option.

There are also several useful options that work equally for either mode of loading data.

`-q` This option hides the verbose debug output from the terminal. However, the terminal does not provide load feedback until the procedure finishes.

`--copy` This option causes Arkime to create a copy of all loaded pcap data in the pcap storage directory, which is `/data/moloch/raw/` by default. This option is recommended, as it will minimize the chances for problems related to filesystem permissions, while at the same time allowing an analyst to load from transient evidence storage devices while retaining utility of the full-packet features exposed in the viewer interface.

`-t` An analyst can add an arbitrary number of tags to a data load, which can later be used for filtering in the Arkime interface. For each desired tag, simply specify the tag text after each "`-t`" flag on the command line. This is a very convenient way to specify a capture location, case number, or other useful metadata about the file(s) being loaded.



Users can query the indexed SPI data using the Arkime viewer. Similar to Wireshark, Arkime has implemented a simple query language for building expressions. It supports logical combinations of individual queries using the "`&&`" and "`||`" operators for AND and OR, respectively. Equality is tested with the "==" operator and inequality with "`!=`". A unique case using the special pseudo-value of "`EXISTS!`" (including the exclamation point) allows the user to query records where a specific field has been identified and indexed. (For example, "`cert.issuer.cn == EXISTS!`".) All of these can be grouped with parentheses to enforce order of operation.

There are several other query types available, depending on the type of data in a given field.

String Fields

String fields can be searched in several ways and may be converted to lowercase in the index for some protocols.

- **Tokens**: It is important to recognize that Arkime often "tokenizes" strings during the indexing process. This means they are broken up based on separators in the source. Characters such as "`-`", "`/`", "`.`", and others are separators, meaning the source string "`www.sans.org`" will match a query of "`www`", "`sans`", or "`org`", as well as the original string.
- **Lists**: This is a shorthand method for doing multiple OR queries within the same field. For example, the query string "`http.uri == [www, arkime]`" is equivalent to "`http.uri == www || http.uri == arkime`".
- **Wildcards**: A "`*`" character within a query expression will match any number of arbitrary characters and a "`?`" character will match any arbitrary single character. For example, the query string "`http.uri == "www.sans.*"`" will match records with the values "`www.sans.org`" and "`www.sans.edu`", whereas "`http.uri == "www.sans.?"`" will match records with the values "`www.sans.a`" and "`www.sans.9`" but not "`www.sans.org`".
- **Regular expressions**: Regex queries can be run against text strings and must be enclosed with forward slashes (such as "`/mx[1-4]\.example\.com/`"). Note that Arkime uses the underlying Elasticsearch regex engine, which is not as fully-featured as the PCRE engine.

IP Address Fields
Queries against IP address fields can be performed using a full or partial IP address or CIDR block notation. For fields that include ports, you can use both "`1.2.3.4:80`" and "172.16.5.0`/24:53`" formats. Similar to strings, lists are also accepted in IP address fields.

Numeric Fields
In addition to the standard "==" and "`!=`" operations, Arkime also allows simple range comparisons using the ">", "<", ">=", and "<=" operators. Lists are also supported.

Date Fields
Dates can be queried with simple equality and inequality and range operators, as well as Splunk-like relative timestamps.[1]

References:
[1] `https://for572.com/xogaz`

Arkime: Filtering Examples

- DNS sessions with hostnames containing "google"

```
host.dns == *google*
```

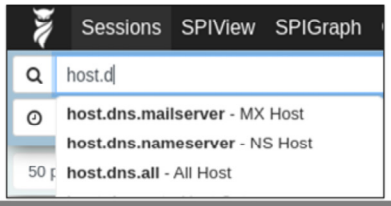
- HTTP POST sessions with Home Depot hosts

```
http.method == POST && host.http == *homedepot.com
```

- TLS sessions that don't support Diffie-Hellman

```
tls.cipher == EXISTS! && tls.cipher != *DHE*
```

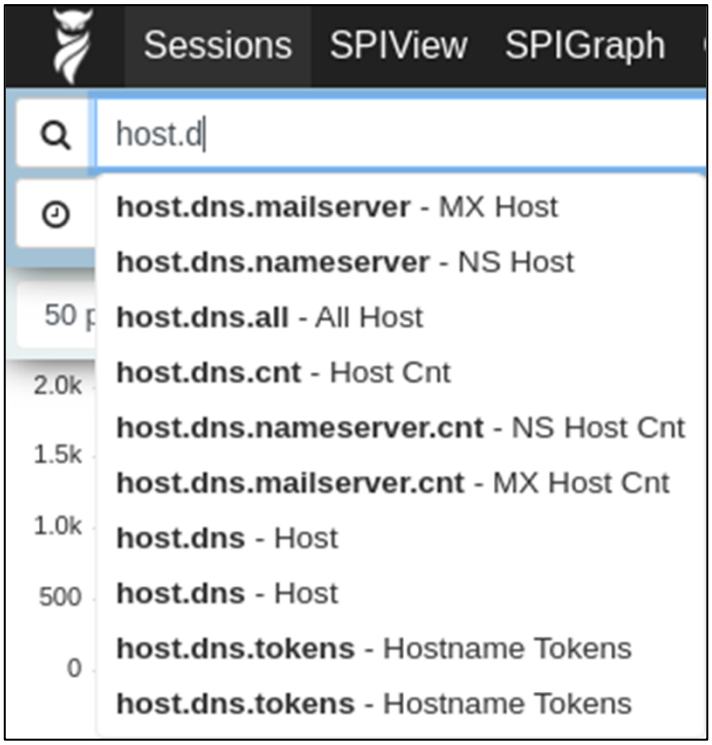
- Predictive field names while typing in search field
- Click the owl for online help

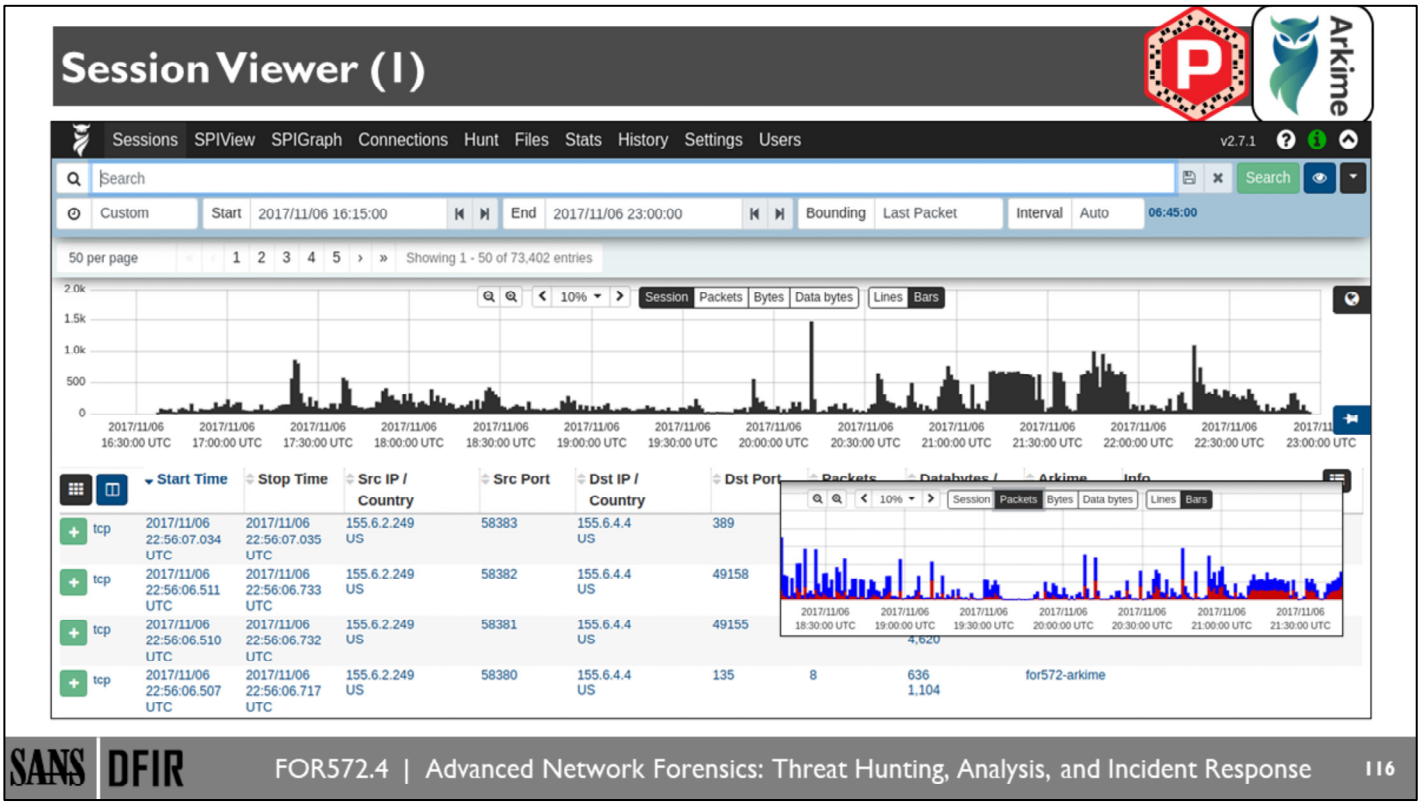


Arkime's filtering syntax provides a rich and detailed means of querying the indexed SPI data contained in the Elasticsearch database. Although this is unfortunately not compatible with the syntax from other familiar tools such as Wireshark or the libpcap BPF, it is generally regarded as intuitive enough for most analysts to effectively use.

The examples above provide a very high-level idea of the types of search that can be performed. One very useful feature within the Arkime interface is that field names are predictively displayed while typing in the search bar. As shown in the screenshot, typing "host" will result in a dynamic dropdown list of fields that contain this string. This kind of feature can make learning new syntaxes much easier than studying reference material.

Another invaluable feature is that each installation includes its own bundled help. This can be accessed by clicking the Arkime owl icon in the upper left-hand corner of the viewer interface.

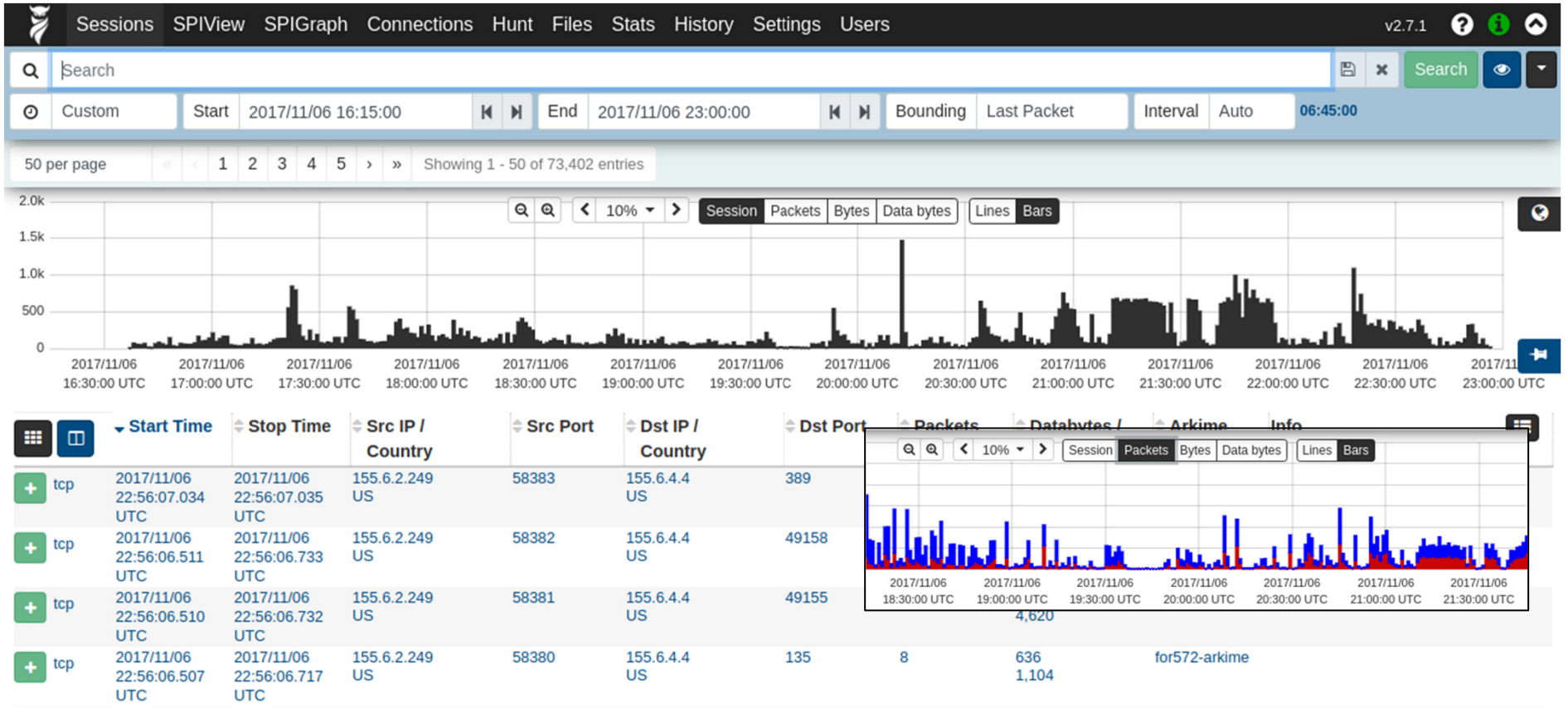


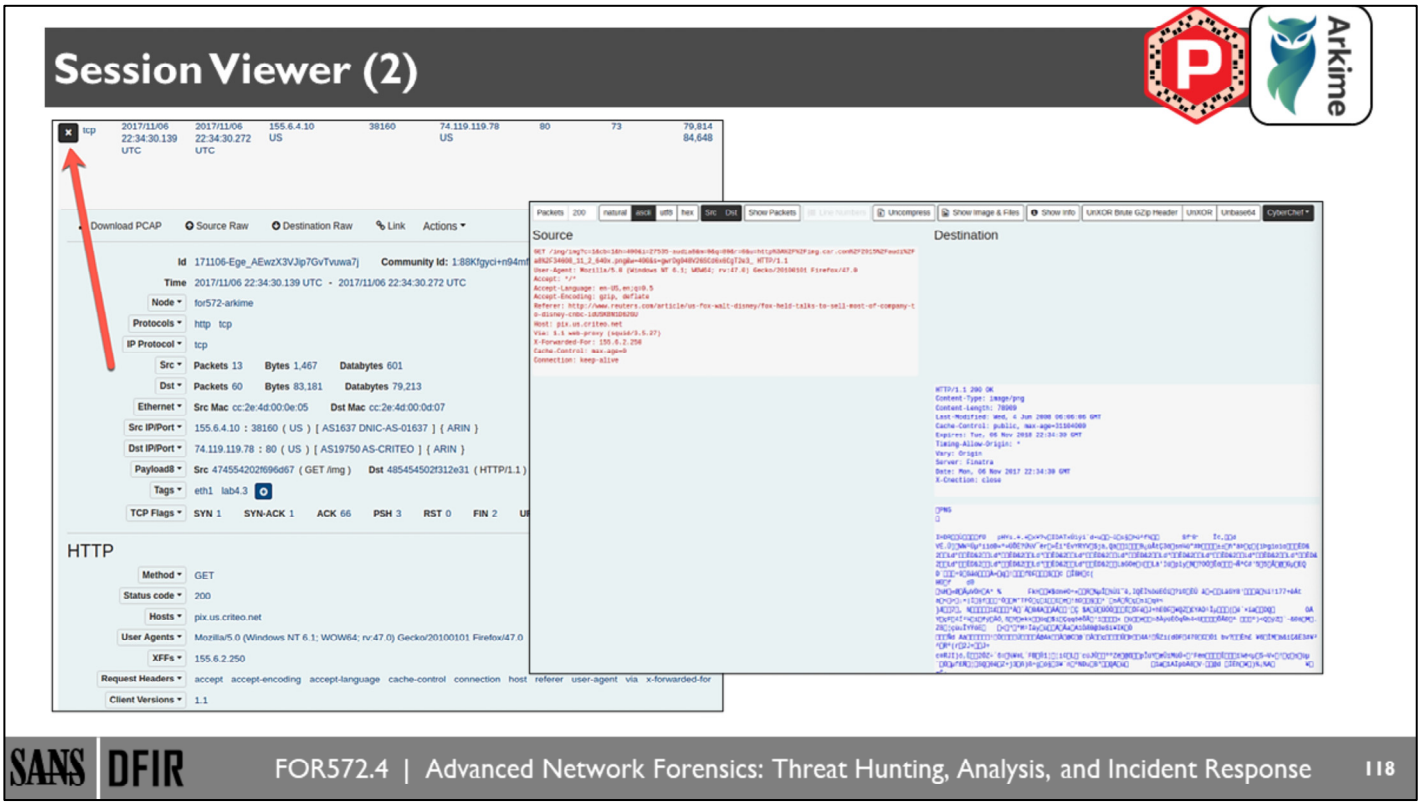


The Arkime Viewer interface first presents each session in a row, along with a typical graph reflecting sessions, packets, or bytes over time. The packets and bytes views depict directionality in red and blue colors, consistent with the client-to-server and server-to-client keys popularized in Wireshark.

Note that each row is a session, as Arkime coalesces the two sides of a conversation to a single record. Since this means the SPI data for each session is also associated in this manner, we will be able to simultaneously search based on client-based and/or server-based SPI data, resulting in matches that consist of the bidirectional conversations as well. This is different from Wireshark, which generally provides per-packet matching, or other typically unidirectional tools and data sets such as NetFlow.

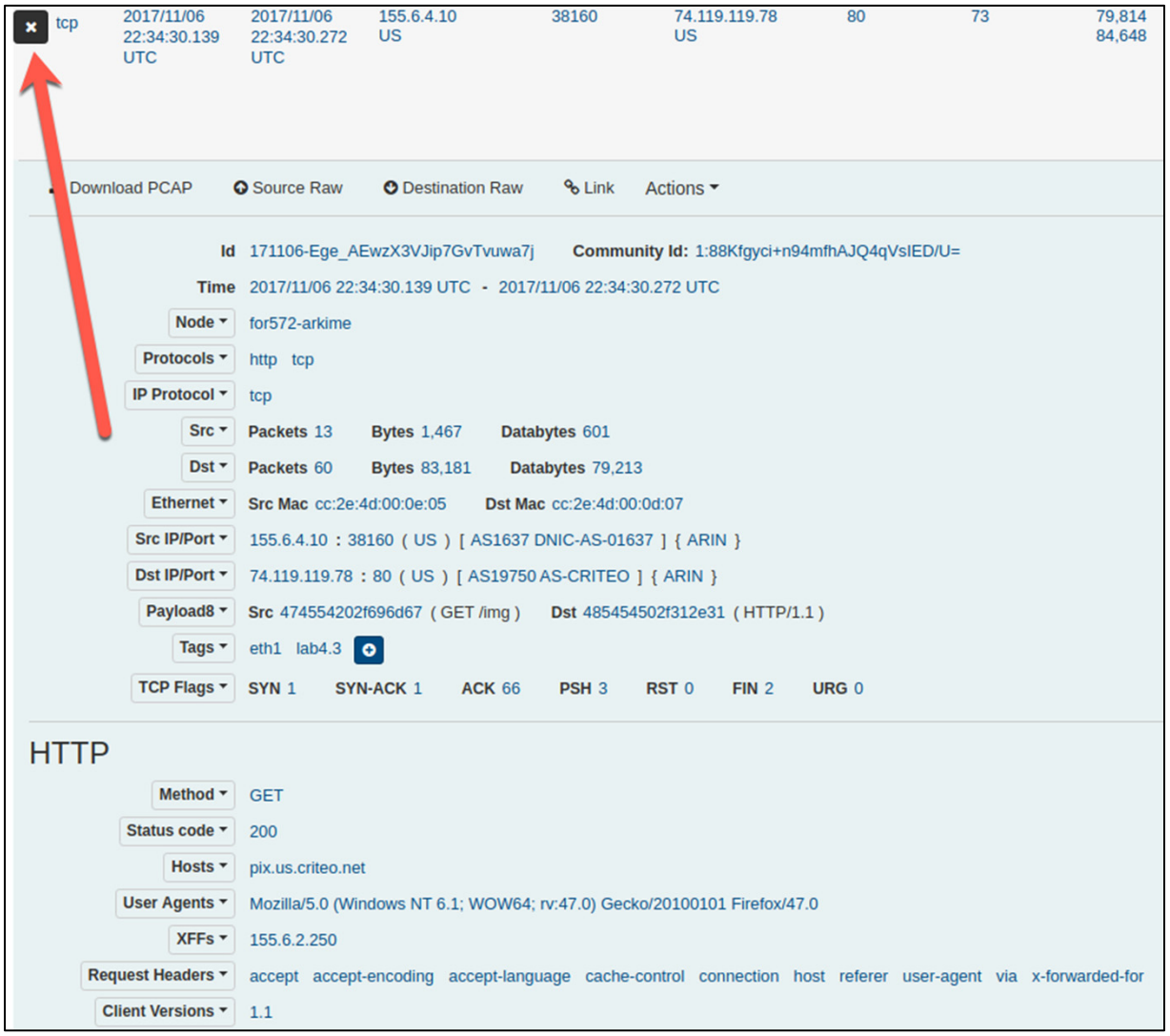
As would be expected in an interface such as this, the time-based view allows selecting a smaller window of the displayed data. Each session's row can be expanded to show the SPI data it contains by clicking the green "+" icon to the far left of the row.

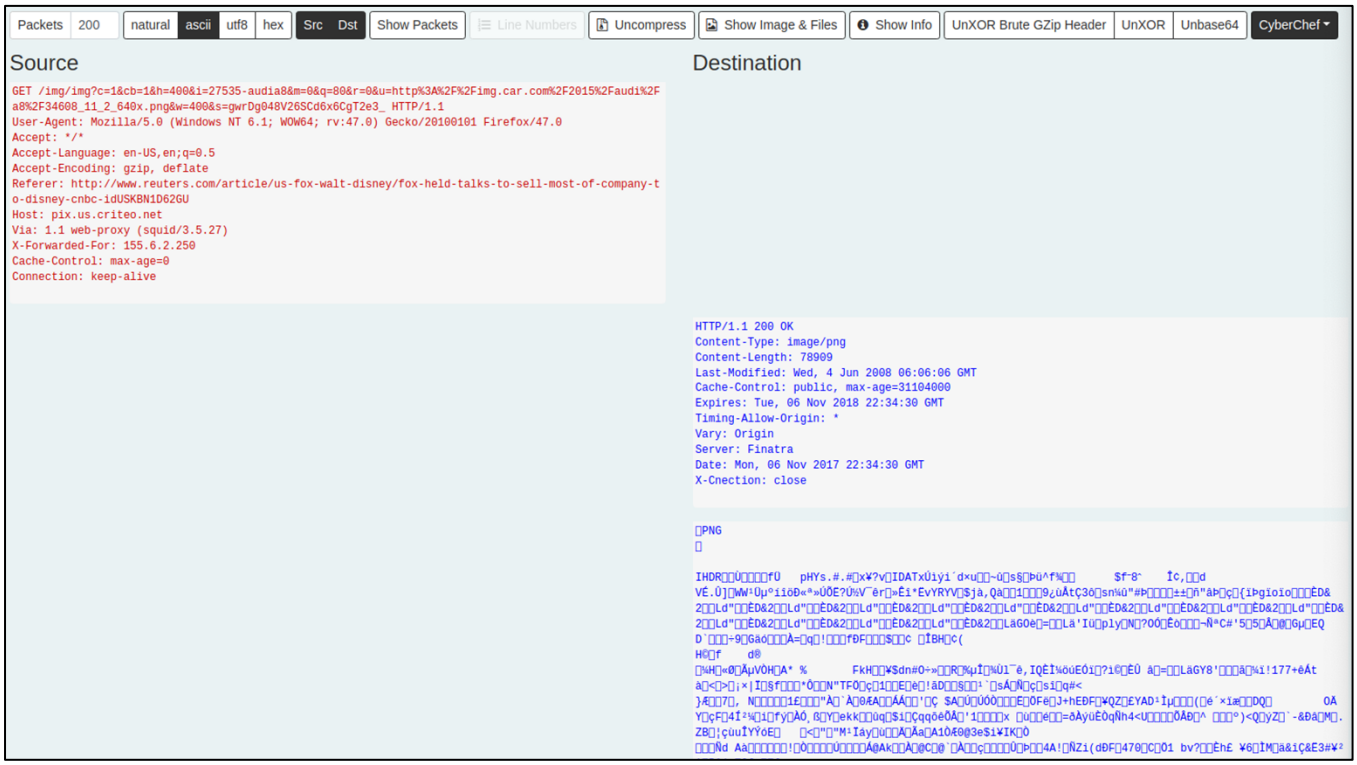


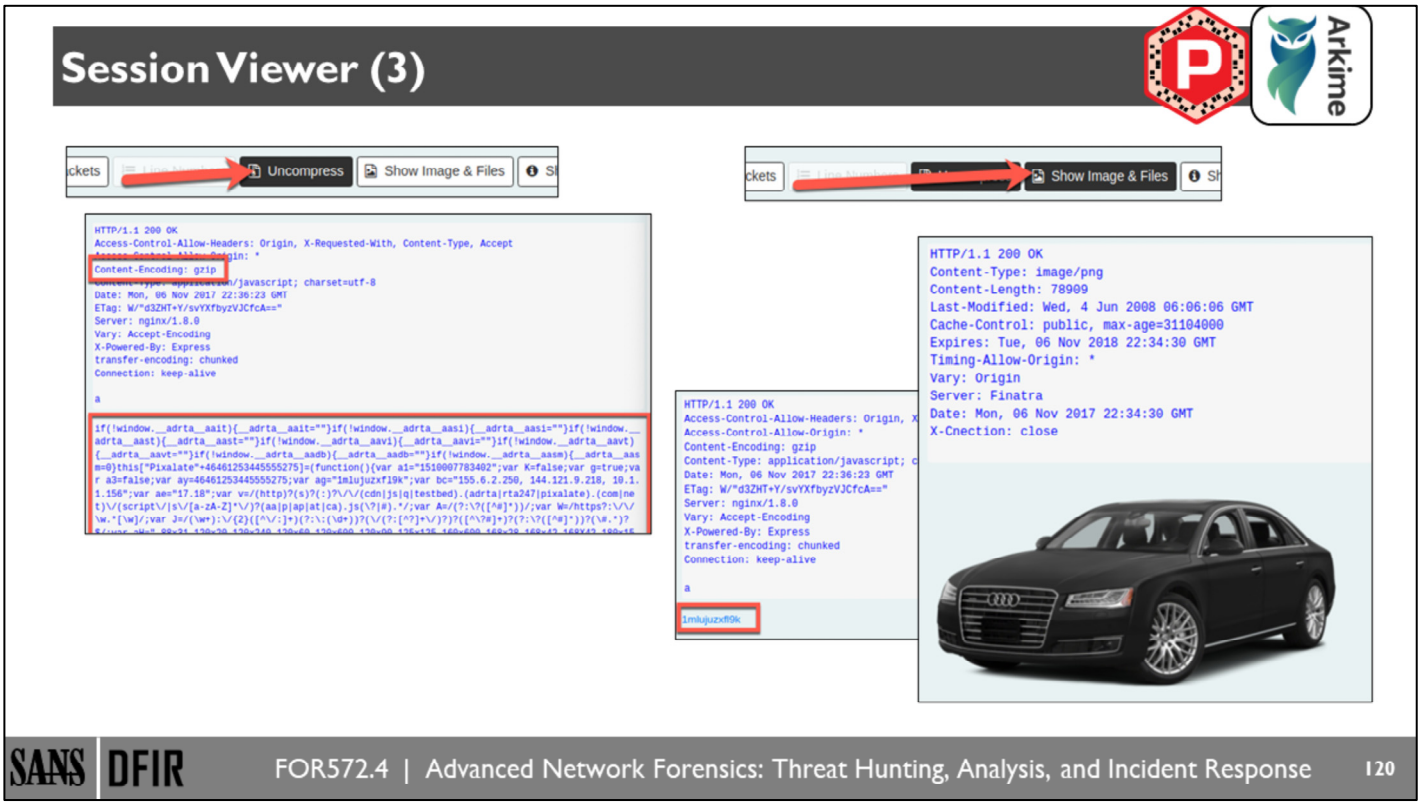


After "unrolling" one of the sessions, Arkime presents all indexed SPI data as depicted above. Each of these fields can be clicked, adding to any existing filter statement—a user-friendly way to explore the available SPI data. Some of the available fields are generic Layer-3 and Layer-4 data. Those higher-layer protocols that Arkime can parse such as DNS, HTTP, SMB, and TLS provide many additional fields. Keep in mind that the SPI data is generated at the time the data is loaded, then stored to Elasticsearch. This provides fast and easy searching in the Arkime Viewer interface because of the Elasticsearch backend. It also allows searching from SPI data even if the source pcap data is no longer available. SPI data fields displayed can be added and removed from the display by the user as well.

If the source pcap data is available (e.g. has not been expired from the collection, is not protected by filesystem permissions, etc.), the client and server sides of the conversation will be displayed in Wireshark-consistent colors and a variety of display options. This expanded session view also allows the user to download a pcap file of just the displayed session, enabling further processing with any other pcap-capable tools. This is a very efficient way to perform interactive and visually-based data reduction.

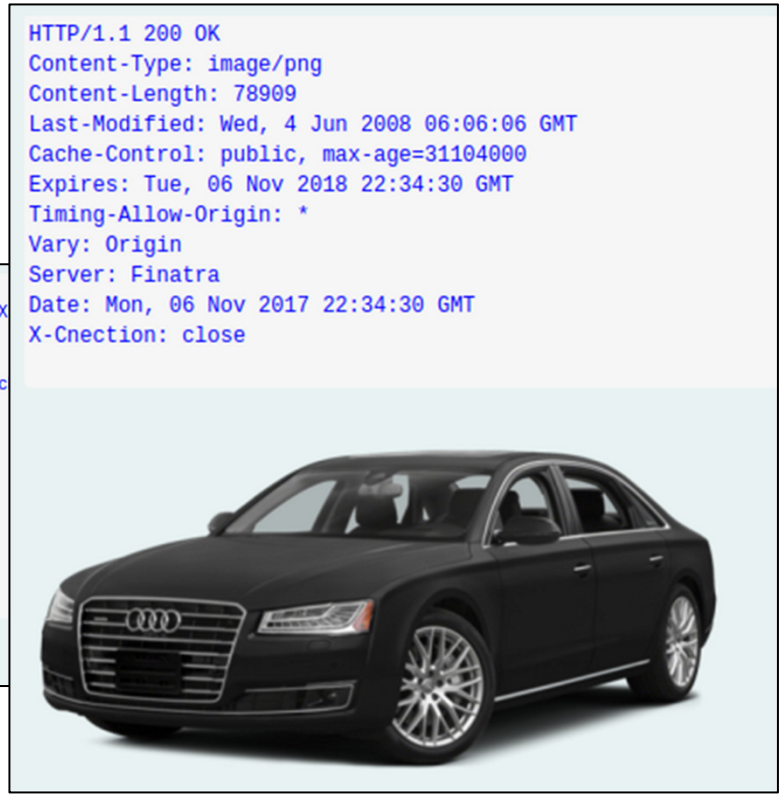


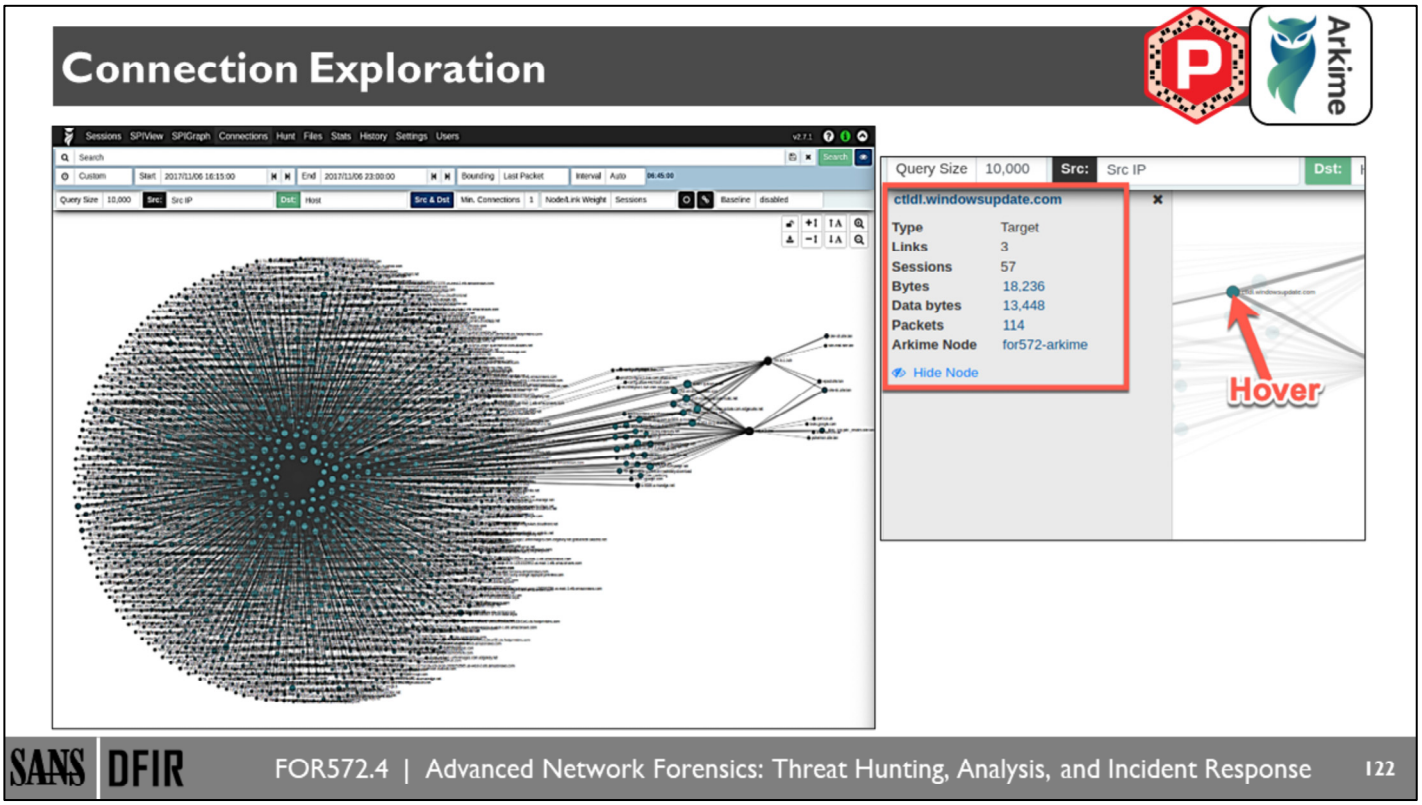




Arkime's on-platform data transforms can greatly accelerate analysis. For example, the user can select whether or not to attempt to decompress and/or decode the content directly within the Arkime Viewer interface. The screenshot on the left reflects the same session as displayed on the previous slide—a gzip-compressed HTTP response. Simply clicking the "Uncompress" button will seamlessly show the decoded content right in the browser.

The user can also select whether images and files should be decapsulated and decoded. Clicking the "Show Images & Files" button causes Arkime to display images inline with other information—such as HTTP headers, as shown on the first screenshot on the right. If the file is not an image, a link is provided, allowing the user to download that reconstructed file. Arkime forces a "`.pellet`" file extension on all file downloads to prevent accidental execution or opening malicious content in an application.





Another very useful feature in the Arkime viewer interface is the "Connections" tab. This tab allows the user to visualize the values from two series of SPI data fields from any available fields of data. In this manner, the user can explore centers of gravity between different items within the source data.

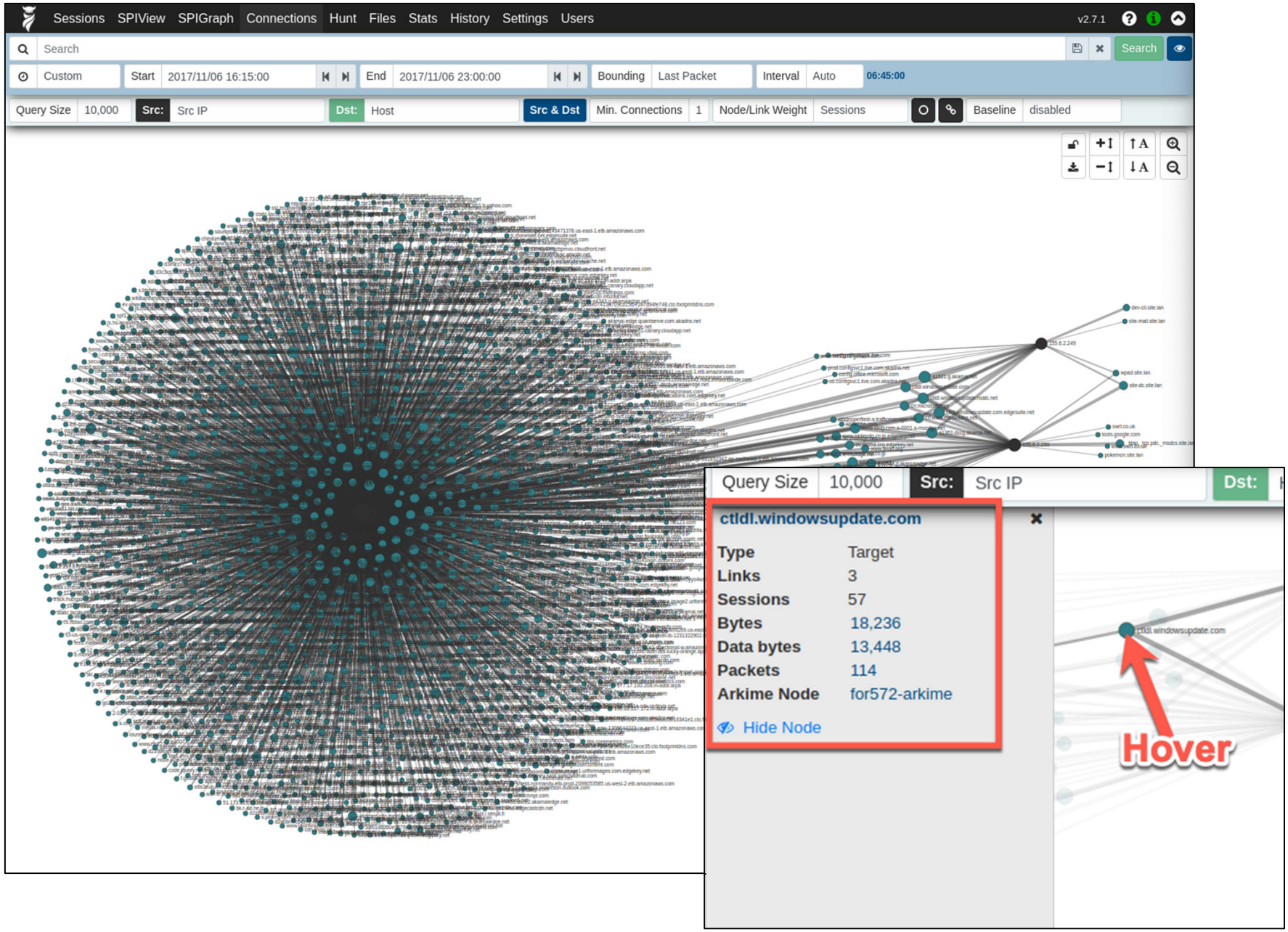
In the example above, the Connections tab reflects the association between source IP address and queried DNS hostname. This creates a clear visual pattern of which hostnames were commonly queried by multiple client IP addresses. The handful of hostnames associated to multiple IP addresses reflect more widely queried hostnames, for example.

This view is both analytic and interactive. The user can interact with the visualized connections by hovering the mouse over an individual node to get more information about it, including link, session, byte, and packet counts for the node. The user can drag nodes around the canvas to more comprehensively explore the presented data.

The analyst can build on the query string through the "AND" and "OR" Expression options.

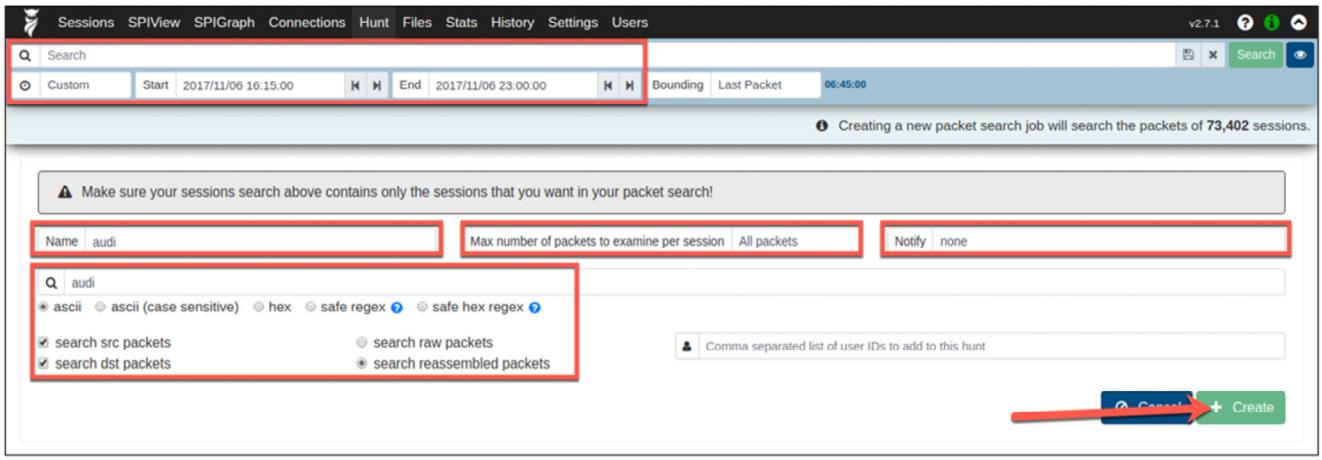
Remember, every available field can be used for the data series, so a wide range of useful associations can be explored, including but not by any means limited to:

- Source IP address and Destination IP address (`ip.src`, `ip.dst`)
- Source IP address and HTTP hostname (`ip.src`, `host.http`)
- SMB username and SMB filename (`smb.user`, `smb.fn`)



Packet Hunts: Raw Packet/Session Searches (1)

- The "Hunt" feature allows raw packet searches
  - Granular options for how to apply the search

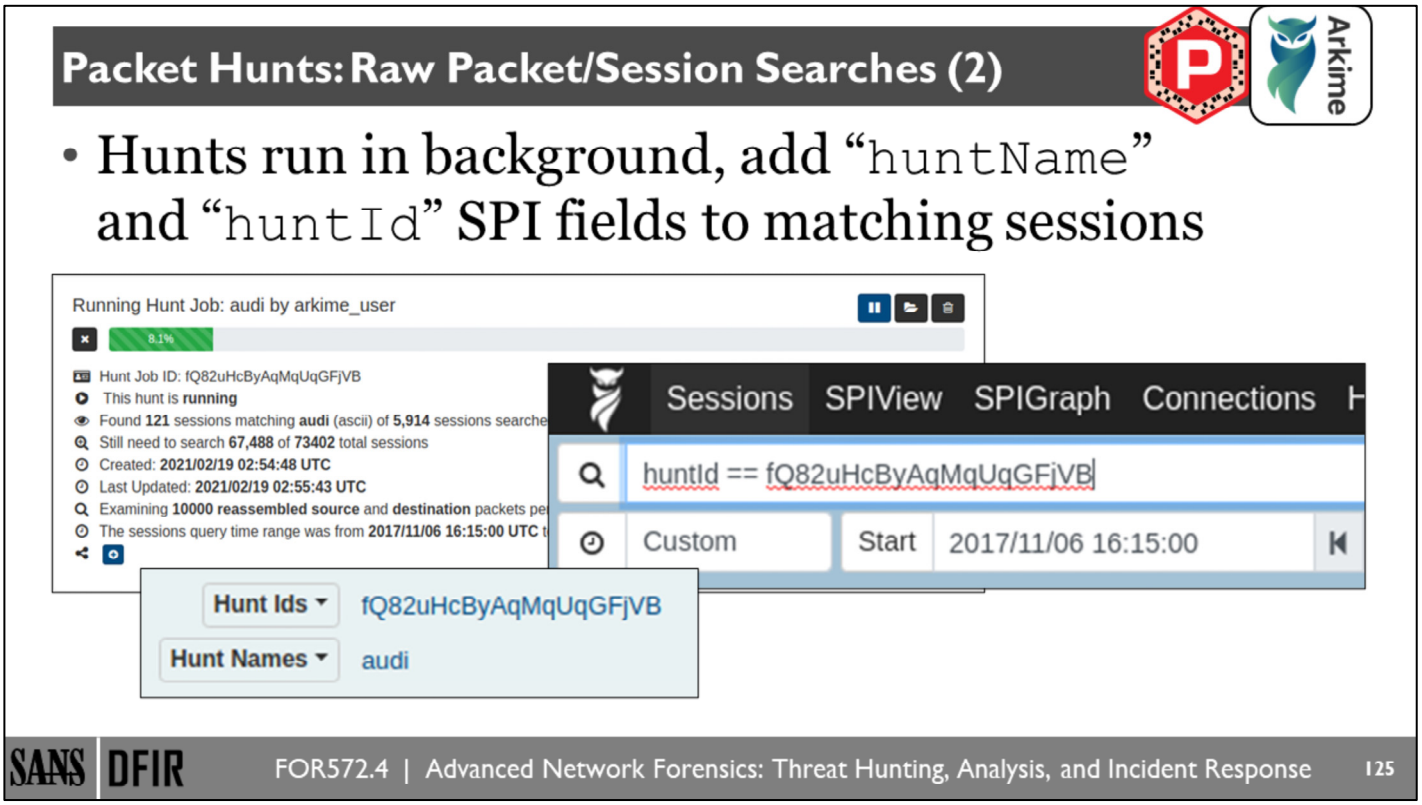


A recently added Arkime feature is the "Hunt" capability, maintained under the tab with the same name.

A Arkime hunt allows the analyst to specify a search task that will look through a specified set of results as identified on the "Sessions" tab, then search the raw pcap content—provided, of course, that those original pcap files are still available to the Arkime viewer interface.

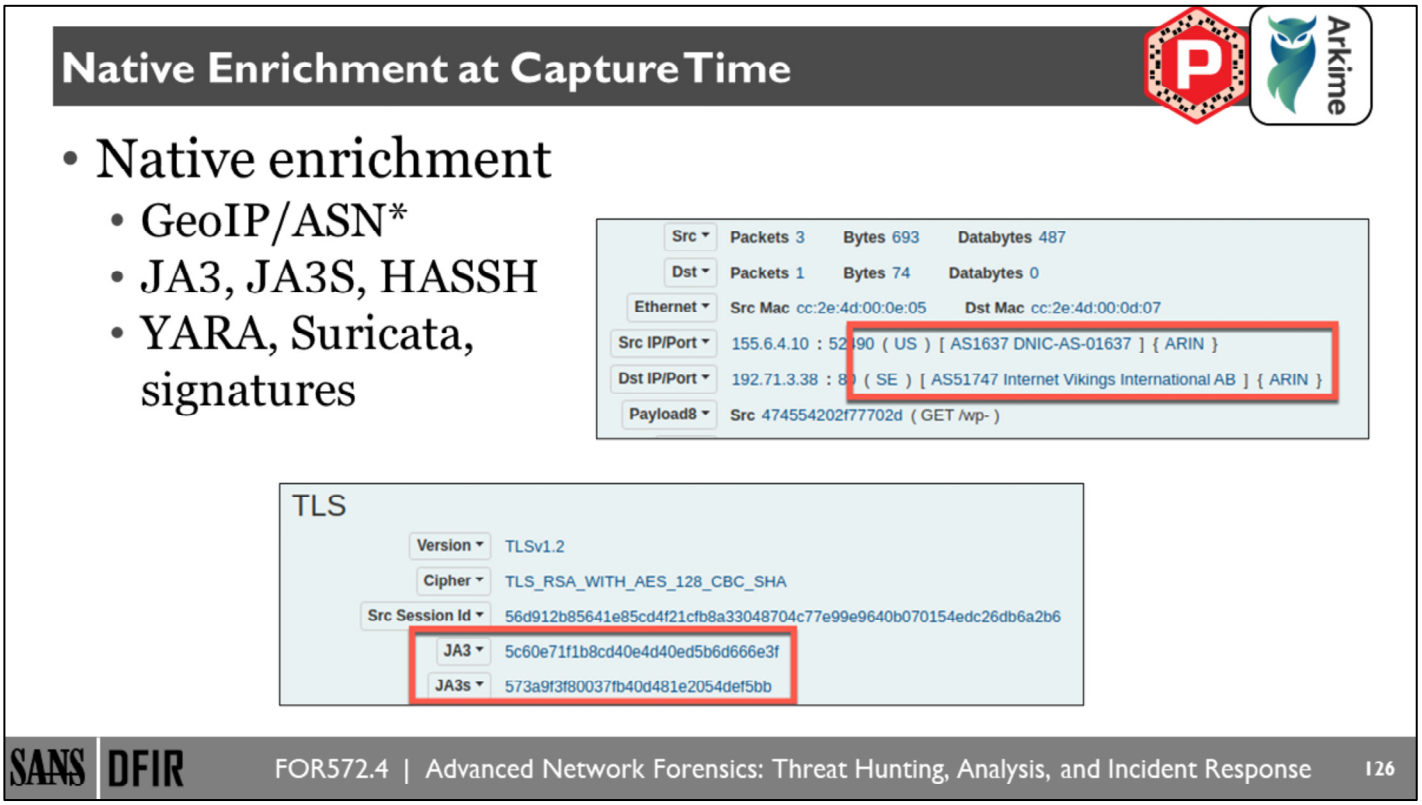
A number of parameters can be specified for each hunt task, including the number of packets in each session to search before moving onto the next, the matching method to use, directionality of packets to be examined, and whether to reassemble packets before searching or to look on a packet-by-packet basis.

When created, the hunt will be queued to run in the background.



When a match is found, Arkime adds two fields to the matching session. The "`huntName`" and "`huntId`" fields are stored to the session's SPI data, allowing fast searching after the hunt is completed and even if the source pcap data has been deleted. Since these fields are part of the SPI data, they will remain searchable even if the source pcap is deleted, or if the hunt job itself is deleted from the interface.

Saved hunts can also be re-run against different sets of traffic sessions, simplifying common or otherwise repetitive searches through large volumes of source traffic.



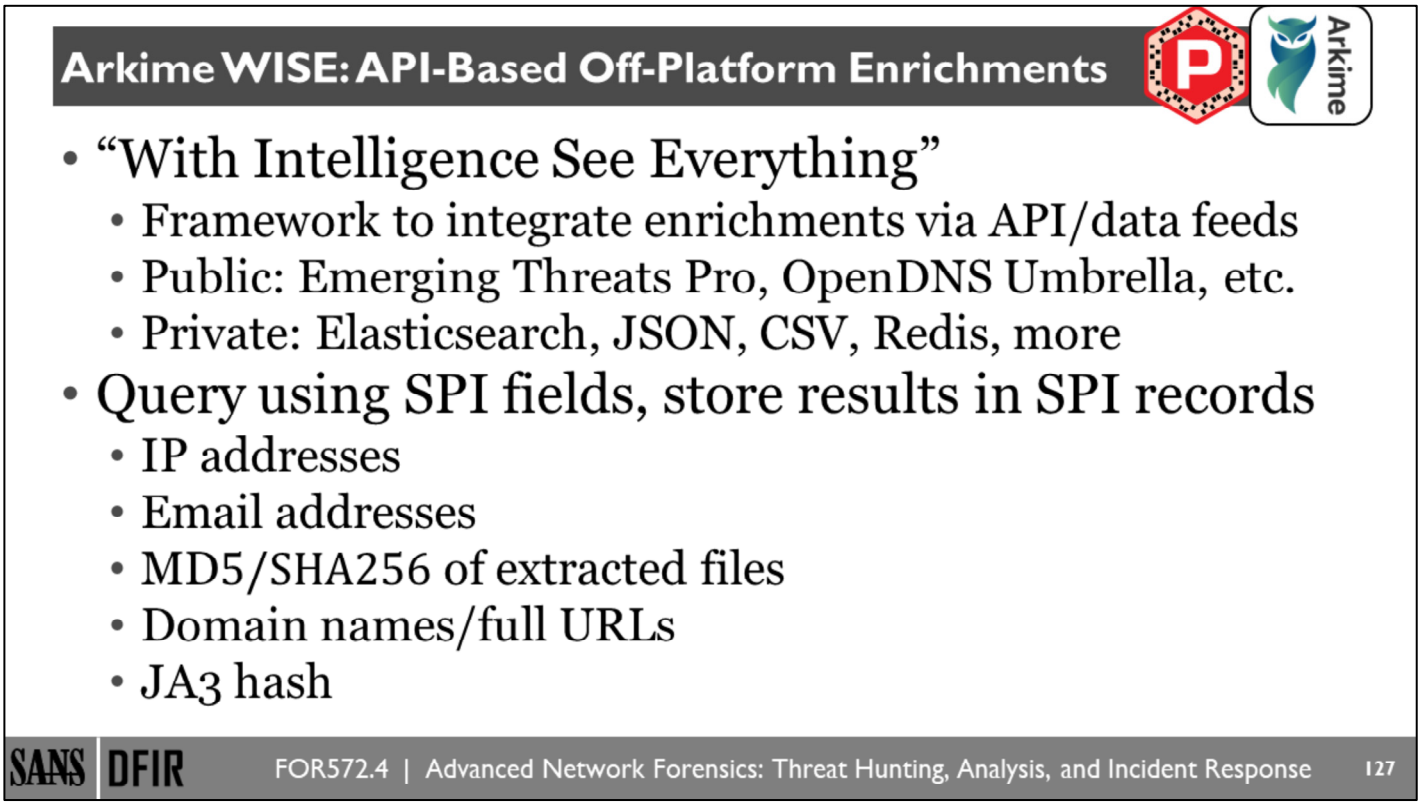
There are many data enrichment fields built into Arkime's core codebase. Virtually limitless additional enrichments can be performed using external data sources as well.

New native enrichments are being integrated with each new code release. One of the original examples of this is the addition of GeoIP and Autonomous System lookups, both through local copies of MaxMind GeoIP database files. A newer addition extends the TLS parser to include calculation of the JA3 fingerprinting hash, which can be used to profile and identify TLS client software libraries based on their advertised TLS negotiation preferences. Later modifications resulted in the addition of the JA3S fingerprint, which profiles the TLS server in the same general manner as the JA3 addresses the client, and the HASSH fingerprint, which carried these same profiling concepts to SSH traffic.

Due to recent changes in the licensing of MaxMind's databases, they cannot be distributed with the Arkime software or the platform distributed with your course materials. To use geolocation and ASN features in the Arkime VM distributed with your course materials, the user must create a free MaxMind account and license key and then use the `geoip_bootstrap.sh` script to load and activate the databases. This script can also enable automatic periodic updates of the MaxMind GeoIP databases.

When the Yara pattern-matching engine[1] is run alongside Arkime (on the same system), Yara's results can be stored with the SPI data for the session where the pattern was matched. While traditionally focused on finding patterns in binary files, the Yara engine can also search network traffic. When a signature is matched, the Yara rule number is added to the session's SPI data. Suricata[2] alerts based on their signatures can also be added when using that platform.

Resources:
[1] https://for572.com/yara
[2] https://for572.com/suricata



The Arkime developers also understand the need to integrate outside intelligence feeds and other data lookups into the tool, since use cases and available data sources vary widely between organizations and teams. The framework for doing this in Arkime is called "WISE", which stands for "With Intelligence See Everything".

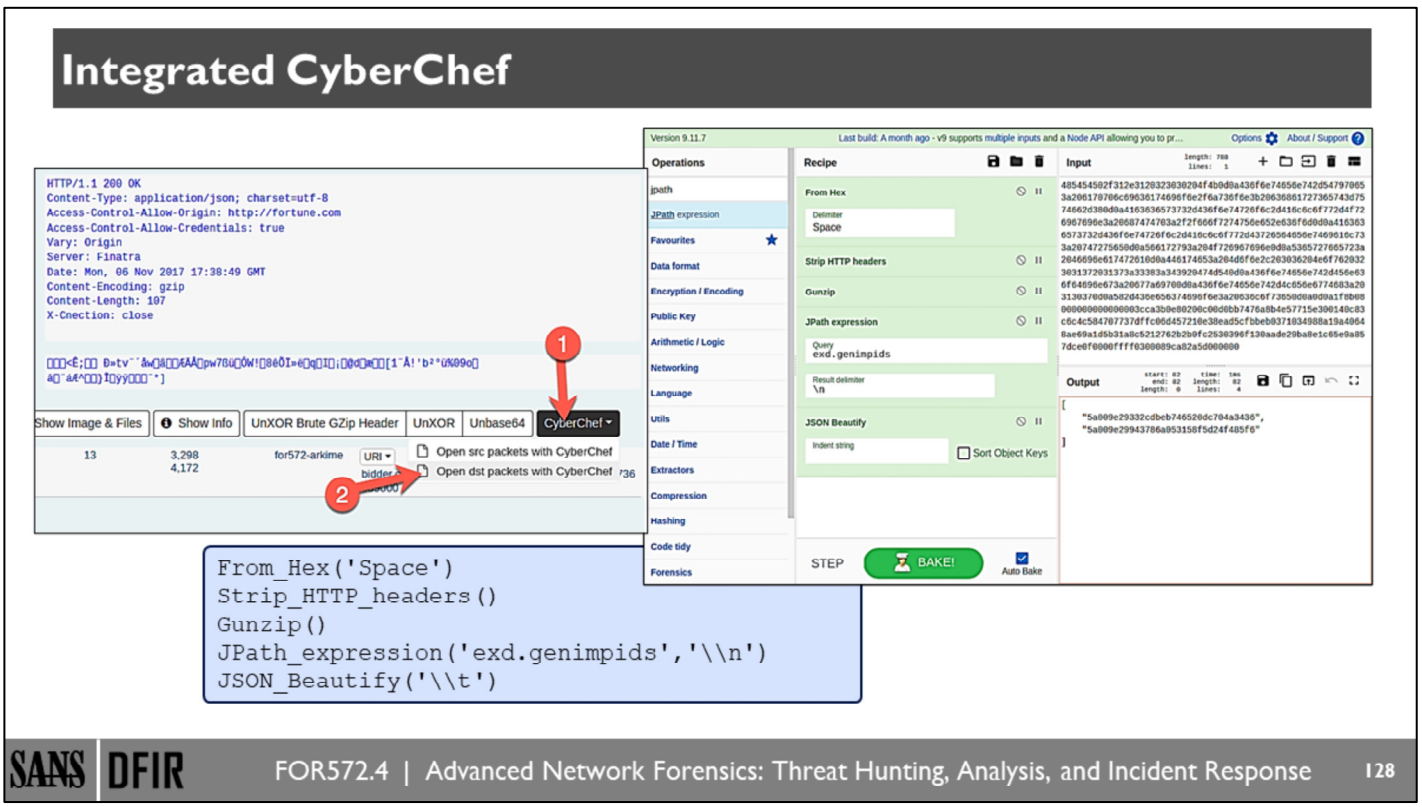
While capturing live data or parsing existing pcap files, Arkime can be configured to use the WISE subsystem, allowing it to obtain additional information about observed values. This may include looking up SPI data such as IP addresses, email addresses, domain and hostnames, MD5 or SHA256 hashes of extracted files, URLs and hostnames against any number of external, API-based enrichment services. WISE lookups can use both free and commercial data sources, and the results are stored in the Elasticsearch database document for the corresponding session.

Arkime's WISE subsystem also allows querying private data sets, such as other Elasticsearch documents, Redis or LDAP databases, local JSON or CSV files, and more. This feature allows the administrator to perform valuable enrichment lookups without sending potentially sensitive information to public resources.

The latest version of available sources is maintained on the Arkime Wiki.[1]

References:

[1] https://for572.com/-021a



Perhaps one of the most exciting recent additions to the Arkime interface has been CyberChef[1]. This offline decoding spell book was open-sourced by the British GCHQ and provides a wealth of capabilities for the forensic analyst, penetration tester, or general data wrangler. The user can select an arbitrary number of encoding/decoding components and implement them in whatever order is needed. The results are displayed live in the interface, providing immediate feedback. The resulting steps, or "recipes", can then be shared using a custom syntax, JSON, or a URL. For example, the recipe shown in the graphical interface above could be represented as shown on the lower left of the slide or as shown in the box at right:

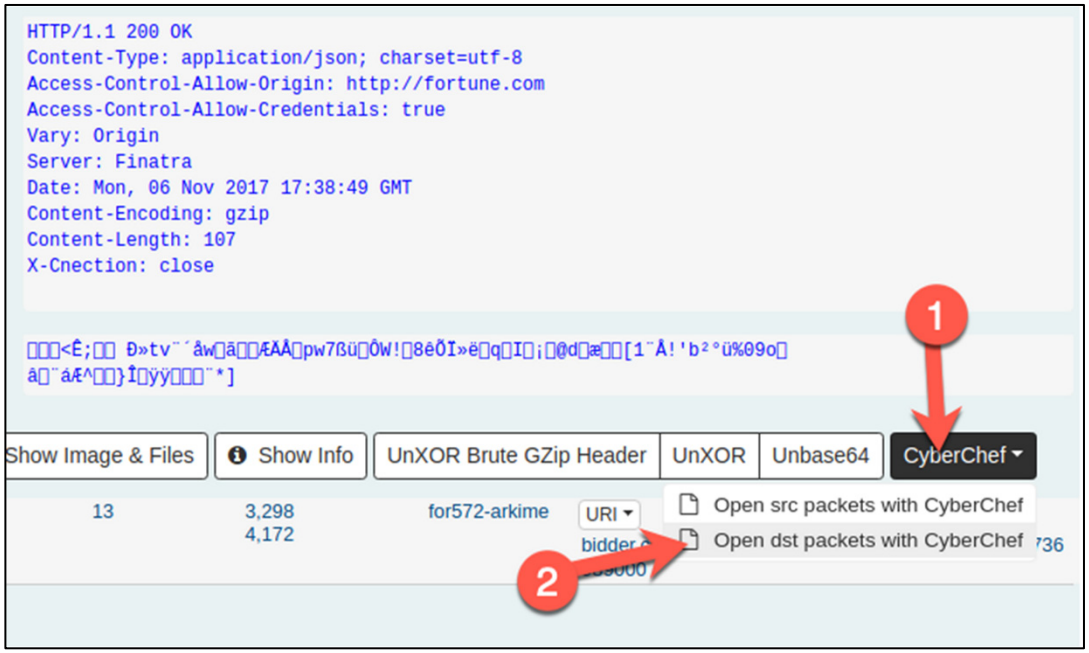
```
[
  { "op": "From Hex",
    "args": ["Space"] },
  { "op": "Strip HTTP headers",
    "args": [] },
  { "op": "Gunzip",
    "args": [] },
  { "op": "JPath expression",
    "args": ["exd.genimpids", "\\n"] },
  { "op": "JSON Beautify",
    "args": ["\\t"] }
]
```

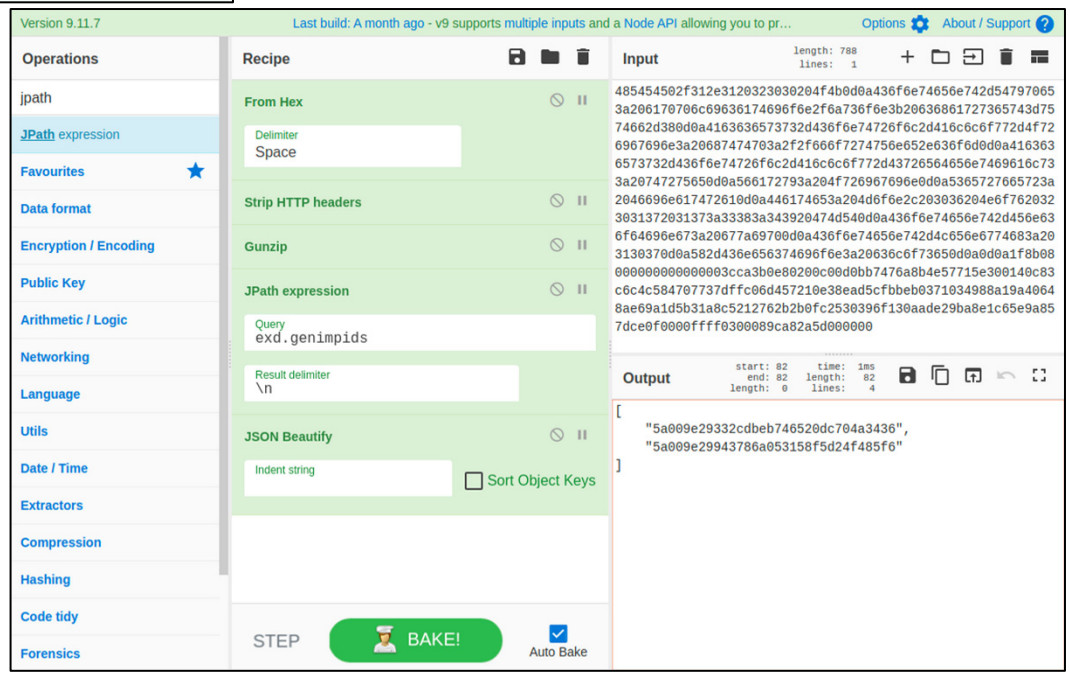
Arkime exposes this functionality from the unrolled session panel. The user can send either the source or destination packet data to the on-board CyberChef instance for handling.

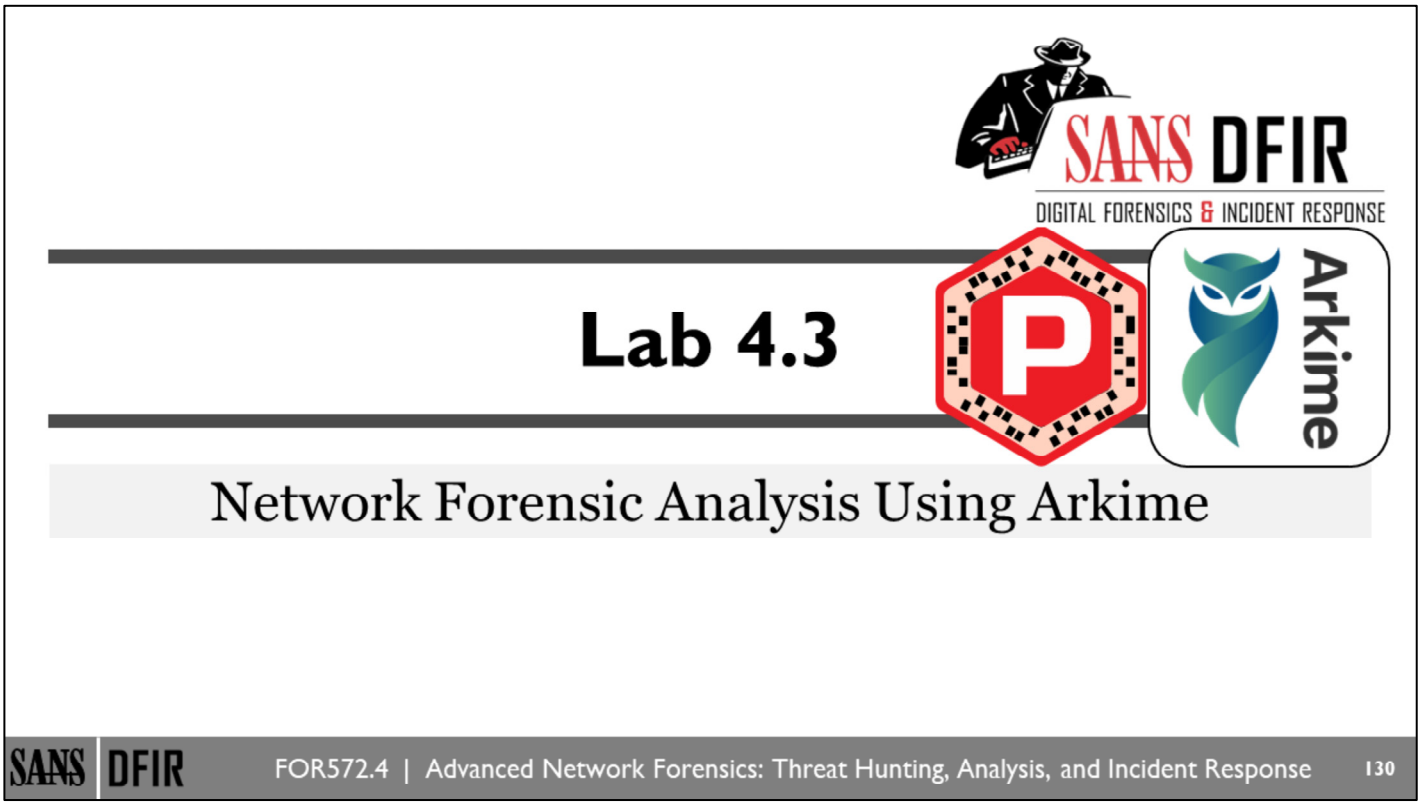
For those interested in learning more about CyberChef, an online version of the decoder is also available[2]. However, as with all online resources and tools, the user must beware of using this version of CyberChef with any sensitive data. Fortunately, downloading the latest version of the tool to run in standalone mode is a very simple and straightforward task.

References:

[1] https://for572.com/cyberchef
[2] https://for572.com/cyberchefonline







This page intentionally left blank.

Lab 4.3 Objectives: Network Forensic Analysis Using Arkime

- Become familiar with the Arkime full-packet capture and analysis platform
- Load captured traffic from existing pcap files into the Arkime platform
- Build search filters for traffic that has been indexed into Arkime
- Identify means of using Arkime to address large pcap collections at scale

This page intentionally left blank.

Lab 4.3 Takeaways: Network Forensic Analysis Using Arkime

- Arkime is a very useful tool for analysis of both existing pcap files and in collection mode
- Enables a good forensic analysis workflow that equally supports both post-incident analysis and live IR actions
- Search filters can quickly identify sessions of interest
- Exploring session associations based on various fields can bring to light correlations that would have otherwise gone unidentified

This page intentionally left blank.

FOR572.4 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response

Commercial Tools, Wireless, and Full-Packet Hunting

Authors:
Phil Hagen, Lewes Technology Consulting, LLC
phil@lewestech.com | @PhilHagen
Mat Oldham
mat.oldham@gmail.com | @roujisecurity

COURSE RESOURCES AND CONTACT INFORMATION

AUTHOR CONTACT
Phil Hagen/Lewes Technology Consulting, LLC
phil@lewestech.com | @PhilHagen

Mat Oldham
mat.oldham@gmail.com | @roujisecurity

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

DFIR RESOURCES
digital-forensics.sans.org
Twitter: @sansforensics

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

This page intentionally left blank.