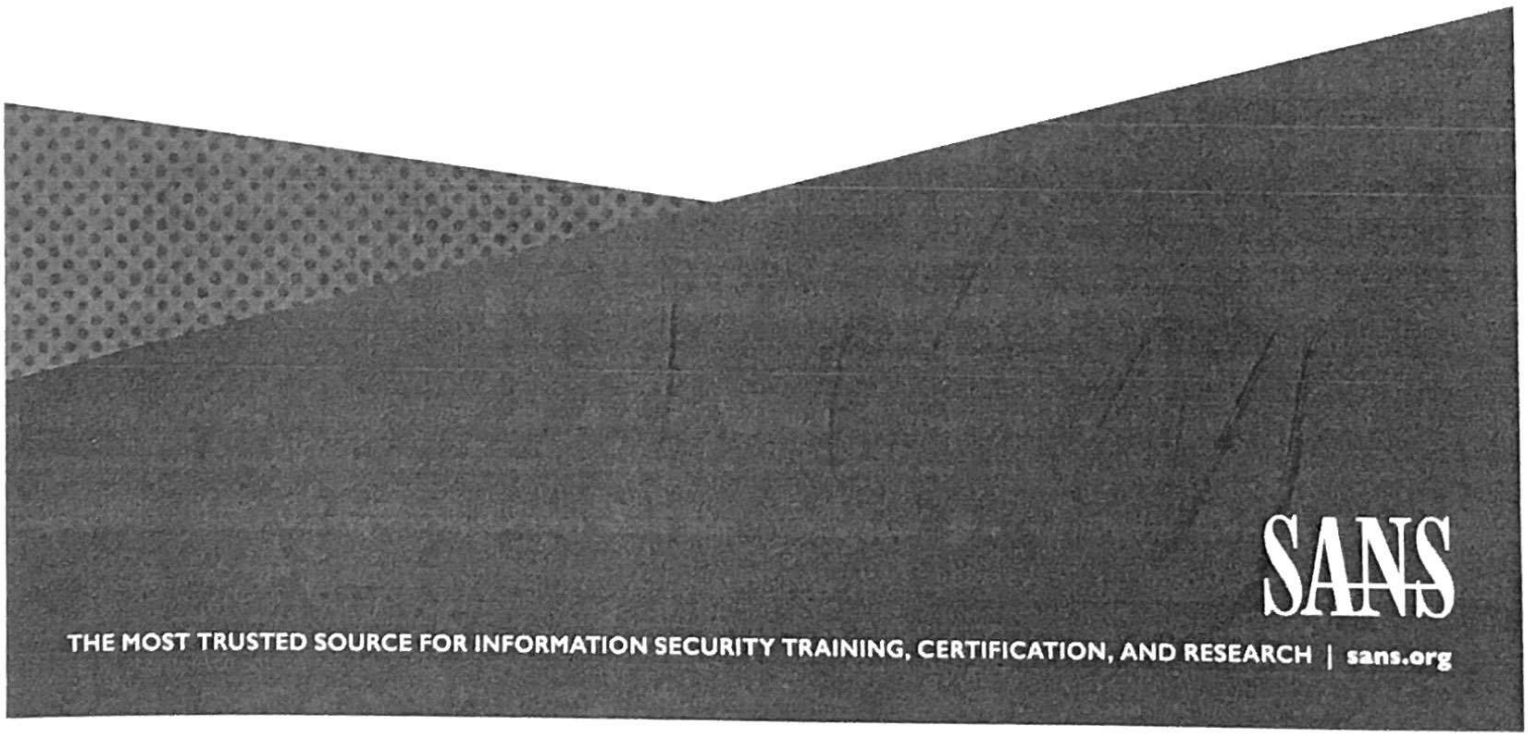
572.3
NetFlow and File Access Protocols

SANS

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

572.3
NetFlow and File Access Protocols



SANS

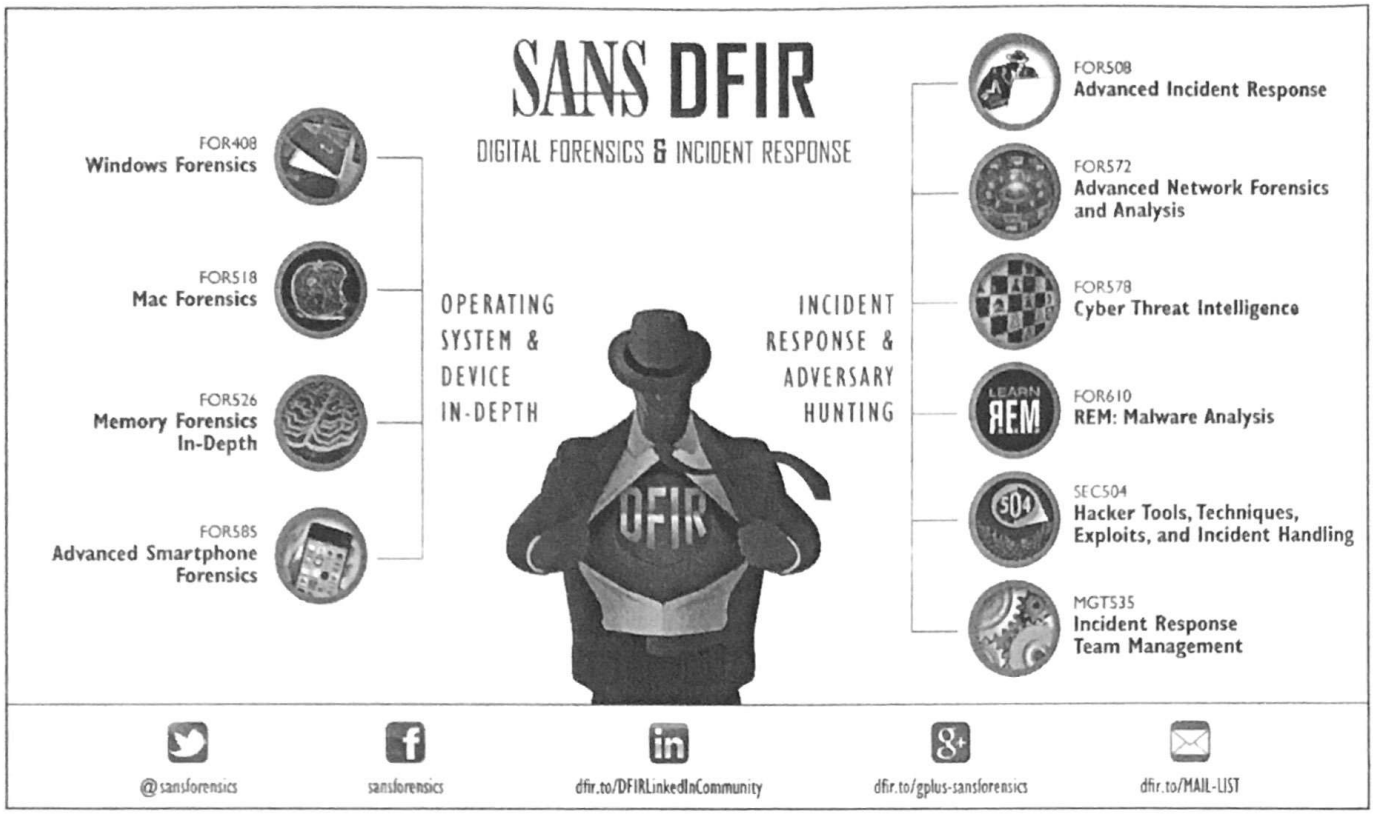
THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

FOR572.3 | Advanced Network Forensics and Analysis

NetFlow and File Access Protocols

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

FOR408
Windows Forensics

FOR518
Mac Forensics

FOR526
Memory Forensics In-Depth

FOR585
Advanced Smartphone Forensics

OPERATING SYSTEM & DEVICE IN-DEPTH

INCIDENT RESPONSE & ADVERSARY HUNTING

FOR508
Advanced Incident Response

FOR572
Advanced Network Forensics and Analysis

FOR578
Cyber Threat Intelligence

FOR610
REM: Malware Analysis

SEC504
Hacker Tools, Techniques, Exploits, and Incident Handling

MGT535
Incident Response Team Management

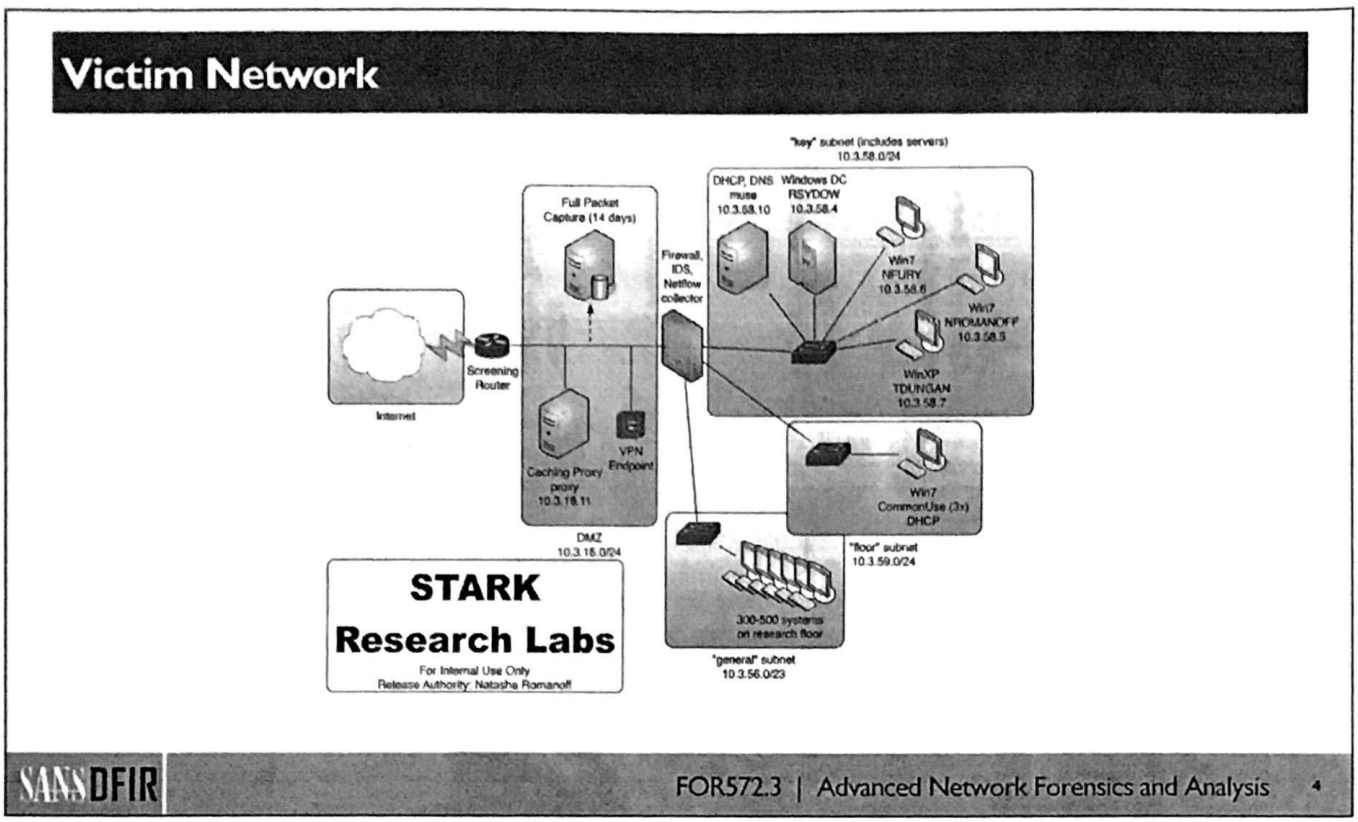
@sansforensics

sansforensics

dfir.to/DFIRLinkedInCommunity

dfir.to/gplus-sansforensics

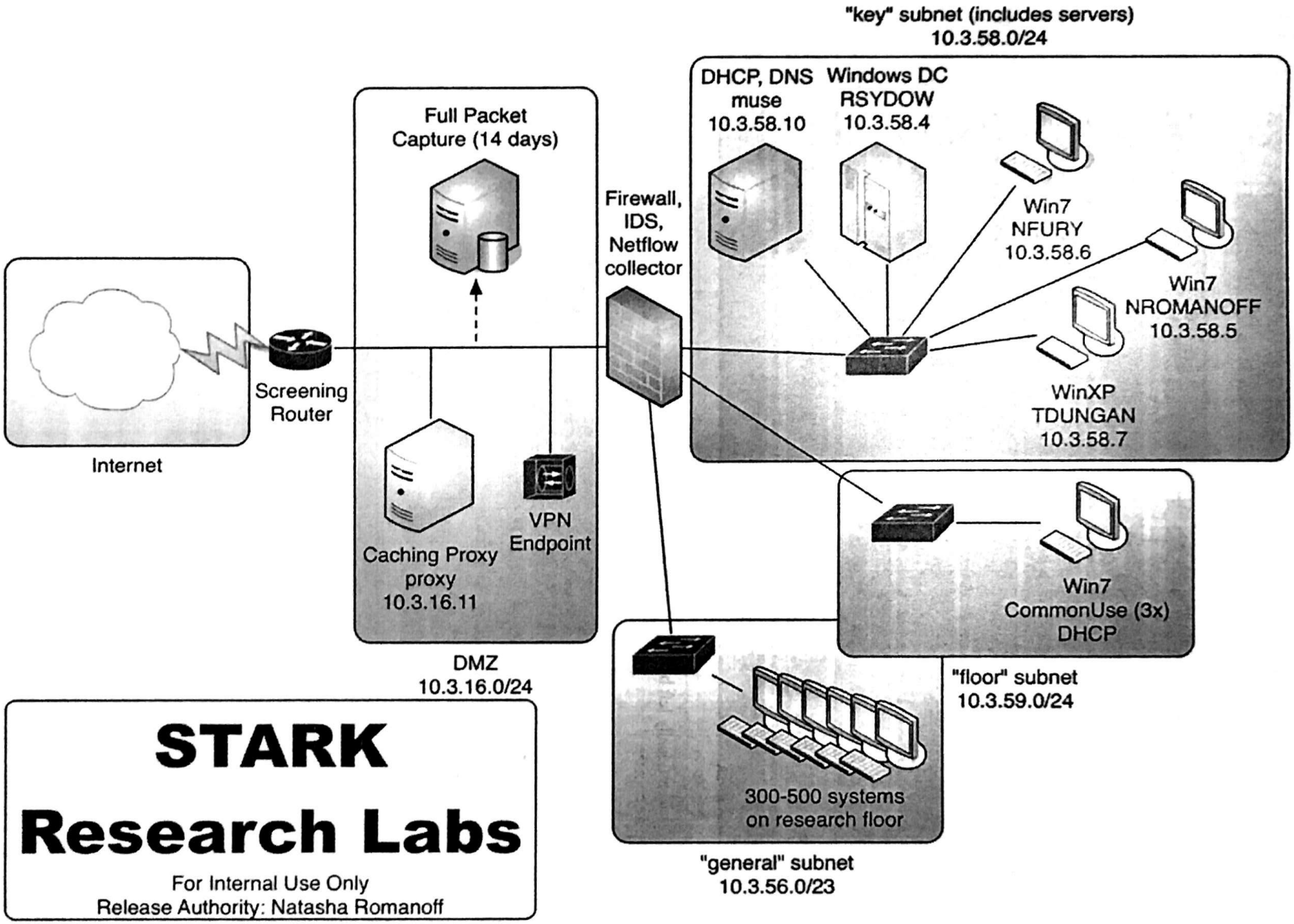
dfir.to/MAIL-LIST



The SRL network is representative of many medium-sized enterprises, with several major internal segments separated by a single firewall, but very little network protection between hosts once on the inside of that perimeter.

Key systems we are aware of (not an exhaustive list):

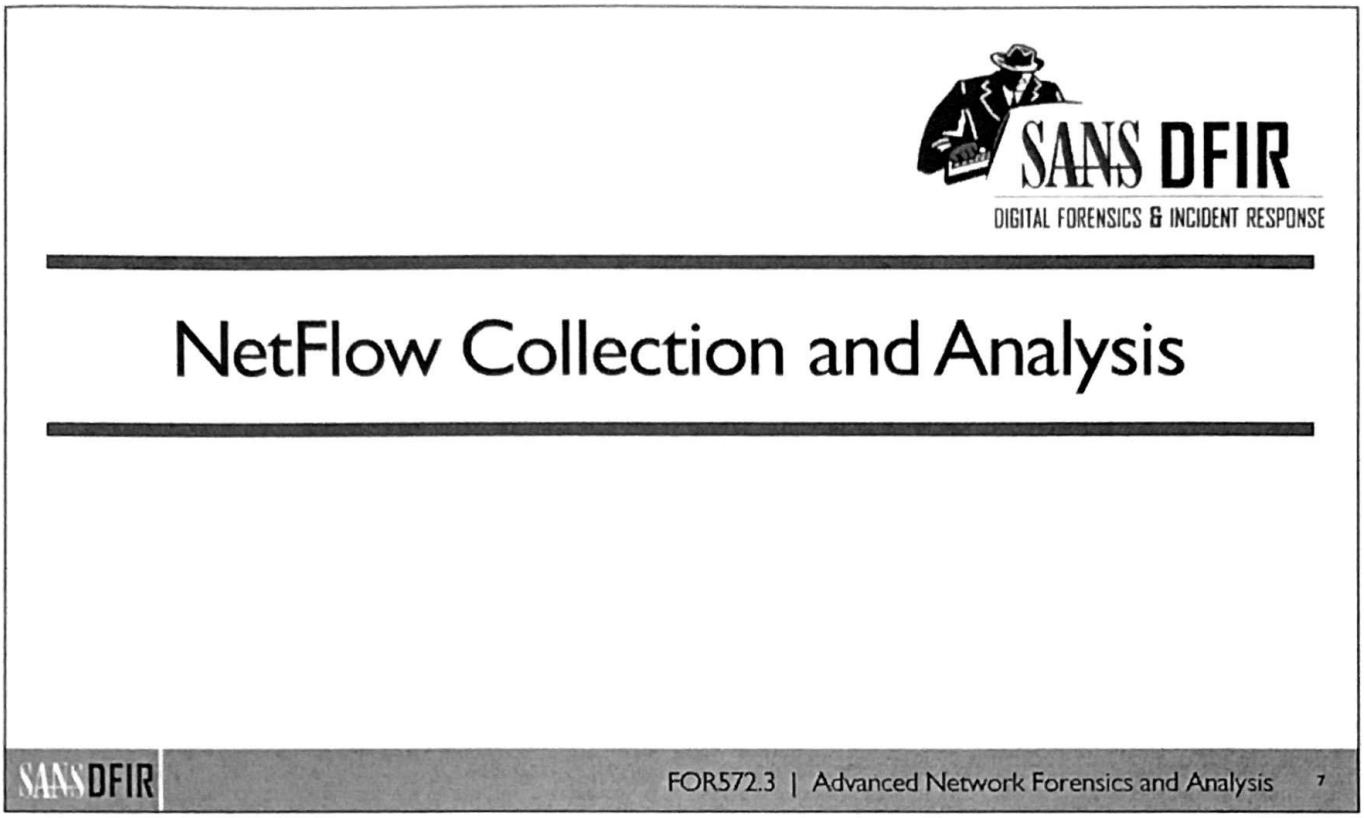
Host	IP Address	Role
Router	10.3.16.1	Inside interface of Internet screening router
Firewall	10.3.56.1/23	General administration and users
	10.3.58.1/24	Key users & SHIELDBASE servers subnet
	10.3.59.1/24	Production floor systems
	10.3.16.99/24	External (DMZ) interface
VPN	10.3.16.5/24	VPN Concentrator external interface
Proxy	10.3.16.11/24	Caching proxy for outgoing HTTP
Sniffer	10.3.56.254/24	Tcpdump rotating buffer. Approx 14 days of storage
Webserver	10.3.16.3/24	External presence of SRL
rsydow	10.3.58.4	SHIELDBASE Domain Controller – Windows 2008r2
tdungan	10.3.58.7	Tim Dungan's R&D workstation
nromanoff	10.3.58.5	Natasha Romanoff's workstation
nfury	10.3.58.6	Nick Fury's workstation



NetFlow and File Access Protocols

- NetFlow Collection and Analysis
- Open-Source Flow Tools
- Lab 06: Visual NetFlow Analysis with SOF-ELK
- File Transfer Protocol (FTP)
- Lab 07: Tracking Lateral Movement with NetFlow
- Microsoft Protocols
- Lab 08: SMB Session Analysis and Reconstruction

This page intentionally left blank.



This page intentionally left blank.

Full-content pcap Files Don't Scale

- For most organizations, full PCAPs are not an option for privacy and data volume reasons
- Multiple sampling locations will result in duplication of data captured
- Moved captured data can be re-captured
- Powerful hardware and storage is required
- Analysis is difficult with large captures
- SME cannot afford hardware for any of this

Although full pcap files are still the holy grail of packet analysis, they don't scale well in a large corporate environment for a few key reasons:

- Privacy laws may prevent the organization from implementing full packet capture hardware and analysis software. Some European Union states are particularly preferential toward citizens' privacy—even on corporate networks.
- Network speed and the nature of modern OSes have significantly increased the volume of network-based communication.
- If packet data is sampled at multiple locations, there will be duplicate packets stored in the repository. This can double storage requirements—or worse!
- Deep analysis of large datasets is nearly impossible without powerful server-based products connected to fast storage. These can be very expensive to implement and maintain.
- In order for the analyst to get the "full picture", the selected analysis software will need to process **all** data. If there are multiple, distributed collection nodes, such as at the organization's Internet gateways in Europe, APAC, and North America, the software must be able to summarize and centralize the data for easy analyst use. This requires international bandwidth, which is not usually cheap.
- Some corporations would not blink at a $1M project cost, but most organizations are not so fortunate. Most Small and Medium Enterprises (SMEs)—which make up the bulk of the business space—would never be able to consider such a solution.

What Is a NetFlow?

- Statistical record of packets with common attributes
  - Source/dest IPs, protocol, source/dest ports
- No content—just metadata
  - Source/dest IPs, protocol, source/dest ports
  - Start and stop times
  - Data volume sent
  - The ingress network interface

A NetFlow is simply a series of packets that at the point of sampling have some common attributes. These common attributes allow the packets to be grouped into a "Flow" record, which can then be recorded for later use. Because we understand how sessions and connections are established, we can therefore form some level of understanding about what is happening between the source and destination systems.

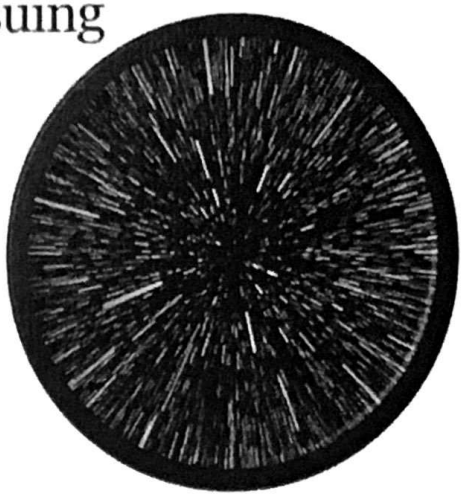
Consider the network occurrences that take place when a client connects their web browser to a web server. The browser makes an outbound connection from a high port (above TCP/1024) to the server's web service port (generally port TCP/80 or TCP/443). During the establishment of this session, a NetFlow sensor observing a link on which the connection is made. The sensor observes the TCP three-way handshake, indicating that a new flow has started. The sensor sees the source and destination IP addresses, layer four protocol, and source and destination ports in these initial packets. The sensor then observes the source and destination systems exchange packets with the same source and destination ports—these will not change without a new three-way handshake. Therefore, the sensor can record the following fact regarding this session:

- At $Time_{Open}$ system A connected to system B
- The port used by system A was $Port_A$ and the port used by system B was $Port_B$
- A volume of $Data_{A-B}$ was passed from system A to system B
- A volume of $Data_{B-A}$ was passed from system B to system A
- The connection closed at $Time_{Closed}$

Note: Although many NetFlow utilities can build NetFlow data from an existing pcap file, a NetFlow cannot be converted back to a pcap. NetFlow doesn't have any data!

Why Use NetFlow for Forensics?

- Warp-speed analysis: Quickly search traffic summaries for events worth pursuing
  - Top talking IP addresses
  - Network-to-network statistics
  - Contact with suspected C2 nodes
  - Traffic spikes (beacon or exfil)
- Encrypted communications
- Long-term evidence collections
  - Ideal for hunting: Using new intel with previously collected evidence



SANS DFIR | FOR572.3 | Advanced Network Forensics and Analysis | 10

If you've never used NetFlow to support DFIR investigations or hunting operations, you may question the value of a technology that limits visibility to layers four and below rather than allocating more resources for full-packet capture or layer seven visibility. The primary reason is one of speed—NetFlow gives an analyst the ability to find evidence of network traffic of interest at a rate many orders of magnitude faster than possible when parsing pcap files. NetFlow also provides a single means of searching all network traffic, regardless of protocol.

An efficient analyst can identify many conditions worth further attention in minutes or seconds, such as:

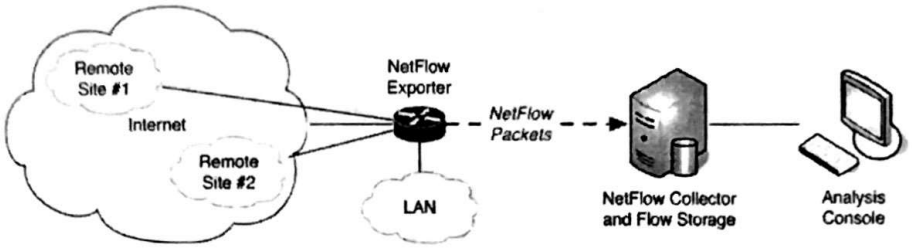
- Highest-volume IP addresses, subnets, or protocols (based on port usage)
- Most common or heavily trafficked network communications (based on ASNs)
- Communications with known-bad IP addresses identified by threat intelligence sources
- Erratic traffic spikes in terms of packet count (which could suggest beaconing) or volume (which could suggest data theft)

Another excellent use case for NetFlow analysis is with encrypted communications. Unless technology such as SSL interception is in use, full-packet capture of these communications is generally a waste of disk space. Especially with the advent of perfect forward secrecy and certificate pinning, extracting value of these encrypted communications is essentially a non-starter. However, NetFlow is invaluable since it's a record of the connection without its respective content.

Lastly, NetFlow perfectly supports the concept of a hunt team operation. Because NetFlow is relatively small to store and does not contain packet content, it is often retained for long periods of time. This allows DFIR and hunt team members to apply emerging intelligence against previously collected evidence, which is the truest definition of what's become known as "threat hunting".

NetFlow Architecture

- The core NetFlow components:
  - Exporter (device with NetFlow collection enabled)
  - Collector (where the NetFlow messages are sent)
  - Storage
  - Analysis Console



To build a NetFlow monitoring infrastructure, the following components are required:

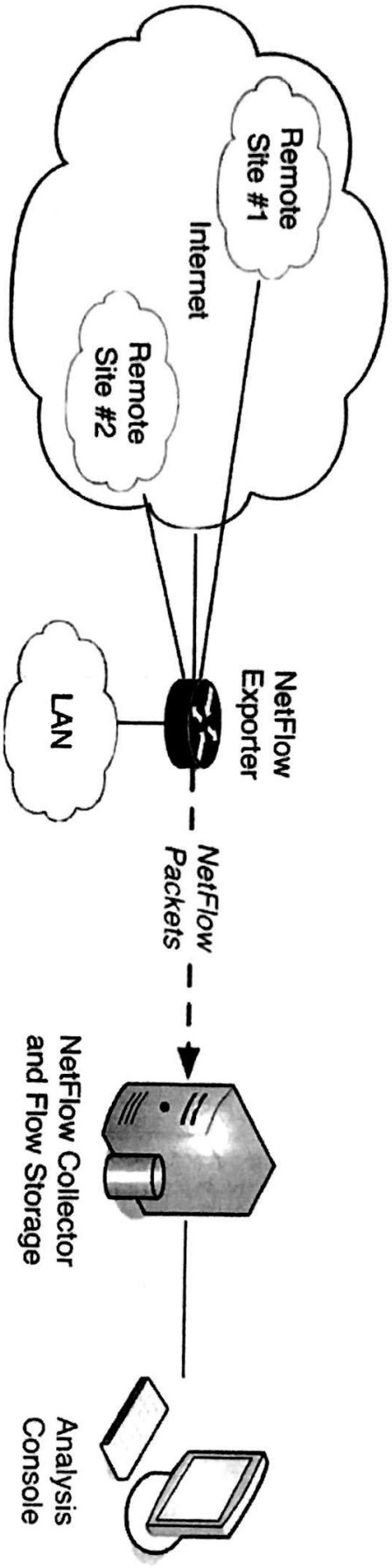
- **Exporter** – The device that has the ability to generate NetFlow records and to export these to a Collector.
- **Collector** – The device that receives the exported flows from one or more Exporter.
- **Storage** – Where the exported flows are stored, indexed ready for processing.
- **Analysis Console** – Where the operator sits and queries on the collector/storage device to gain an understanding of the movement of data on the networks being monitored.

All these components can be on one system (as is the case in developing, training, or testing configurations) and this is usually on open source operating systems (e.g., Linux Servers).

A more common deployment scenario is that border and core routers or those devices at chosen points are used for exporting data on flows. The flows are generally exported to a combined collector/storage device and analysis is conducted on the combined device by means of thin clients, web browsers, or command-line tools.

Planning for NetFlow before the network's configuration is finalized is considerably easier to implement and provision. Although small in size, NetFlow records do build up in the system, but like many behavioral monitoring systems, the benefit is in having old or historic data for as long as possible.

Also note that while we tend to think of a router as the NetFlow exporter (and use the classic router icon in most of the course materials), there are a number of additional platforms that can serve this function. Firewalls, switches with Layer 3 and Layer 4 visibility, and even endpoints running traffic monitoring probes can export NetFlow. Although exporting from all endpoints in even a moderately sized environment may not be a very practical undertaking, running an endpoint probe that's connected to the revenue port on a tap would be a great method for establishing tactical visibility during an incident response.



NetFlow Origins and IPFIX

- Cisco, 1990: Speed improvement for routing table and ACL lookups
- Initial decision recorded in flow record
  - Same session == same decision
- NetFlow version 5 still commonly used but limited
  - IPv4, unidirectional, inflexible data structure, etc.
- NetFlow v9, IPFIX, sFlow, J-Flow, AppFlow, etc.
  - Same analysis concepts, different data points

Cisco developed and built NetFlow as a performance improvement on their router equipment's IOS in the 1990s. It was originally developed as a route caching tool (called IP route-cache in Cisco IOS 11) that would be used in the routing decision process to improve efficiency by recording the routing decision upon the Flow's first packets. As subsequent packets from the same flow were received, the router used the flow record to determine routing and Access Control List (ACL) decisions rather than undergo a much slower route/ACL recalculation for each packet.

There are two important aspects that an analyst must consider before using NetFlow evidence:

- The interfaces on which NetFlow has been enabled. If traffic is being monitored selectively, the analyst must know what has been excluded.
- The type of NetFlow records generated:
  - Standard NetFlow generates or updates a record for each packet received on the monitored interface.
  - Sampled NetFlow tracks only every *n* packets or flows, depending on the NetFlow system and configuration.

Note that sampled NetFlow will under-represent data volumes because every packet's payload is not checked so its payload is not tallied. Sampled flow collections are not considered suitable for forensic purposes for this reason, although they still may support broader trending use cases.

NetFlow version 5 is still widely used despite its age and technical shortcomings. From a technical perspective, NetFlow v5 has a very specific data structure that can be somewhat constraining. For starters, IP address fields are allotted only 32 bits, so IPv6 is off the table. NetFlow v5 still represents flows in unidirectional form, meaning a successful TCP connection is defined by two individual flows. Additionally, modern network architectures have become quite complex and a model developed over 25 years ago certainly never accounted for NAT, MPLS, VPNs, load balancing, or a host of other features we take for granted.

NetFlow technology has adapted over the years, with new versions of the standard from a variety of vendors. Cisco created versions 7 and 9, and NetFlow v9 is now commonly used across many enterprises. The Internet Engineering Task Force (IETF) determined that vendor-driven protocols may not be in the best interests of an open and standards-based Internet. In the early 2000s, the IETF (still working with Cisco) created an open standard based on NetFlow v9 called the Internet Protocol Information Export, or IPFIX.

Interestingly, while many vendors have adopted flow tracking software that mimics NetFlow, they each brand it differently even though Cisco doesn't claim any trademark on the term.

- Juniper's implementation is called jFlow.
- Citrix's implementation is called AppFlow.
- sFlow is implemented by vendors like Huawei, IBM, Netgear, Alcatel Lucent, and others.

Background reading on the NetFlow version can be found at the IETF website. NetFlow v9 was originally defined in RFC 3954.[1] When the IETF finalized IPFIX (aka NetFlow v10), it will be defined by RFC 7011.[2]

Most modern routing hardware and server OSes support a variety of these standards. Cisco generally supports versions 5 and 9, and most non-Cisco equipment supports versions 5 and 9, as well as IPFIX.

References:

[1] `http://for572.com/hz1dw`
[2] `http://for572.com/gyxsw`

NetFlow Exports Use UDP

- UDP == not reliable/guaranteed
  - Problem amplified when including multiple flows in a single transmission
- Not just a speed consideration
  - Records can be forwarded to **multiple** collectors to facilitate different types of analysis
- Stream Control Transmission Protocol (SCTP)
  - Can mitigate some of these shortcomings

UDP is used to communicate completed flows' data from the NetFlow sensor (or exporter) and its configured storage device(s) (called collectors). This may at first seem like a poor choice from an availability point of view since UDP is an "unreliable" protocol that does not inherently guarantee delivery. However, there are several reasons UDP may be preferable:

- UDP traffic imposes low protocol overhead. The guaranteed message delivery provided by TCP requires higher CPU power and memory, and not all devices can handle this load when operating at high sustained bandwidth.
- Because UDP lacks connection management capabilities, it can be "split" or simultaneously directed to multiple NetFlow collectors. This allows, for example, the network and security teams to deploy their own collectors from the same cluster of exporters.

To reduce the risk of lost data, some devices now implement Stream Control Transmission Protocol (SCTP). In NetFlow configurations, this provides a confirmation from the recipients that each transmission was correctly received. If not, the sender can retransmit the data. If the transmission is confirmed as successful, the exporter deletes that item of data.

On some platforms (particularly with large or fast backbone links), the NetFlow exporting sensor will also provide interim updates to ongoing flows before they end. This is because tracking long data transmission can significantly impact the exporter's performance. Additionally, should the NetFlow record be lost, the collector would still have at least a partial record of the flow event.

NetFlow v5 Header and Flow Record

- Header precedes flow records in export packet
- Up to 30 flow records each

	0	1	2	3
0x00	Version		Record Count	
0x04	Exporter Uptime (msec)			
0x08	UNIX Time (Sec)			
0x0C	UNIX Time (Nsec)			
0x10	Flow Sequence			
0x14	Eng Typ	Eng ID	Interval	

	0	1	2	3
0x00	srcAddr			
0x04	dstAddr			
0x08	nextHop			
0x0C	inputSNMP		outputSNMP	
0x10	dPkts			
0x14	dOctets			
0x18	first			
0x1C	last			
0x20	srcPort		dstPort	
0x24	pad1	tcpFlags	prot	TOS
0x28	srcAS		dstAS	
0x2C	srcMask	dstMask	pad2	

The byte map on the left reflects the header fields for a NetFlow version 5 UDP-based export. The fields are decoded as:

Bytes	Contents	Description
0x00-0x01	Version	NetFlow export format version number
0x02-0x03	Record Count	Number of flows exported in this packet (1-30)
0x04-0x07	Exporter Uptime	Current time in milliseconds since the export device booted
0x08-0x0b	UNIX Time (Sec)	Current count of seconds since January 1, 1970, 0000 UTC
0x0c-0x0f	UNIX Time (NSec)	Residual nanoseconds since January 1, 1970, 0000 UTC
0x10-0x13	Flow Sequence	Sequence counter of total flows seen
0x14	Engine Type	Type of flow-switching engine
0x15	Engine ID	Slot number of the flow-switching engine
0x16-0x17	Sampling Mode/Interval	First 2 bits hold the sampling mode; remaining 14 bits hold value of sampling interval

On the right are the fields for a NetFlow version 5 record, of which there are between 1 and 30 in each exported flow packet. The items in bold are of most interest to the forensicator.

Bytes	Contents	Description
0x00-0x03	**srcAddr**	**Source IP address**
0x04-0x07	**dstAddr**	**Destination IP address**
0x08-0x0b	**nextHop**	**IP address of next hop router**
0x0c-0x0d	inputSNMP	index of input interface
0x0e-0x0f	outputSNMP	index of output interface
0x10-0x13	**dPkts**	**Packets in the flow**
0x14-0x17	**dOctets**	**Total number of Layer 3 (payload) bytes in the packets of the flow**
0x18-0x1b	**first**	**SysUptime at start of flow**
0x1c-0x1f	**last**	**SysUptime at the time when last packet of flow was received**
0x20-0x21	**srcPort**	**TCP/UDP source port number or equivalent**
0x22-0x23	**dstPort**	**TCP/UDP destination port number or equivalent**
0x24	pad1	Unused (zero) bytes
0x25	**tcpFlags**	**Cumulative OR of TCP flags**
0x26	**prot**	**IP protocol type (for example, TCP = 6; UDP = 17)**
0x27	TOS	IP type of service (ToS)
0x28-0x29	**srcAs**	**Autonomous System Number of source, either origin or peer**
0x2a-0x2b	**dstAs**	**Autonomous System Number of destination, either origin or peer**
0x2c	srcMask	Source address prefix mask bits
0x2d	dstMask	Destination address prefix mask bits
0x2e-0x2f	pad2	Unused (zero) bytes

Again, the items in bold are of the most interest to the analyst. For example understanding the source and destination addresses [**srcaddr** and **dstaddr**] together with the route a flow took [**nexthop**] and associated Autonomous System Numbers (ASNs) [**src_as** and **dst_as**] will allow the forensicator to track down the data's overall source, destination, and intermediate path.

Furthermore, the total number of layer three bytes [**dOctets**] will provide insight about the volume of data sent to the destination. (Note, however, that this figure cannot confirm the data was received—a firewall, routing problem, or other condition may have prevented its ultimate receipt.) However, this is of great particular interest when disk or system forensics, or other non-technical intelligence has suggested that an attacker may have transferred large volumes of data from the network.

NetFlow v9 Header and Template Record

- Exporter informs collector of data structures using multiple template records

	0	1	2	3
0x00	Version		Record Count	
0x04	Exporter Uptime (msec)			
0x08	UNIX Time (Sec)			
0x0C	Export Packet Count			
0x10	Source ID			

	0	1	2	3
0x00	Flowset ID <=255		Template Length	
0x04	Template ID >255		Field Count	
0x08	Field 1 Type		Field 1 Length	
0x0C	Field 2 Type		Field 2 Length	
0x10	...		...	
0x14	Field N Type		Field N Length	

Turning our attention to the equivalent byte map for records from NetFlow version 9 exports, we see some similarities but the underlying data structure is far more complicated than NetFlow v5, with many more available fields.

First, the overall export header is generally the same. However, the UNIX time is limited to seconds, and each exporter counts the number of export packets rather that exported flows. Additionally, NetFlow v9 coalesces several fields into a single "Source ID" field that ensures unique tracking for an individual exporter.

One of NetFlow v9's key features is the use of data templates to define the structure for each flow record. Each exporter publishes one or more templates to their respective collectors that explains the field layout for those export messages. There are 87 standard record types defined by the RFC, which include several vendor proprietary fields for expansion purposes.

An exporter can use multiple templates, differentiated by their Template ID value. This may be useful, for example, for a router that handles multiple network segments that exhibit different traffic profiles on each.

References:
`http://for572.com/hz1dw`

NetFlow v9 Data Record and Content Types

- Data records exported per defined template(s)

	0	1	2	3
0x00	Flowset ID >255		Flowset Length	
0x04	Rec 1 Field 1 (Per template length)			
	Rec 1 Field 2 (Per template length)			
	...			
	Rec 1 Field N (Per template length)			
	Rec 2 Field 1 (Per template length)			
	...			
	Rec N Field N (Per template length)			

(Padding as needed to line up on 32-bits)

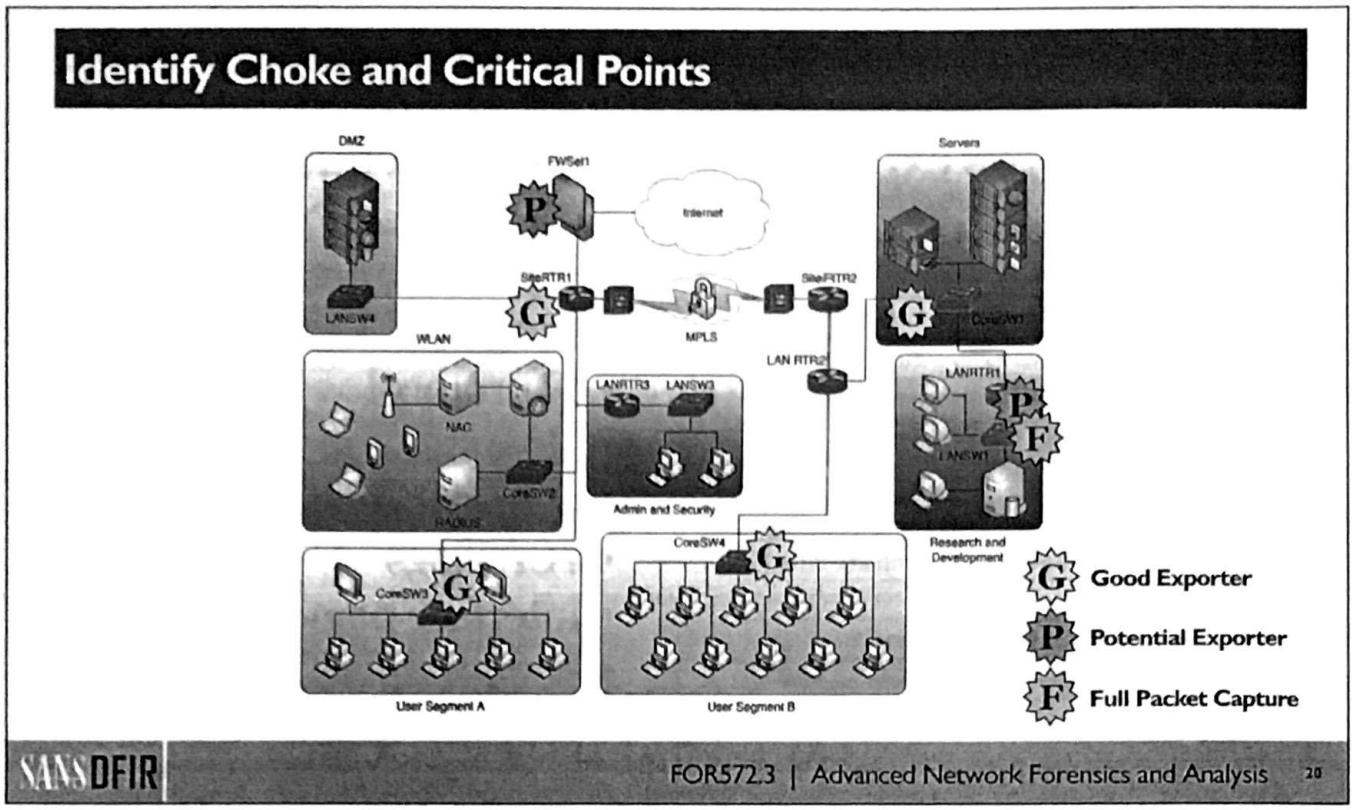
- 87 defined data fields
  - Directional IPv4 and IPv6 addresses, netmasks, MAC addresses, VLANs, ports
  - IP protocol
  - Flow start, end times
  - Byte, packet, flow counts
  - TCP Flags
  - Interface name, desc.
  - Vendor-specific fields

The templating system makes a clear map of the NetFlow v9 data records difficult to describe. They are practically arbitrary, according to each exporter's configuration. The record starts with a Flowset ID, which is also equal to the Template ID used for the subsequent record, followed by the total byte count for the flowset. (Different templates can be exported in the same packet by including multiple flowset IDs, each at the offset defined by the current flowset length value.)

The record fields in each data flowset reflect the values defined in the corresponding NetFlow template. Because each field's type and length are advertised in the template, the collector can "walk" the data per those instructions.

Again, with 87 standard values possible for each flow record, these records can become quite complex. Many of the fields are familiar from NetFlow v5, but with the advent of directionality, most of the addressing fields are not present in both source and destination directions.

References:
`http://for572.com/1h7rb`



In the example above, there are multiple locations where data could be collected. Additionally, there are different levels of data criticality and value. By understanding these, along with the volumes of data produced on each segment, the analyst can plan an efficient and effective capture solution.

The obvious initial location would be the FWSet1. However, by dropping back a level to SiteRTR1, more internal data can be seen—specifically between the three internal segments at the site. This subtle difference allows far greater visibility and internal activity monitoring. Administrators and analysts will have visibility of site-to-site connectivity in addition to internal flows.

Attackers tend to focus on the Critical Triangle of Compromise:

- The important data-hosting servers and systems (the target)
- The Administrators' desktop systems (the user group with complete control of the network)
- The Internet (where the attackers come from and want to take data to)

To assist in Malware hunting, the inclusion of the CoreSW3 & CoreSW4 would allow tracking workstation-to-workstation connections, which would not be visible at SiteRTR1.

In this case, with a smaller user group in the R&D Section, Full Packet Capture (FPC) on LANSW1 may suit the organization's risk appetite. However, this should be supported by NetFlow from the corresponding router. To complete the second remote site, the CoreSW1 would be a useful vantage point from which to observe server-to-server communication that is internal to the server LAN at the remote site.

Note: The Core Switches referred to in this diagram are Cisco 6509 or similar Enterprise equipment.

Although designing a NetFlow implementation is beyond the scope of this course, the following notional steps may be useful if you are assisting in the design decisions before such a deployment is performed.

1: Identify the critical data

It is not cost-effective for most organizations to capture all network traffic from every host on their network. However, by identifying and prioritizing the organization's most critical data, we can often reduce the surface area to manageable levels.

With the critical data identified, the location of the data, its backups and support systems should then be identified in the environment.

2 & 3: Understand the LAN/WAN links and map the network

It is amazing how many organizations lack a comprehensive (i.e. all systems) network map/diagram at even a high level (i.e., just the OS & IP address identified). If your organization lacks such a product, creating one should be a top priority. Without a high level diagram, how can you feel confident that the barrier coverage is complete?

Start by mapping the network route from the critical data to the Internet and to the system's administrator's desktops. This forms the Critical Triangle of Compromise, as attackers from the Internet will often target administrators to get to critical data. By mapping these you can see the barriers and routers between each of these three locations.

4: Identify choke and critical network points

With a suitable network diagram, identify the choke points where several subnets or VLANs join and consider these for some level of data capture (we will decide the type later).

Then, identify the critical network supporting switches and routers and ascertain their capabilities for capture. These devices are between user and data and the Internet (if the systems are able to connect to the Internet).

5: Identify critical data centers of gravity

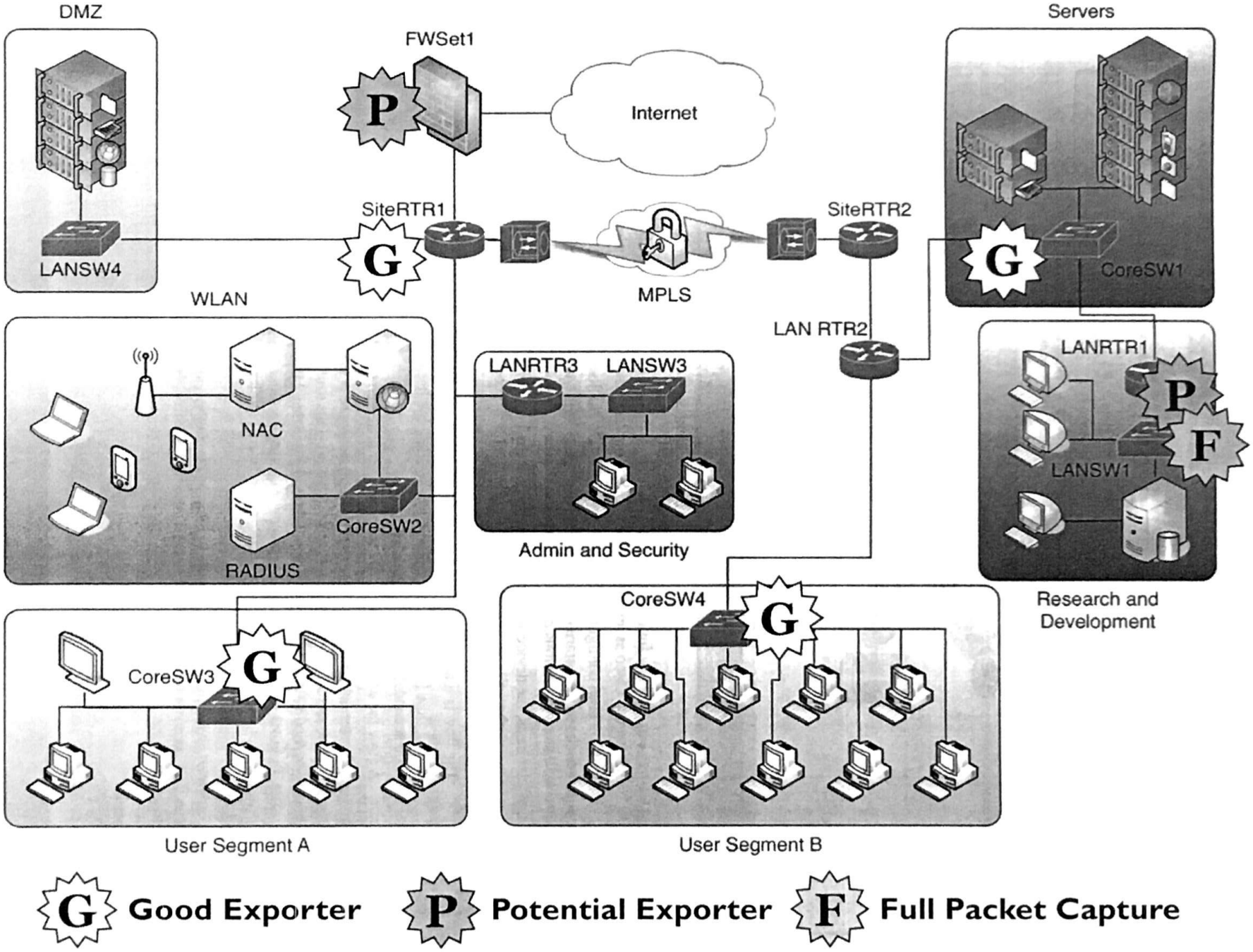
Centers of gravity are places in the environment that have high concentrations of a particular object—in this case, critical data. By defining these centers, the analyst gets away from old thinking like "it's a Domain Controller it must be important" or "that's a file server it must be important" and has then thinking about what is actually important in protection terms. Critical data centers of gravity include different things depending upon the role and industry of the organization, however, the following are examples:

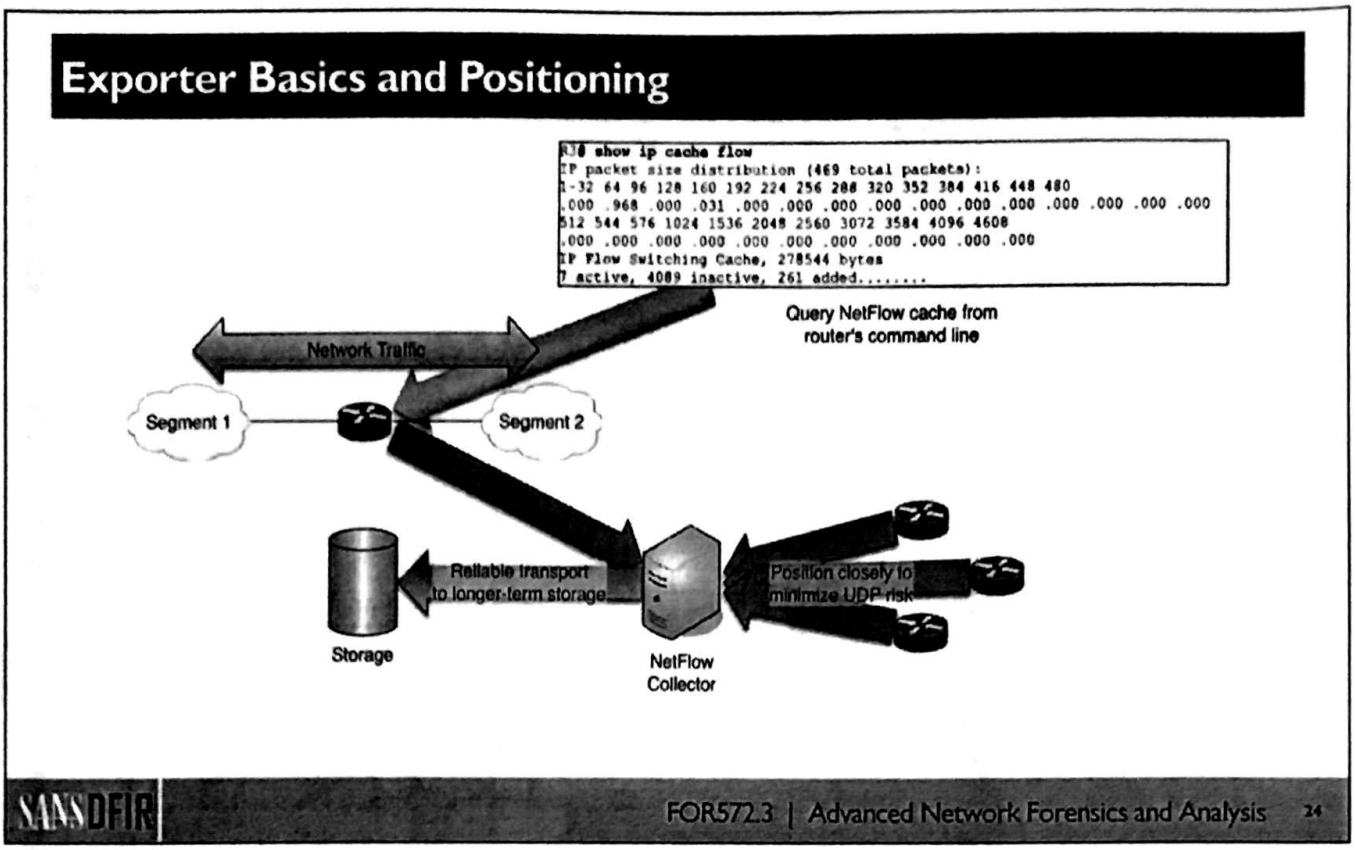
- Source code repositories (especially in software companies)
- Accounting systems (especially in financial services and publicly traded companies)
- System administrator systems (as they tend to contain plans, passwords, diagrams, and software)
- Senior Exec systems (especially in publicly traded systems)
- In-house developed web applications using AD credentials but not SSO (as userid/passwords pass through)
- DNS servers (as these direct users on the Internet and internally)

6 & 7: Plan NetFlow exporters and Identify full packet capture points
At high volume locations such as Internet gateways, enable NetFlow/IPFIX and send their flow data to a central server for storage and analysis. At critical data choke points and critical network devices, implement both NetFlow/IPFIX and full packet capture.
Send the flows to the central console and the full packet captures (FPCs) to a suitably configured storage server. Full captures can be implemented on many IDS/IPS devices or—in a pinch—a homebuilt system.

If storage is an issue, consider another retired desktop with several 3TB SATA drives attached and a Linux OS to act as the storage server. Ideally, you would want to use shell or other scripts to compress data and delete old captures when storage gets low.

8: Confirm legal compliance
Ensure that the organization's legal staff are thoroughly aware of the new capture plans and how the team plans to use the data and where. The *SANS LEG523: Law of Data Security and Investigations* course may assist organizations in defining their position with regard to the interception, storage, and use of digital evidence captured from the network and how it differs from hard drive storage.



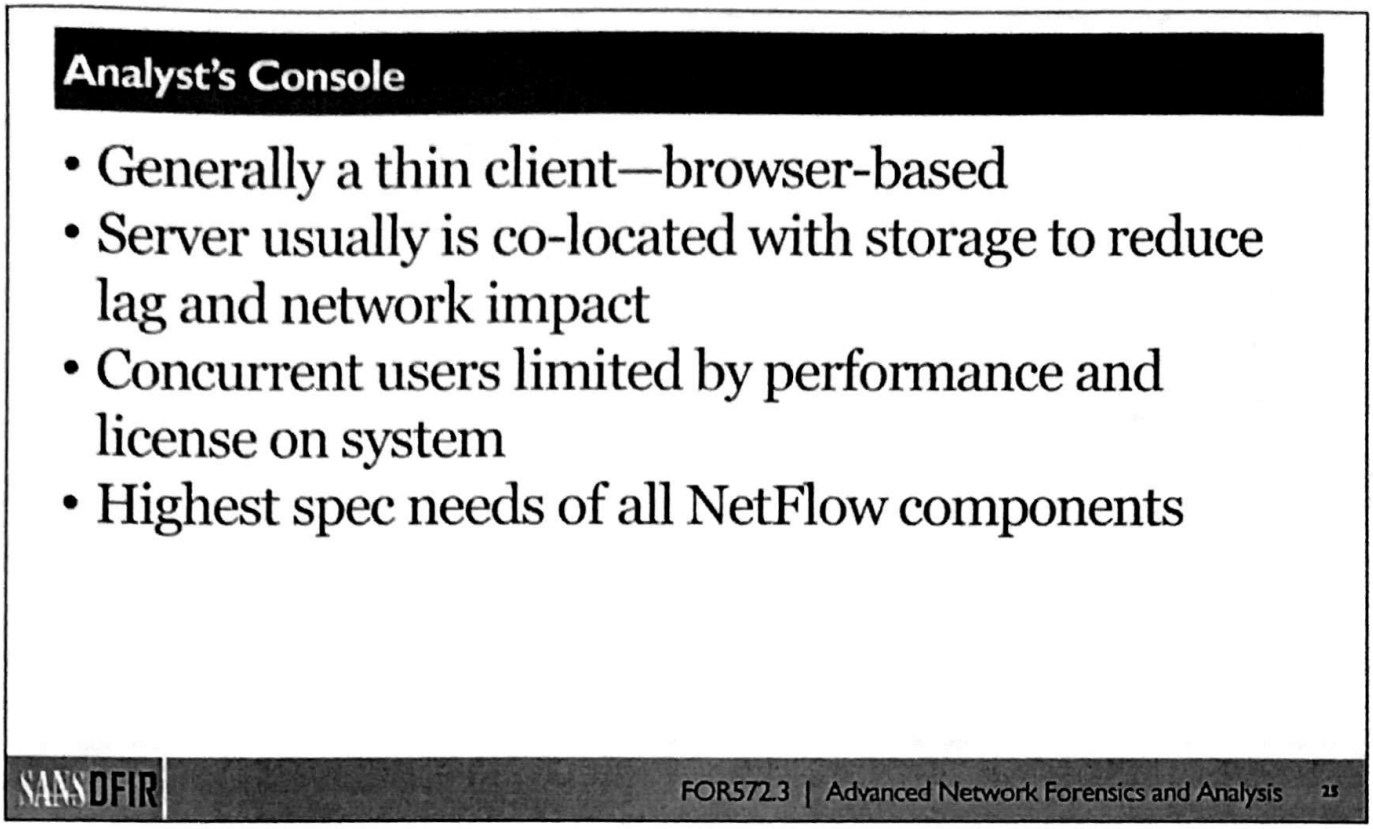


Now that we've seen NetFlow functionality at a high level, let's examine how an individual exporter actually works. Although we use Cisco as an example here, other implementations work in the same basic manner.

1. A device that has been configured as a NetFlow exporter will allocate part of its memory for the NetFlow cache. It uses this cache to temporarily store metadata relating to traffic that it has observed on an active interface. The exporter will store the various data points that define a flow and other related metadata.
2. On a regular basis, the device will check for aged records in the NetFlow cache. In the Cisco environment, the "`show ip cache flow`" command will display the list of records currently in the cache. These are flows that meet any of the following three criteria:
    - The flow is inactive (No data for a defined period of time [default = 15 seconds])
    - A long flow (A session that continues to see data but has existed for a defined period of time [default = 1800 seconds])
    - A flow for which a RST or FIN flag has been observed (TCP only)
3. When the exporter determines the flow is closed out, its record is packaged into an export packet with up to 29 additional flows, and sent to the appropriate collector(s). This uses UDP transport by default, so it's easy to see the risk associated with use of an unreliable protocol for investigative data. One lost NetFlow export could eliminate 30 flows of data. However, by placing exporters as close as possible to their respective collectors, the opportunities for packet loss is minimized. Some software NetFlow solutions have additional means of mitigating this risk, and there are transport protocols (such as SCTP) that will help as well.
4. The collector stores the NetFlow records to longer-term storage, which can use a reliable transport mechanism such as TCP, or even just the internal storage bus (SATA, SCSI, NAS/SAN, etc.) provided by the collector's hardware platform.

References:

`http://for572.com/4hv7t`

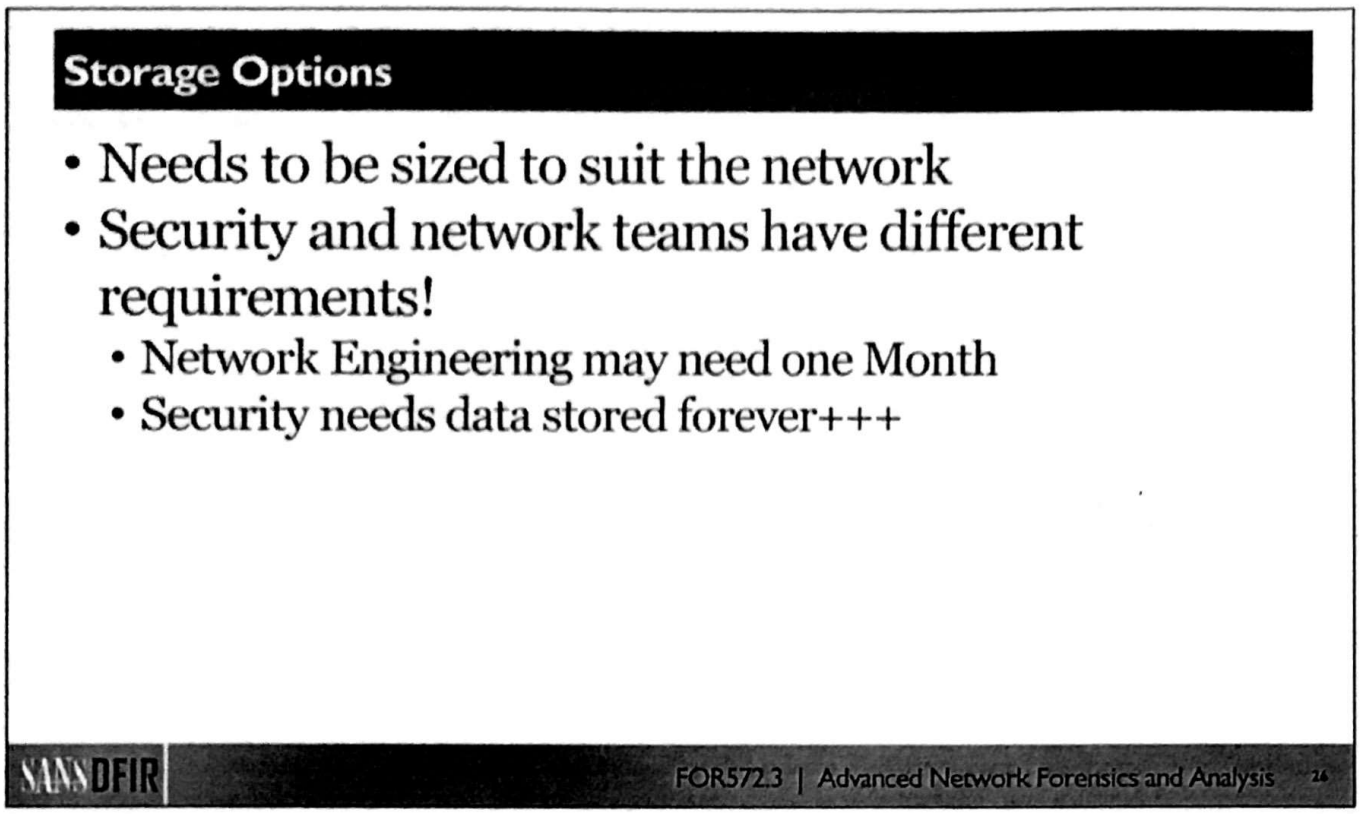


The analysis consoles are usually browser-based or a similar "app" that's often just a re-skinned or framed browser with some authentication and branding on top.

In a thin client or browser-based solution, the console is provided by a server that is "nearby" or on the same system as the storage. This topology reduces the network impact of querying and recalling vast amounts of data from the data store.

In commercial offerings, the number of concurrent users is generally limited by hardware or performance, and of course licensing. In open-source implementations, performance tends to be the only limiting aspect. It is worth noting that several concurrent users can quickly place a significant load on an analysis server. Simultaneously drilling down into the underlying data from a large number of NetFlow consoles is very I/O-intensive. Depending on the number of users, network load can also be an issue.

Of all the components in the NetFlow infrastructure, this is the most performance hungry. Concurrent users place massive load on CPU and Disk I/O simply by browsing the data. Additionally, recalling long-term historical data requires all of the NetFlow records to be stored locally and available to the analysis server. This creates ever-growing storage requirements that will need monitoring and scaling to ensure performance is not impacted and data is not lost.



Many storage options are determined by the intended analysis software. The software links the stored NetFlow records to the human analyst, and it usually configures and manages the storage for this reason. As with all deployment aspects, the storage for the NetFlow records needs to be planned and provisioned to suit the network.

It is worth noting that the Security and Network Engineering teams' requirements are vastly different. The network team generally requires the NetFlow data in near-real-time, because their mission focuses on factors such as changes in bandwidth, hardware uptime, link saturation, etc. The recent data—perhaps for the last few weeks or months—is most critical. Some teams may keep snapshots of old data as well, but this is rare and a very low-fidelity source at best.

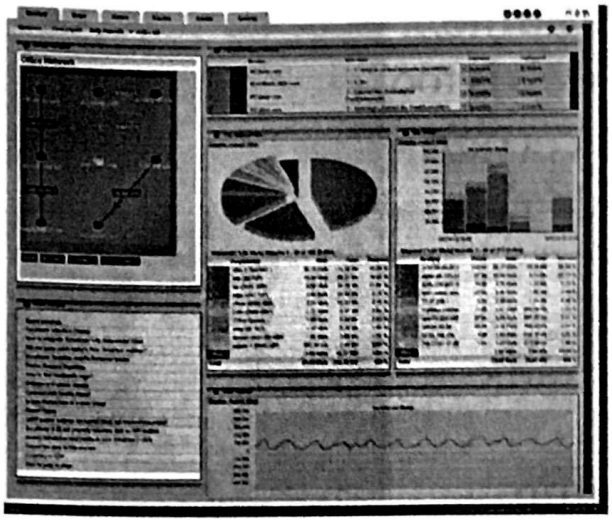
However, DFIR team members will seek maximum retention of all NetFlow records. They point to publications such as Mandiant's Annual Threat report (2016),[1] which stated that the average time between a compromise and its detection was 146 days. Being an average, the mathematically inclined will quickly see that many incidents took far, far longer to discover.

It is for reasons such as these that the security teams often prefer to use their own NetFlow collection and storage solutions. This model allows them to address the longer-term retention requirement without impacting the Network Engineering team's architecture and planning. Security operators can then identify communications between compromised internal hosts and their Internet-based command and control servers many months or even years after an attack. This data longevity is something that most IT and network departments cannot undertake.

References:

[1] http://for572.com/u0srz

Storage Formats Vary Between Collectors



- Various storage methods
  - Database
  - Binary files
  - ASCII files
- Open-source tools often use files
  - Stored in typical directory structure

There are a variety of different data formats used to store NetFlow records. Commercial tools tend to use databases. For example, Plixer Scrutinizer (shown in the screenshot) uses the MySQL Server.[1]

Meanwhile open-source tools (such as `nfcapd`, SiLK PS, etc.) tend to use binary and ASCII file formats for storing their NetFlow records. The SOF-ELK VM we will use later in class uses the Elasticsearch document storage and search database. Some, such as `nfcapd`, can store data in multiple file formats. This is may be due to the open-source community's approach to code development or their desire to allow users to choose the best storage solution for their overall NetFlow strategy and architecture. For example, one may want to use multiple tools to access and analyze the same data set. We will cover some of these tools in greater depth in the Open Source NetFlow tools section of the course.

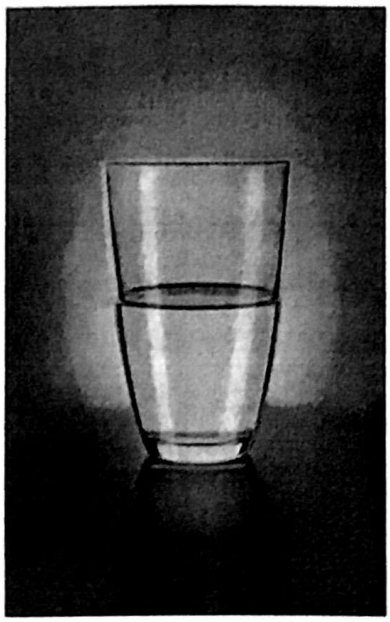
With file-based storage, accessibility is relatively easy—regardless of the format. Therefore, for some organizations, the decision of what tool to implement is more driven by the user's requirements to further mine the NetFlow data using other tools rather than just the initial collection and presentation ones.

References:

[1] `http://for572.com/h6ldq`

Analytic Process Must Account for Lack of Content

- pcap is a challenge due to volume...
  ...limited view from NetFlow's is too
- Findings and leads are less certain
  - TCP/80: HTTP? SSH? Raw socket?
  - Small flow events: AJAX? C2?
  - Large flow events: VPN? Exfil?
  - Benefits from endpoint intel, traffic baselines, etc.
- Silver lining: Encrypted traffic looks same as the rest in NetFlow

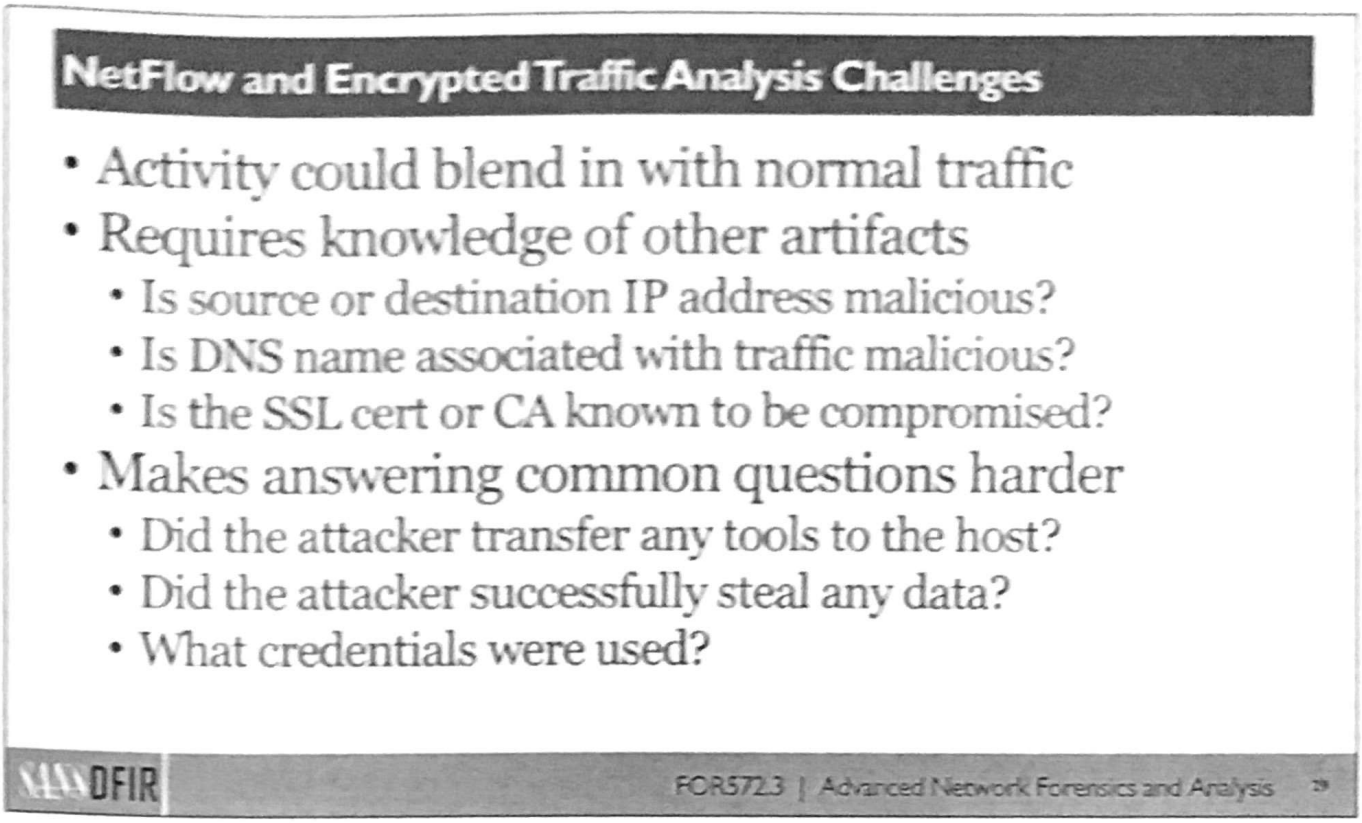


Despite being a very valuable alternative to full-packet capture's legal and technical constraints, NetFlow analysis is no panacea. The inherent lack of content means we are left with "investigative leaps of faith" about the contents of a connection or even the service used to provide the content.

The most obvious example is that merely observing traffic that uses a well-known service port is something that cannot be confirmed with NetFlow evidence alone. If there is no log evidence, full-packet capture, or host-based analysis available to confirm a TCP/80 NetFlow event **actually** contains HTTP traffic, we are left with an assumption—which is certainly not ideal. However, such a hypothesis is reasonable in the incident response world. Of course, having a hypothesis is not bestowing blind trust, so we must always watch for anything that would refute it—protocol errors from an HTTP proxy log, alerts from a protocol-aware intrusion detection system, etc.

Other methodologies we have to improve our understanding of NetFlow involve the baselines of normalcy from the environment being investigated. For example, suppose 95% of the usual HTTP connections from the engineering department's network enclave involve endpoints in 20 specific autonomous systems (AS). When a new AS appears in the top 95% of traffic, this becomes an anomaly we should track down with lower-level investigation. Similarly, if you can cross-check inbound traffic to a public-facing web server with IP addresses that are known to participate in a botnet or traffic redirection service, it becomes easier to characterize the traffic as suspicious or outright malicious—even without any content at all. In fact, this intelligence- and baseline-driven profiling becomes a foundation of threat hunting—finding evil where you didn't know it existed.

Fortunately for today's network forensic analyst, the ability to use NetFlow-based investigative techniques provides the same value when looking at encrypted communications as with those in traditional plaintext protocols. Consider that a properly encrypted channel is functionally opaque—the content is assumed to be inaccessible and may as well be written off—even discarded in some cases. However, the remaining portion, the headers, are ideally summarized in a format that mirrors the artifacts we have available from NetFlow. Looking forward to a more heavily encrypted Internet, this is undoubtedly a valuable skill to have.



In an ideal world, an incident responder could use network traffic analysis to identify exactly what an attacker accomplished during their activity in a victim's environment. However, in reality, our limited scope of evidence (in terms of available artifacts or data retention) and the use of undocumented protocols and encryption can be a significant hindrance to those efforts. Using limited-scope evidence such as NetFlow or examining encrypted traffic can make it easy for an attacker to hide among legitimate traffic. This is the classic example of finding one particular needle in a stack of thousands of other needles. When you consider the exponentially increasing volume of data, you must consider that in a typical network environment, the needle hunt gets moved to a factory where they also make magnets. Not an easy task.

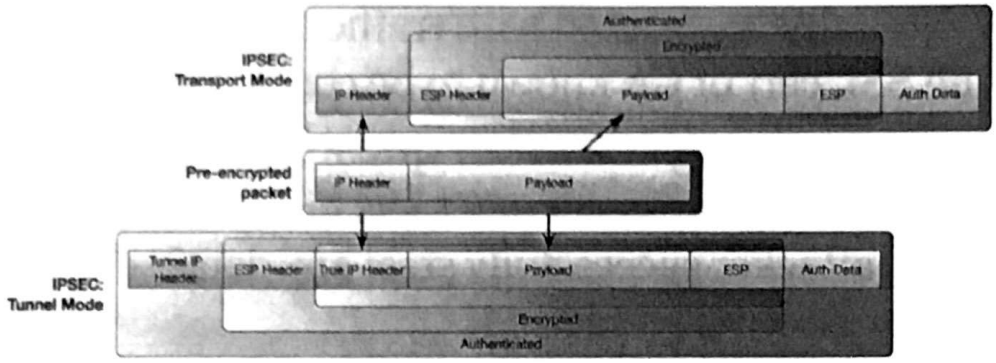
Put simply, it becomes quite difficult to answer common questions about the suspicious activity:

- Did the attacker transfer any tools to the compromised host?
- Did the attacker successfully steal any data?
- What credentials were used?
- How long was the attacker active?
- Was the attack the result of a nation-state or a nuisance actor?

Although an analyst may not be able to answer all of these questions, understanding the underlying network protocols and how to analyze them can enable us to make informed, educated guesses that benefit the incident response even when full traffic content is unavailable.

IPsec: Mode Determines Artifacts

- Establishes network tunnel between two endpoints
  - Host-to-host or network-to-network connections
- Commonly used for Virtual Private Networks

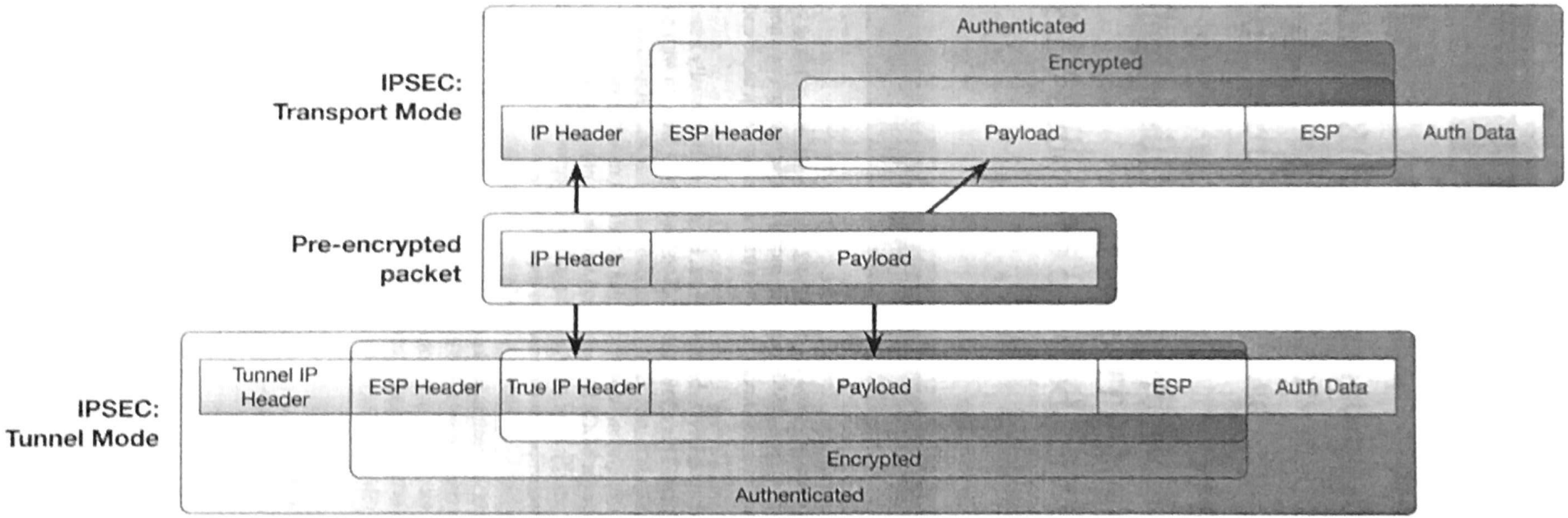


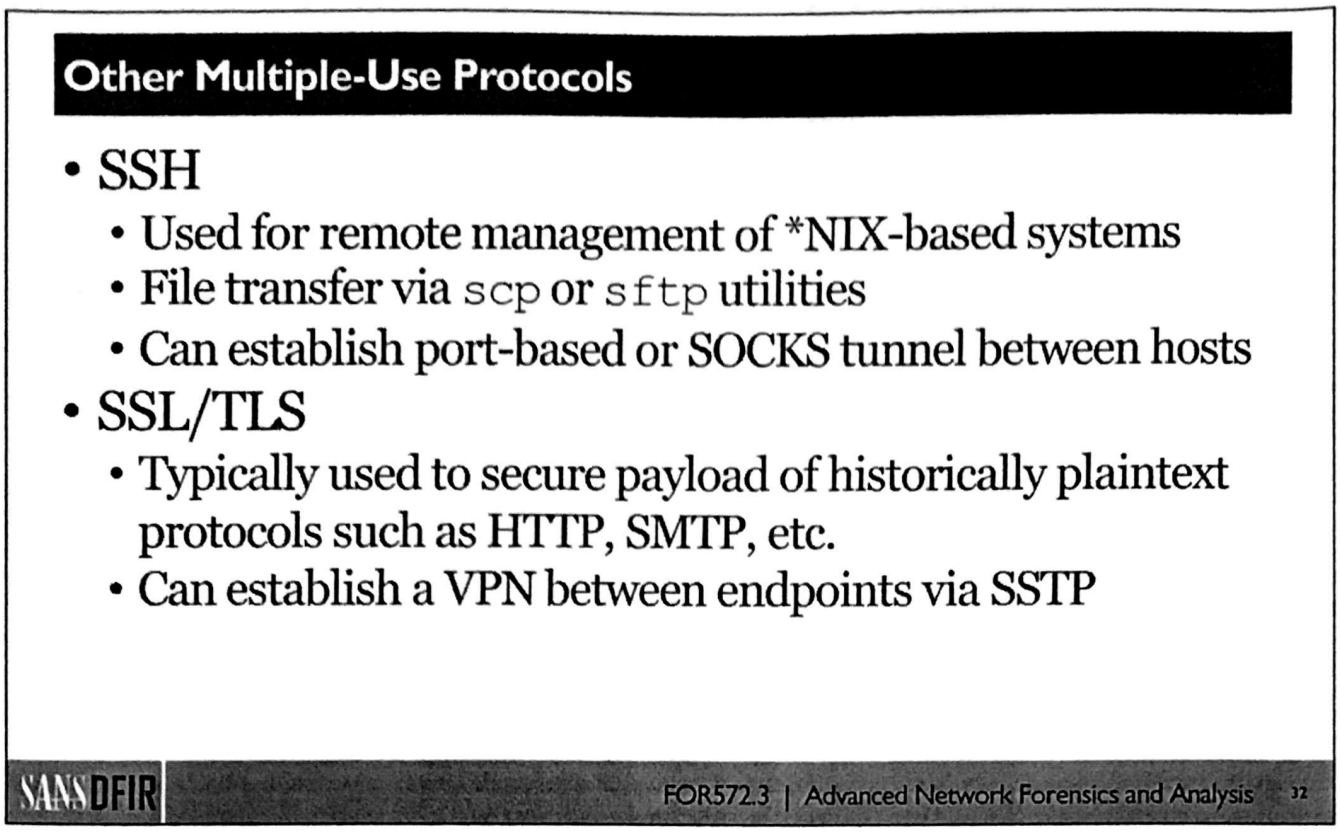
From an encrypted point of view, consider the available artifacts from a common encryption protocol, Internet Protocol Security, or IPsec. IPsec is a protocol suite that is used to secure IP-based communications by authenticating and/or encrypting each IP datagram of a communication session. It is typically used between two hosts using what is known as "transport mode" or it used between two networks using "tunnel mode". Using transport mode, only the payload of the IP packet is authenticated/encrypted. In tunnel mode, the entire packet is authenticated/encrypted, then encapsulated in a secondary packet that handles the routing of the communication between the two end points (aka through the tunnel).

IPsec is most commonly used for creating Virtual Private Networks (essentially two separate networks, typically geographically separated such as a remote office with the main corporate office), host to network communications (such as remote user access to a corporate network) or host-to-host communications (such as private chat).

When using IPsec in transport mode, the IP header is used as the true destination for the encrypted traffic. The payload itself is encrypted and authenticated but the true IP header remains intact and visible to an outside observer. However, when using IPsec in tunnel mode, the IP header is used to direct the IP address between two systems. However, the IP header does not represent the final destination. Traffic contained within the encrypted tunnel also contains information about the true destination IP address for the intended traffic in addition to the payload.

From a NetFlow perspective, consider that observing an IPsec connection in transport mode, the IP addresses we have available are the true endpoint IPs participating in the connection. However, a NetFlow record for a tunnel mode connection only contains the IP addresses for the two VPN endpoints, and we lose all visibility to how many systems are behind each end of the connection. Because there is no artifact that would reflect in NetFlow to indicate if the connection is in transport or tunnel mode, we may have to rely on volume or timing artifacts to characterize the mode of operation.



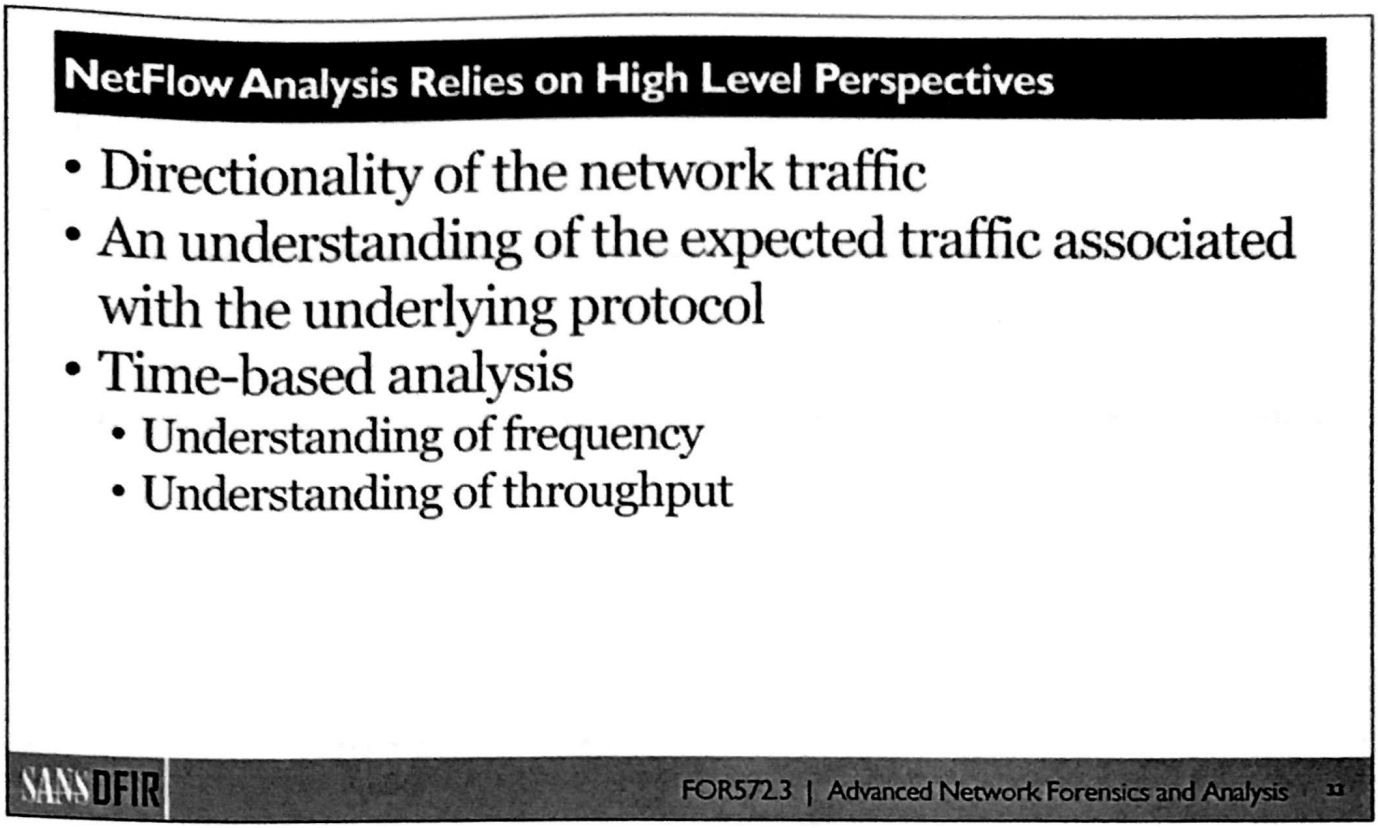


A number of other protocols are commonly used for encrypting network traffic, including both SSL (or its follow-on, TLS) and SSH. Similar to IPsec, SSL and SSH can both encrypt traffic between two hosts or create a tunnel between two endpoints that encapsulates additional protocols transmitted between those endpoints.

SSH is often used by attackers targeting UNIX-based systems, because SSH is usually installed and activated by default to enable remote management. This allows the attackers' activity to blend in with normal traffic. The SSH protocol is also used to transfer files via numerous different command-line and GUI-based utilities. It can also be used to port-forward traffic from one system to another either locally or remotely on a given system and is also capable of creating a Socket Secure (or SOCKS) connection which can proxy all TCP connections to an arbitrary IP address. If analyzing SSH-based activity from NetFlow, the difference between these different use cases may be apparent only from the average throughput and duration of the connection.

SSL is commonly used by attackers for custom backdoors since SSL encryption is a well-documented standard and fairly easy for even an entry level criminal software developer to implement. Although SSL was originally created as a wrapper around plaintext protocols, it is now common to see it used for VPN connections, since Microsoft Windows now prefers Secure Sockets Tunneling Protocol (SSTP) for its VPN connections.[1] From a NetFlow perspective, an SSTP connection often appears to be a long-running HTTPS connection over TCP/443.

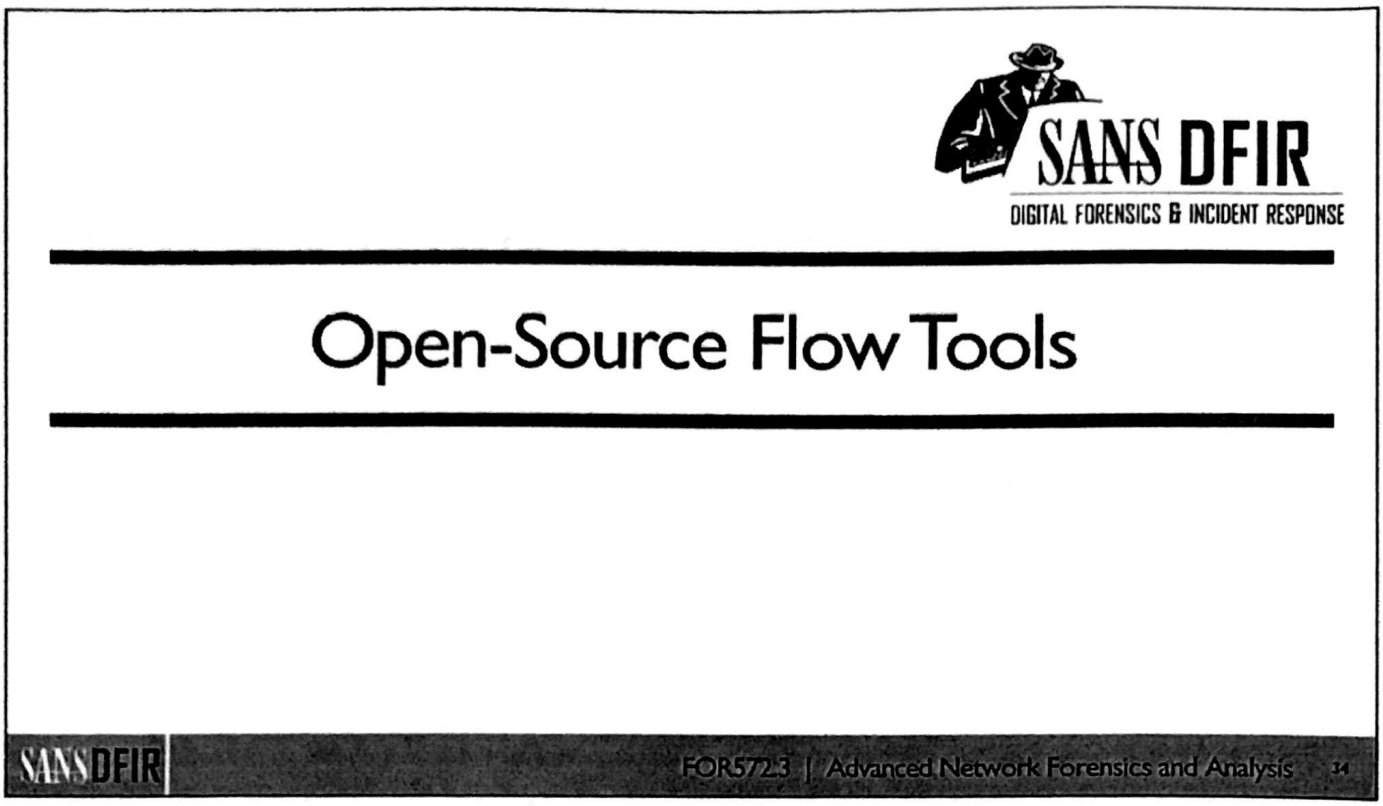
References:
[1] `http://for572.com/9f8n6`



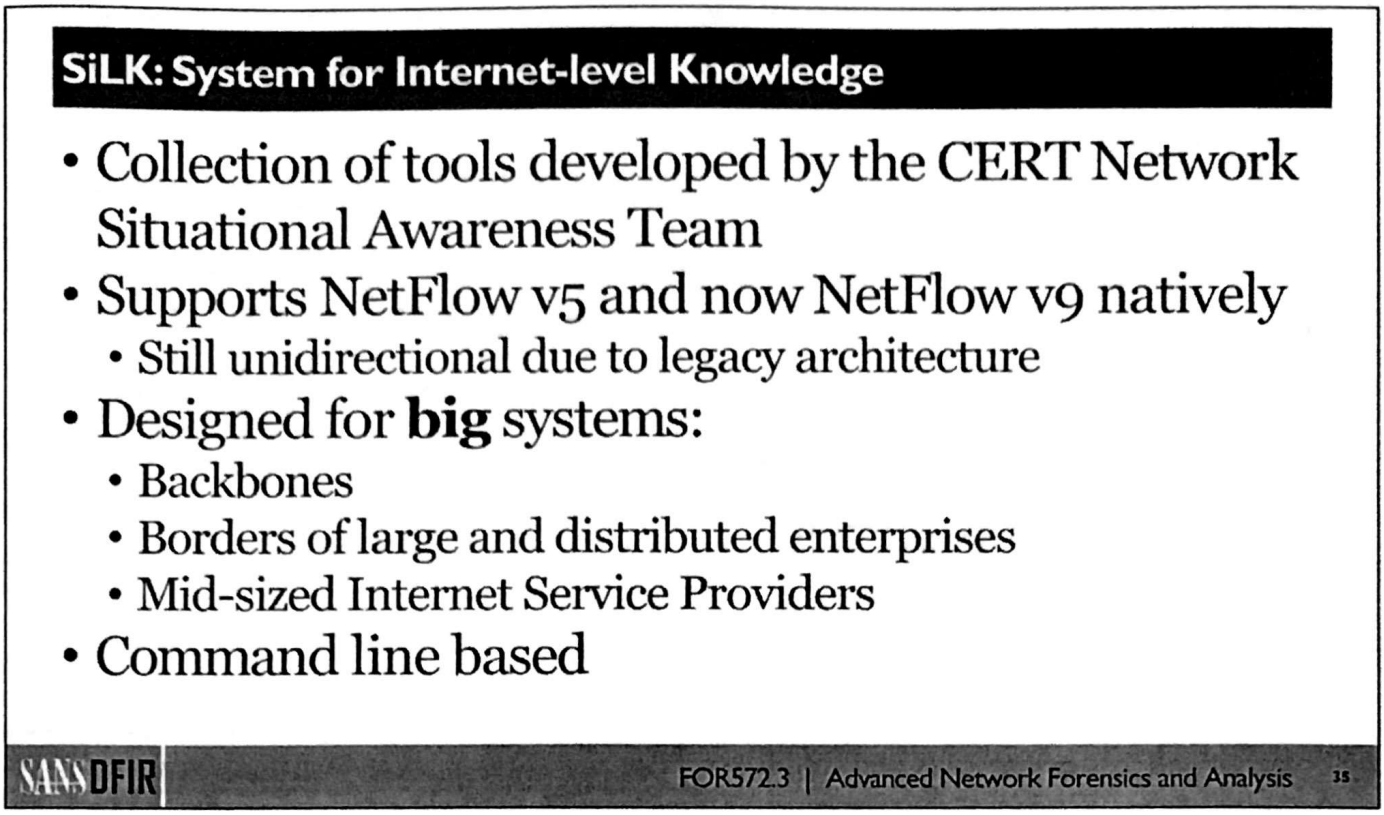
When analyzing any traffic without content (NetFlow record or encrypted traffic), an analyst must consider several factors to better characterize what occurred. These factors may include:

- The directionality of the network traffic (e.g., client to server or server to client): By understanding the typical direction of the traffic, an analyst can use basic NetFlow analysis techniques to determine how much data was transmitted from the attacker to the compromised system or from the compromised system to the attacker. This information helps to determine whether data theft occurred or the attacker transferred additional tools to the compromised machine. This can be complicated, however, because most NetFlow cannot track the originator of a bidirectional connection. Instead, we must often rely on well-known ports or endpoint corroboration to determine which endpoint is the "client".
- A basic understanding of the expected traffic associated with the underlying protocol: Through having a basic understanding of the protocol used, an analyst can determine deviations from normal conditions. For example, some protocols such as IPSec are very chatty, consistently sending synchronization information back and forth between the endpoints. However, SSL tunnels typically do not send as much synchronization information. Therefore a significant amount of SSL traffic between two endpoints is more likely to indicate high levels of activity than a similarly chatty IPsec connection. This notion doesn't apply only to encryption protocols, as discussed previously with regard to the multiple use cases for an SSH connection.
- An understanding of time-based analysis (including both frequency of activity and throughput): Through understanding frequency and throughput as it relates to a given network protocol, an analyst can detect spikes indicative of file transfers, blips that suggest data returned from commands issued by an attacker or even long-term periods of an attacker's dormancy.

By combining these factors, an analyst can typically determine a good deal about actions that caused network traffic to occur even without full-packet capture. This applies equally to NetFlow observations or examination of traffic over an encrypted tunnel.



This page intentionally left blank.



Now, we'll take a look at a few open-source tools that generate, receive, process, and analyze NetFlow data. First, we'll review the System for Internet-Level Knowledge, or SiLK.[1]

SiLK was originally conceived and designed by Suresh Konda Ph.D., with the intent to facilitate security analysis of large networks. It is a collection of UNIX/Linux command-line tools that manipulate the stored data collected from a number of NetFlow and IPFIX sensors. The tool set was developed and continues to be maintained by the CERT Network Situational Awareness Team (CERTNetSA) at Carnegie Mellon University.

It was designed with high-speed and high-bandwidth networks in mind. The designers have positioned it to support traffic analysis on Internet backbone links, borders of large, distributed enterprises or mid-sized Internet Service Providers. Although all modern distributions of the tool support NetFlow v5 and v9, its fundamental architecture still dates back to when NetFlow v5 was the only game in town. This means that even with bidirectional flows such as with NetFlow v9, SiLK can represent that data only as unidirectional flows.[2]

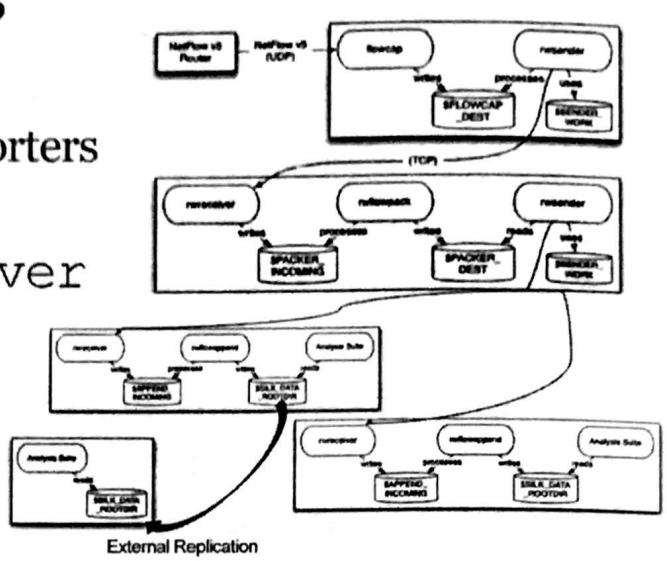
References:

[1] `http://for572.com/vgk01`

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

SiLK: Use with Remote Sites

- SiLK mitigates the UDP reliability problem
  - Place collectors near exporters
- Utilizes TCP from `rwsender` to `rwreceiver`
- Analysis conducted on local data at each tier
  - Allows different ACLs, data retention, etc.

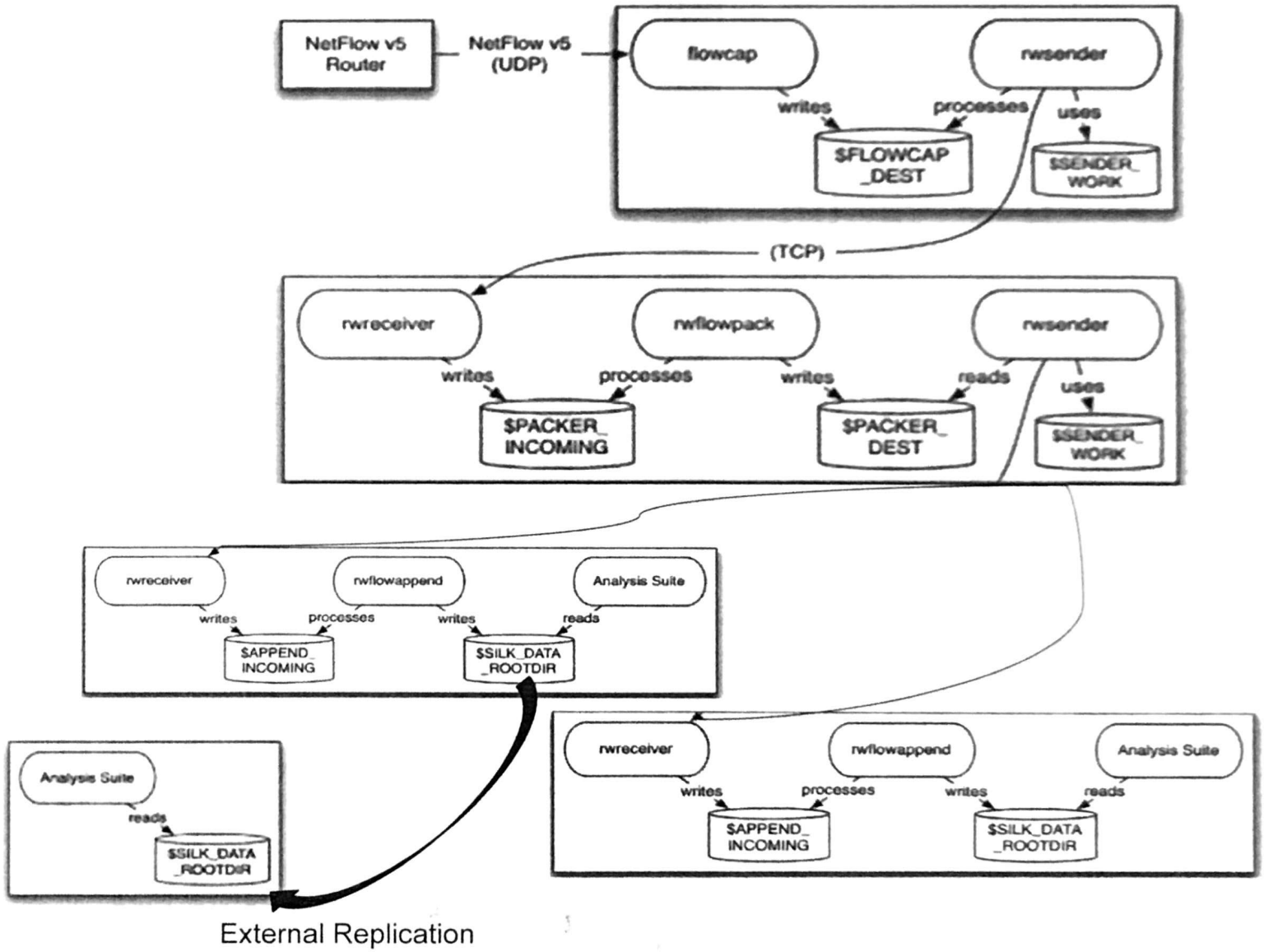


One excellent aspect of SiLK is that it transmits data from collectors to other nodes within the architecture. This mitigates the traditional problems associated with using the unreliable UDP transport mechanism. Although SiLK still captures data via UDP from the source exporter (using `flowcap`), it caches that data locally, and then the `rwsender` daemon transmits the data to another node via TCP. This is depicted in the first section of the diagram above.

At the next node, the user deploys the `rwreceiver` daemon to collect the TCP traffic and `rwflowpack` daemon to collect that TCP data and store it in the SiLK binary file format. If this node is not the final destination for those SiLK files, the `rwflowpack` tool can be configured to write the data in smaller files that `rwsender` can pick up and send to another end node (or nodes). This process is shown in the middle section of the diagram.

Finally, at the end storage node, an `rwreceiver` instance receives and saves the data for the `rwflowappend` daemon to write to the final location. The host operating system can replicate these files, which are stored on its native file system to additional analyst consoles as required. This final component is shown in the bottom part of the diagram.

Although complex, this demonstrates why SiLK can be scaled to handle enterprise scale deployments. Like most Unix/Linux tools, SiLK is modular enough to allow the user to implement only what they need to achieve the necessary results without the additional services, tools and components that often contribute to bloat with solutions built for other operating systems.



nfcapd / nfpcapd

- nfcapd receives NetFlow data from exporters
  - Receives NetFlow v5, v7, and v9
- Saves the data to regular, binary files
  - Filename shows start time: nfcapd.YYYYMMddhhmm
- Files rotate every five minutes (288/exporter/day)
  - Can be coalesced in post-processing
- Excellent for tactical capture during DFIR
  - Also suitable for small-to-mid-size long-term collectors
  - nfpcapd distills pcap to nfcapd NetFlow—use as first pass for pcap analysis

One component of the nfdump suite of tools is nfcapd. This utility is a daemon that listens on a UDP port for inbound NetFlow data from trusted exporters. Upon receiving data, nfcapd writes it to the configured file system location in a binary format. It collects NetFlow versions 5, 7, and 9.

It is often difficult to scope NetFlow storage requirements, but the generally accepted rule is to allow 1MB of NetFlow storage per 2GB of network traffic. As with any such scoping, this guideline should be adjusted for your environment based on real-world circumstances.

When writing NetFlow data, nfcapd uses a binary file format and filenames with the following filename structure:

```
nfcapd.YYYYMMDDhhmm
```

This naming scheme allows records to be sorted in directory listings simply by using their filenames. By default, nfcapd rotates files every 5 minutes, meaning it will generate 288 files ***per exporter*** per day.

Another interesting nfcapd feature is the "-R" option, which forwards flows as they are received to a second system. In the below example, nfcapd is listening on port 9227, writes NetFlow data from the system with IP address 10.0.1.1 to the /var/local/flow/router1/ directory, and simultaneously forwards those flows to the system at IP address 10.0.3.2, also on port 9227.

```
$ nfcapd -p 9227 -w -D -R 10.0.3.2/9227 ⏎
  -n router1,10.0.0.1,/var/local/flows/router1
```

Perhaps one of the most useful DFIR tools in the `nfdump` suite is the `nfpcapd` utility. (Note it's "`nfpcapd`" and not just "`nfcapd`"!) This utility will read from a pcap file, and then write `nfcapd`-compatible output files. This is an invaluable first step to take when provided with a large pcap collection. It permits the analyst to use NetFlow-based techniques, which are extremely fast, as a precursor to more computationally expensive processes involving the full content pcap file. The following example command reads the "`gigantic.pcap`" file and writes compressed output in date-hashed format to the `/cases/for572/netflow/` directory.

```
$ nfpcapd -r gigantic.pcap -S 1 -z -l /cases/for572/netflow/
```

nfdump

- Command line based
- Reads binary input from `nfcapd`
- Applies filters against collected NetFlow
  - `tcpdump`-like syntax (with some inconsistencies)
- Outputs the results in ASCII or binary
  - Binary files for further nfdump processing
  - Four stock ASCII formats: raw, line, long, extended
  - Custom ASCII formats also possible

Once the NetFlow data is collected into files using the `nfcapd` daemon or distilled from pcap using `nfpcapd`, the analyst can run queries against the data using the `nfdump` utility. `nfdump` is a command-line tool that reads and filters the `nfcapd` data files, presenting the results to the user in a flexible manner. Because `nfdump` provides a number of analytic features that can help the analyst to address the questions they have about the evidence, it is an excellent tool to use for fast characterization of network traffic.

`nfdump` is also extremely fast—an important feature when dealing with large files from fast networks, expansive NetFlow collection architectures, or long-term NetFlow historical archives. Whether addressing "live" NetFlow data that is still being written out or historical archives spanning months or years, query speed is a major factor.

`nfdump` reads the binary `nfcapd` file format, applies filters to the source data, then presents output in a variety of formats. The power of `nfdump` comes from the analyst's ability to build an iterative investigative process, in which the results from one query refine a hypothesis and then feed into the next query run against the data. As with most such tools, `nfdump`'s value is the result of a combination of the tool's inherent capabilities and the user's ability to leverage them in an analytic workflow. By itself, `nfdump` will not "find evidence" of unusual, suspicious, or outright malicious network activity.

We will review `nfdump`'s input and output functionality in the following slides.

nfdump: Input

- Reads from files or directories of `nfcapd` data
- Reads from single file

```
$ nfdump -r /var/netflow/site-rtr-1/2016/11/08/nfcapd.201611080635 ...
```

- Read recursively from directory tree
  - Recursively walks subdirectories—usually date-hashed

```
$ nfdump -R /var/netflow/site-rtr-1/2016/11/ ...
```


This slide shows the various input options that `nfdump` provides. The tool can read from a single file (which usually covers just five minutes), or from a directory full of files. This is a helpful mode to read data because `nfdump` will recursively navigate through directories as well.

Reading a single file is accomplished with the "`-r`" flag. In the following example, the command will read data from the "`site-rtr-1`" exporter on November 8, 2016 from 06:35 to 06:40 (collector local time).

```
$ nfdump -r /var/netflow/site-rtr-1/2016/11/08/nfcapd.201611080635 ...
```

Recall that `nfcapd` generally uses one directory for each NetFlow exporter. This allows an analyst to easily query data collected from just a single exporter using its designated directory tree. Because it is also common to store NetFlow data in a "date-hashed" directory structure, limiting by time frame is also easy. For example, the following command expands the command above to use all data from "`site-rtr-1`" during the entire month of November 2016.

```
$ nfdump -R /var/netflow/site-rtr-1/2016/11/ ...
```

nfdump: Filters

- Some of the powerful filters include:
  - S/D IP address: `host` – IPv4/6 or FQDN (performs DNS)
  - S/D Network: `net a.b.c.d/z` (CIDR notation)
  - Protocol: `proto` (`tcp` | `udp` | `icmp` | `gre` | `ah` | `<num>`)
  - S/D Port: `port <num>`
  - AS network: `as <num>`
- Directionality: `src port`, `dst host`, `dst as`
- Standard logic can be used to link expressions
  - `<expr> and <expr>`, `<expr> or <expr>`, `not <expr>`

The filters that `nfdump` provides are very flexible and easy to implement. This slide shows some of the more common filters, but for a full list, consult the "FILTER" section of the `nfdump`(1) man page. IP address fields (on a per-IP or CIDR block basis), layer four protocols and ports, as well as BGP autonomous system numbers (ASNs) can be used to build filters against collected NetFlow data. (Although many of these filters use the same syntax as `tcpdump`, it is important to remember they are not true BPFs.)

Many of these can be qualified by their directionality using the "`src`" or "`dst`" query modifiers. (As with BPFs, omitting the directionality modifier will match traffic that matches the filter in either direction.)

By combining simple filters with the logical parameters "and", "or", and "not", the analyst can build complex filter specifications. For example, in the command below, the analyst wants to identify the first ten systems that connected through the Internet-facing router (router1) to any Internet server on port 80 or 443, but did not use the internal web proxy (system 10.3.16.11) between noon (`nfcapd` local time!) on July 12, 2016 and noon on July 13, 2016.

```
$ nfdump -R /var/local/flows/router1/2016/07/ -t '2016/07/12.12:00:00-2016/07/13.12:00:00' ⏎
  -c 10 'proto tcp and (dst port 80 or dst port 443) and not src ip 10.3.16.11'
```

The following slides contain annotated example searches that reflect realistic analysis requirements you might face in the field.

nfdump: Output (1)

- "line" output format (default)

```
$ nfdump -R /var/log/flows/ -O tstart -o line 'host 36.249.80.226'
Date first seen          Duration Proto  Src IP Addr:Port       Dst IP Addr:Port   Packets Bytes Flows
2016-08-30 06:59:52.338     0.001 UDP    36.249.80.226:3040 -> 92.98.219.116:1434  1       404   1
```

- "long" format adds flags

```
$ nfdump -R /var/log/flows/ -O tstart -o long 'proto tcp and port 25'
Date first seen          Duration Proto Src IP Addr:Port      Dst IP Addr:Port   Flags    Tos Packets Bytes Flows
2016-08-30 06:53:53.370   63.545 TCP   113.138.32.152:25   -> 222.33.70.124:3575 .AP.SF   0      62  3512     1
2016-08-30 06:53:53.370   63.545 TCP   222.33.70.124:3575 -> 113.138.32.152:25  .AP.SF   0      58  3300     1
```

- "extended" format adds statistics

```
$ nfdump -R /var/log/flows/ -O tstart -o extended 'proto tcp and port 25'
Date first seen       Duration Proto Src IP Addr:Port      Dst IP Addr:Port   Flags Tos Packets Bytes pps bps Bpp Flows
2016-08-30 06:53:53.370 63.545 TCP   113.138.32.152:25   -> 222.33.70.124:3575 .AP.SF 0      62  3512   0 442  56  1
2016-08-30 06:53:53.370 63.545 TCP   222.33.70.124:3575 -> 113.138.32.152:25  .AP.SF 0      58  3300   0 415  56  1
```


The slide above shows the "line", "long", and "extended" formats for nfdump output.

Note that each line of output includes a single unidirectional NetFlow event—so a bidirectional communication channel is reflected by two flow records unless aggregation (which is discussed in a few slides) is used.

```
$ nfdump -R /var/log/flows/ -O tstart -o line 'host 36.249.80.226'
Date first seen          Duration Proto  Src IP Addr:Port       Dst IP Addr:Port   Packets Bytes Flows
2016-08-30 06:59:52.338     0.001 UDP    36.249.80.226:3040 -> 92.98.219.116:1434  1       404   1

$ nfdump -R /var/log/flows/ -O tstart -o long 'proto tcp and port 25'
Date first seen          Duration Proto Src IP Addr:Port      Dst IP Addr:Port   Flags    Tos Packets Bytes Flows
2016-08-30 06:53:53.370   63.545 TCP   113.138.32.152:25   -> 222.33.70.124:3575 .AP.SF   0      62  3512     1
2016-08-30 06:53:53.370   63.545 TCP   222.33.70.124:3575 -> 113.138.32.152:25  .AP.SF   0      58  3300     1

$ nfdump -R /var/log/flows/ -O tstart -o extended 'proto tcp and port 25'
Date first seen       Duration Proto Src IP Addr:Port      Dst IP Addr:Port   Flags Tos Packets Bytes pps bps Bpp Flows
2016-08-30 06:53:53.370 63.545 TCP   113.138.32.152:25   -> 222.33.70.124:3575 .AP.SF 0      62  3512   0 442  56  1
2016-08-30 06:53:53.370 63.545 TCP   222.33.70.124:3575 -> 113.138.32.152:25  .AP.SF 0      58  3300   0 415  56  1
```

nfdump: Output (2)

- Custom formats: Build report tables in the shell, use shell tools to do basic analytics

```
$ nfdump -R /var/log/flows/ -O tstart -o 'fmt:%pr %sap -> %dap %byt' 'proto tcp and port 25'
Proto  Src IP Addr:Port          Dst IP Addr:Port    Bytes
TCP    113.138.32.152:25   ->    222.33.70.124:3575     3512
TCP    222.33.70.124:3575  ->    113.138.32.152:25      3300
```

```
$ nfdump -R /var/log/flows/ -O tstart -o 'fmt:%ts %te %td %sa %da %dp'
Date first seen          Date last seen          Duration     Src IP Addr      Dst IP Addr Dst Pt
2016-03-12 23:58:59.386 2015-03-12 23:59:00.933   1.547   205.186.148.46    157.55.39.191  35449
2016-03-12 23:58:59.599 2015-03-12 23:59:00.322   0.723   205.186.148.46    72.78.204.217  50071
2016-03-12 23:59:00.116 2015-03-12 23:59:00.995   0.879        127.0.0.1        127.0.0.1  59783
2016-03-12 23:59:00.813 2015-03-12 23:59:01.220   0.407        127.0.0.1        127.0.0.1  53414
2016-03-12 23:59:01.851 2015-03-12 23:59:01.851   0.000   205.186.148.46     70.32.65.152     53
2016-03-12 23:59:02.351 2015-03-12 23:59:02.351   0.000     70.32.65.152   205.186.148.46  56879
```


The slide above shows two examples of `nfdump`'s custom format specification. The `nfdump` man page contains a list of all possible custom output tokens for each version of the utility.

Note that `nfdump` fully supports IPv6; however, it will truncate IPv6 addresses in its output for readability. To have the full IPv6 addresses appear in the output, simply use the "`line6`", "`long6`", or "`extended6`" output formats.

```
$ nfdump -R /var/log/flows/ -O tstart -o 'fmt:%pr %sap -> %dap %byt' 'proto tcp and port 25'
Proto  Src IP Addr:Port          Dst IP Addr:Port    Bytes
TCP    113.138.32.152:25   ->    222.33.70.124:3575     3512
TCP    222.33.70.124:3575  ->    113.138.32.152:25      3300

$ nfdump -R /var/log/flows/ -O tstart -o 'fmt:%ts %te %td %sa %da %dp'
Date first seen          Date last seen          Duration     Src IP Addr      Dst IP Addr Dst Pt
2016-03-12 23:58:59.386 2015-03-12 23:59:00.933   1.547   205.186.148.46    157.55.39.191  35449
2016-03-12 23:58:59.599 2015-03-12 23:59:00.322   0.723   205.186.148.46    72.78.204.217  50071
2016-03-12 23:59:00.116 2015-03-12 23:59:00.995   0.879        127.0.0.1        127.0.0.1  59783
2016-03-12 23:59:00.813 2015-03-12 23:59:01.220   0.407        127.0.0.1        127.0.0.1  53414
2016-03-12 23:59:01.851 2015-03-12 23:59:01.851   0.000   205.186.148.46     70.32.65.152     53
2016-03-12 23:59:02.351 2015-03-12 23:59:02.351   0.000     70.32.65.152   205.186.148.46  56879
```

nfdump: Example Scenario (1)

- Malware analysis indicates C2 on UDP/8765 with "apt.domain.com" (174.37.46.52)
  - First 5 of the day (UTC) :

```
$ nfdump -r /var/local/flows/nfcapd.201603120000 -O tstart -c 5 ⏎
    'proto udp and dst port 8765 and host 174.37.46.52'
Date first seen          Duration Proto   Src IP Addr:Port          Dst IP Addr:Port  Packets  Bytes Flows
[redacted]                  0.000 UDP      10.3.58.6:45524   ->   174.37.46.52:8765        1     78     1
2016-03-12 00:23:01.356     0.000 UDP      10.3.59.76:33597  ->   174.37.46.52:8765        1     76     1
2016-03-12 01:18:01.746     0.000 UDP      10.3.58.85:57002  ->   174.37.46.52:8765        1     78     1
2016-03-12 01:36:01.851     0.000 UDP      10.3.58.7:38306   ->   174.37.46.52:8765        1     53     1
2016-03-12 02:12:02.256     0.000 UDP      10.3.58.4:32833   ->   174.37.46.52:8765        1     53     1
```

  - First one last night (1800-0900 UTC):

```
$ nfdump -R /var/local/flows/ -t '2016/03/11.18:00:00-2016/03/12.09:00:00' -O tstart -c 1 ⏎
     'proto udp and dst port 8765 and host 174.37.46.52'
Date first seen          Duration Proto   Src IP Addr:Port          Dst IP Addr:Port  Packets  Bytes Flows
[redacted]                  0.000 UDP      10.3.58.6:11574   ->   174.37.46.52:8765        1    750     1
```


Consider an IP address that was identified through malware analysis (or threat intelligence, or some other means) as suspicious or malicious: 174.37.46.52. Evidence shows that traffic over UDP port 8765 is of interest and the analyst wishes to use NetFlow to scope the clients that have exhibited this behavior.

In the top example, he runs `nfdump` on a 1-day capture from March 12, 2016 (indicated by the filename and confirmed by examining the file's contents). He asks for the first 5 hosts that sent packets on the suspected protocol and port.

```
$ nfdump -r /var/local/flows/nfcapd.201603120000 -O tstart -c 5 ⏎
  'proto udp and dst port 8765 and host 174.37.46.52'
Date first seen          Duration Proto   Src IP Addr:Port          Dst IP Addr:Port  Packets  Bytes Flows
2016-03-12 00:09:01.356     0.000 UDP      10.3.58.6:45524   ->   174.37.46.52:8765        1     78     1
2016-03-12 00:23:01.356     0.000 UDP      10.3.59.76:33597  ->   174.37.46.52:8765        1     76     1
2016-03-12 01:18:01.746     0.000 UDP      10.3.58.85:57002  ->   174.37.46.52:8765        1     78     1
2016-03-12 01:36:01.851     0.000 UDP      10.3.58.7:38306   ->   174.37.46.52:8765        1     53     1
2016-03-12 02:12:02.256     0.000 UDP      10.3.58.4:32833   ->   174.37.46.52:8765        1     53     1
```

He's seeking to identify "patient zero"—the first host that connected to the suspicious external IP address on the specified protocol and port. The first event was just nine minutes into the time covered by the source file, and the interval between subsequent events was typically much wider than nine minutes—it's reasonable to assume at this point that the traffic matching this behavior may have existed prior to March 12. Therefore, he expands his view of the evidence to cover all of last night at 1800 to this morning at 0900 (per the collector system's local time—UTC, of course).

```
$ nfdump -R /var/local/flows/ -t '2016/03/11.18:00:00-2016/03/12.09:00:00' -O tstart -c 1 ⏎
    'proto udp and dst port 8765 and host 174.37.46.52'
Date first seen          Duration Proto   Src IP Addr:Port          Dst IP Addr:Port  Packets  Bytes Flows
2016-03-11 22:04:48.411     0.000 UDP      10.3.58.6:11574   ->   174.37.46.52:8765        1    750     1
```

Because the first instance of this behavior was over four hours into the window covered by the evidence, it's a reasonable hypothesis that this is the "first" event matching this traffic profile.

nfdump: Example Scenario (2)

- What else potential "patient zero" did:

```
$ nfdump -R /var/local/flows/ -t '2015/03/11.18:00:00-2015/03/12.09:00:00' 'host 10.3.58.6'
Date first seen          Duration Proto   Src IP Addr:Port        Dst IP Addr:Port  Packets  Bytes Flows
2015-03-11 22:00:16.532   24.581 TCP      10.3.58.6:10581  ->     174.37.46.94:80        81   4281     1
2015-03-11 22:00:16.532   24.581 TCP      174.37.46.94:80  ->     10.3.58.6:10581       547 812916     1
2015-03-11 22:04:48.411    0.000 UDP      10.3.58.6:11574  ->     174.37.46.52:8765       1    750     1
...
```

- Determine common Autonomous System, identify connections with that provider all month:

```
$ whois 174.37.46.94|grep AS
OriginAS:       AS36351

$ whois 174.37.46.52|grep AS
OriginAS:       AS36351
```

```
$ nfdump -q -R /var/local/flows/ -t '2015/03/01-2015/04/01' ⏎
    -o 'fmt:%sa %da' 'dst as 36351' | sort | uniq
  10.3.57.104    174.37.150.67
   10.3.58.61      174.36.84.1
   10.3.58.83     174.36.51.31
    10.3.58.6     174.37.46.94
...
```


Now, the analyst wishes to find out what else the apparent patient zero did, and expands the same time frame of a search to cover the entire set of traffic to or from that internal IP address. The observed flows may suggest that an HTTP download preceded the potential C2 activity by a just a few minutes.

```
$ nfdump -R /var/local/flows/ -t '2015/03/11.18:00:00-2015/03/12.09:00:00' 'host 10.3.58.6'
Date first seen          Duration Proto   Src IP Addr:Port        Dst IP Addr:Port  Packets  Bytes Flows
2015-03-11 22:00:16.532   24.581 TCP      10.3.58.6:10581  ->     174.37.46.94:80        81   4281     1
2015-03-11 22:00:16.532   24.581 TCP      174.37.46.94:80  ->     10.3.58.6:10581       547 812916     1
2015-03-11 22:04:48.411    0.000 UDP      10.3.58.6:11574  ->     174.37.46.52:8765       1    750     1
...
```

Finally, the analyst notices the close proximity between the two IP addresses identified above. Using a tool such as WHOIS, the analyst identifies that both IP addresses are part of the same autonomous system with ASN 36351, a particular hosting provider.

```
$ whois 174.37.46.94|grep AS
OriginAS:       AS36351

$ whois 174.37.46.52|grep AS
OriginAS:       AS36351
```

With an operating hypothesis that the attacker may have additional infrastructure with this hosting provider, he looks for IP addresses that have accessed systems within this particular autonomous system, or "Internet neighborhood". This list would be of use in determining if any further investigative action is necessary on the identified internal hosts.

```
$ nfdump -q -R /var/local/flows/ -t '2015/03/01-2015/04/01' -o 'fmt:%sa %da' 'dst as 36351' | ⏎
    sort | uniq
  10.3.57.104    174.37.150.67
   10.3.58.61      174.36.84.1
   10.3.58.83     174.36.51.31
    10.3.58.6     174.37.46.94
...
```

nfdump: Aggregating Flows

- Aggregate flows for better statistics (`-a`)
  - "`-a`": Five flow keys: S/D IP addresses, protocol, S/D ports

```
$ nfdump -r nfcapd.201605300000 -a -O tstart
Date first seen          Duration Proto Src IP Addr:Port          Dst IP Addr:Port    Packets  Bytes Flows
2016-05-29 23:54:41.647    342.994 TCP  205.186.148.167:443     ->  66.249.66.156:45326     411 537757     1
2016-05-29 23:54:41.647    343.004 TCP    66.249.66.156:45326   ->  205.186.148.167:443     248  20174     1
2016-05-29 23:54:48.069  86545.814 ICMP   198.46.150.48:0       ->   205.186.148.46:8.0    1433 131836   249
2016-05-29 23:54:48.069  86545.814 ICMP   205.186.148.46:0      ->    198.46.150.48:0.0    1433 131836   249
2016-05-29 23:56:12.119   2109.580 TCP    205.186.148.46:443    -> 72.219.212.172:51296     189  96711     6
2016-05-29 23:56:12.119   2109.580 TCP    72.219.212.172:51296  ->   205.186.148.46:443     144  61374     6
```

- "`-A`": Custom aggregation

```
$ nfdump -q -R /var/local/flows/ -O tstart -t '2013/07/01-2013/07/18' -A srcip,dstport ⏎
  -o 'fmt:%sa -> %dp %byt %fl'
   188.165.15.27 ->     80     2291     1
  73.189.228.129 ->     80     4123     1
  205.186.148.46 ->    443    44077     4
  205.186.148.46 ->     53      697    11
```


Because of the 5-minute interval `nfcapd` uses per file by default as well as the various ways in which NetFlow exporters determine a flow's end time, `nfdump` may report a single flow in multiple records. To coalesce these records back into a single event, `nfdump` provides the means to aggregate records based on specified traffic characteristics.

The default aggregation uses the five key flow values: source and destination IP addresses, layer four protocol, and source and destination ports. Any flow records that share the same values for these five fields will be aggregated together into a single result record, totaling the packets, bytes, flows, etc. and calculating other values like time window accordingly.

The example shown reflects an aggregation from a single day of traffic. The first two lines show a single connection (one flow in each direction) between these two hosts lasting just under six minutes. The third and fourth records show ICMP flows with a "duration" of the entire day. This is due to the inherent lack of discrete end state for ICMP traffic. The underlying activity here is a "ping" every five minutes (also see in the flow count of 249) for availability monitoring purposes. However, because this exporter's ICMP timeout is set to a value greater than five minutes, the overall "flow" as determined by the exporter never actually ends. Similarly, the last two records show a flow event approximately 30 minutes in duration and 6 flows in each direction. This was webmail activity, in which the HTTPS socket is kept open by AJAX requests.

```
$ nfdump -r nfcapd.201605300000 -a -O tstart
Date first seen   Duration Proto Src IP Addr:Port  Dst IP Addr:Port  Packets  Bytes Flows
2016-05-29 23:54:41.647   342.994 TCP  205.186.148.167:443    ->  66.249.66.156:45326     411   537757     1
2016-05-29 23:54:41.647   343.004 TCP    66.249.66.156:45326  ->  205.186.148.167:443     248    20174     1
2016-05-29 23:54:48.069 86545.814 ICMP   198.46.150.48:0      ->   205.186.148.46:8.0    1433   131836   249
2016-05-29 23:54:48.069 86545.814 ICMP   205.186.148.46:0     ->    198.46.150.48:0.0    1433   131836   249
2016-05-29 23:56:12.119  2109.580 TCP    205.186.148.46:443   -> 72.219.212.172:51296     189    96711     6
2016-05-29 23:56:12.119  2109.580 TCP    72.219.212.172:51296 ->   205.186.148.46:443     144    61374     6
```

The `nfdump` utility also provides custom aggregation capabilities. This is especially helpful to populate a report table with just specific values, or to use shell primitives for a fast and effective way to identify patterns from broad collections. For example, a good method to identify port scanning activity would be to aggregate solely by source IP and destination port would be "**`nfdump -A srcip,dstport`**". This works because when a scanner is hunting for open ports, the source port usually changes with each new connection, but the target port (say, TCP/22 for SSH) remains constant. By aggregating in this way, we can identify source systems with high connections attempt counts destined for static, well-known port numbers (not just 22).

The command below looks in all flow directories under "`/var/local/flows/`", and in the files covering the period between July 1, 2016 and July 17, 2016. It will aggregate all flows sharing the same source IP and destination port to a single output record. The analyst has requested a custom output format, displaying the following fields: (source IP address) -> (destination IP port) (bytes sent) (number of flows).

```
$ nfdump -q -R /var/local/flows/ -O tstart -t '2016/07/01-2016/07/18' -A srcip,dstport ⏎
  -o 'fmt:%sa -> %dp %byt %fl'
   188.165.15.27 ->     80     2291     1
  73.189.228.129 ->     80     4123     1
  205.186.148.46 ->    443    44077     4
  205.186.148.46 ->     53      697    11
```

nfdump: Bidirectional Aggregation

- Auto-merge flows into bidirectional view (-b)

```
$ nfdump -R /var/local/flows/ -O tstart -b -o 'fmt:%ts %pr %sap %dap %ibyt %obyt'
Date first seen          Proto      Src IP Addr:Port      Dst IP Addr:Port  In Byte Out Byte
2015-03-12 23:58:28.136 TCP      205.186.148.46:80   183.252.52.11:32954     65261     2098
2015-03-12 23:58:48.013 TCP    123.125.71.111:37341   205.186.147.131:80       490        0
2015-03-12 23:58:56.299 TCP    189.251.16.177:51028    205.186.148.46:80       268      172
2015-03-12 23:58:59.386 TCP      205.186.148.46:80   157.55.39.191:35449     62430     1399
2015-03-12 23:58:59.599 TCP      205.186.148.46:80    72.78.204.217:50071       930     3461
2015-03-12 23:59:01.299 TCP      205.186.148.46:80 205.186.148.46:45493       347      328
```

- Port-based directionality assumption (-B)

```
$ nfdump -R /var/local/flows/ -O tstart -B -o 'fmt:%ts %pr %sap %dap %ibyt %obyt'
Date first seen          Proto      Src IP Addr:Port      Dst IP Addr:Port  In Byte Out Byte
2015-03-12 23:58:28.136 TCP    183.252.52.11:32954   205.186.148.46:80       2098     65261
2015-03-12 23:58:48.013 TCP    123.125.71.111:37341   205.186.147.131:80       490        0
2015-03-12 23:58:56.299 TCP    189.251.16.177:51028    205.186.148.46:80       268      172
2015-03-12 23:58:59.386 TCP    157.55.39.191:35449    205.186.148.46:80      1399     62430
2015-03-12 23:58:59.599 TCP     72.78.204.217:50071    205.186.148.46:80      3461      930
2015-03-12 23:59:01.299 TCP  205.186.148.46:45493    205.186.148.46:80       328       347
```


A known difficulty with interpreting `nfcapd`-collected NetFlow evidence is that each flow is represented as unidirectional—meaning a typical traffic exchange is represented to the user as two flows. The `nfdump` utility provides a solution for this, however, and can automatically merge flows into their bidirectional representations. This is accomplished with one of two command line switches, "`-b`" and "`-B`".

To automatically perform bidirectional aggregation with no assumption for the initiating IP address, use the "`-b`" switch. In this mode, `nfdump` will use whatever record appears first in the binary input file as the client-to-server side of the conversation.

```
$ nfdump -R /var/local/flows/ -O tstart -b -o 'fmt:%ts %pr %sap %dap %ibyt %obyt'
Date first seen          Proto      Src IP Addr:Port      Dst IP Addr:Port  In Byte Out Byte
2015-03-12 23:58:28.136 TCP      205.186.148.46:80   183.252.52.11:32954     65261     2098
2015-03-12 23:58:48.013 TCP    123.125.71.111:37341   205.186.147.131:80       490        0
2015-03-12 23:58:56.299 TCP    189.251.16.177:51028    205.186.148.46:80       268      172
2015-03-12 23:58:59.386 TCP      205.186.148.46:80   157.55.39.191:35449     62430     1399
2015-03-12 23:58:59.599 TCP      205.186.148.46:80    72.78.204.217:50071       930     3461
2015-03-12 23:59:01.299 TCP      205.186.148.46:80 205.186.148.46:45493       347      328
```

However, with the "`-B`" switch, `nfdump` will assume the initiating IP by assuming a port below 1024 belongs to the server. If both the client and the server use high ports, `nfdump` reverts back to "`-b`" behavior for that particular connection.

```
$ nfdump -R /var/local/flows/ -O tstart -B -o 'fmt:%ts %pr %sap %dap %ibyt %obyt'
Date first seen          Proto      Src IP Addr:Port      Dst IP Addr:Port  In Byte Out Byte
2015-03-12 23:58:28.136 TCP    183.252.52.11:32954   205.186.148.46:80       2098     65261
2015-03-12 23:58:48.013 TCP    123.125.71.111:37341   205.186.147.131:80       490        0
2015-03-12 23:58:56.299 TCP    189.251.16.177:51028    205.186.148.46:80       268      172
2015-03-12 23:58:59.386 TCP    157.55.39.191:35449    205.186.148.46:80      1399     62430
2015-03-12 23:58:59.599 TCP     72.78.204.217:50071    205.186.148.46:80      3461      930
2015-03-12 23:59:01.299 TCP  205.186.148.46:45493    205.186.148.46:80       328       347
```

nfdump: "TopN" Statistics

- Command line syntax: `-s stat[/order]`
- Count by : `proto`, `ip`, `port`, `as`, `src*`, `dst*`, more
- Order by: `flows`, `packets`, `bytes`, `pps`, `bps`, `bpp`

```
$ nfdump -R /var/local/flows/ -t '2016/10/01-2016/11/01' -s ip/bytes -s dstport/bytes
Top 5 IP Addr ordered by bytes:
Date first seen          Duration    Proto        IP Addr    Flows(%)     Packets(%)       Bytes(%) pps    bps  bpp
2016-10-06 23:56:26.395 1669042.445 any         127.0.0.1  4.9 M(17.0)   1.1 G(72.0)    4.4 T(93.1) 650 21.1 M 4056
2016-10-10 23:55:40.001 1669113.643 any   205.186.148.46 18.4 M(63.6) 238.1 M(15.8) 172.4 G( 3.6) 142 826138  724
2016-10-01 23:58:55.136 1668894.058 any   205.186.148.167  5.2 M(18.1) 177.4 M(11.8) 151.3 G( 3.2) 106 725126  852
2016-10-05 00:30:48.938 1642988.710 any    75.101.135.185    524( 0.0)   11.9 M( 0.8)   14.5 G( 0.3)   7  70474 1221

Top 5 Src Port ordered by bytes:
Date first seen          Duration    Proto      Dst Port    Flows(%)     Packets(%)       Bytes(%) pps    bps  bpp
2016-10-02 23:58:52.923 1668895.917 any            3306  1.6 M( 5.5) 560.7 M(37.2)    4.3 T(90.5) 335 20.5 M 7638
2016-10-06 23:58:55.136 1668894.058 any             443  2.9 M(10.1) 119.3 M( 7.9) 150.6 G( 3.2)  71 722078 1262
2016-10-09 23:57:54.858 1668954.925 any              80  3.3 M(11.3)  99.4 M( 6.6) 125.8 G( 2.7)  59 602883 1265
2016-10-10 23:56:40.862 1669052.782 any             993 235208( 0.8)   11.3 M( 0.8)   22.9 G( 0.5)   6 109703 2025
```


Network engineers often use the "TopN" statistics output to see who is using the most bandwidth. However, it has other capabilities when used in an investigation and driven by intelligence. If we know about malicious domains, IP addresses, or C2 communication ports, the analyst can quickly identify systems that warrant additional attention. For example, you may want to know what systems have:

- Communicated the longest (the earliest "first seen"): Indicates first machines compromised
- Sent the most traffic (large bytes sent/received): Indicates they may have been used as staging systems or may be the source of data loss
- Connected to the most hosts (highest number of flows): May show "jump point" scanners

`nfdump`'s TopN statistics can be ordered by a number of fields by using the "**`-s statistic[/orderby]`**" command line option.

The following is a subset of valid "`statistic`" values:

`proto`	-	Protocol numbers
`record`	-	Aggregated NetFlow record count
`ip`	-	Any (source or destination) IP addresses
`srcip`	-	Source IP addresses
`dstip`	-	Destination IP addresses
`Port`	-	Any (source or destination) ports
`srcport`	-	Source ports
`dstport`	-	Destination ports
`as`	-	Any (source or destination) AS numbers
`srcas`	-	Source AS numbers
`dstas`	-	Destination AS numbers
`if`	-	Any (input or output) interface numbers
`inif`	-	Input interface numbers
`outif`	-	Output interface numbers

The statistic reports can be sorted with the following "`orderby`" values:

`flows`	-	Number of flow records
`packets`	-	Number of packets
`bytes`	-	Total bytes
`pps`	-	Packets per second
`bps`	-	Bytes per second
`bpp`	-	Bytes per packet

```
$ nfdump -R /var/local/flows/ -t '2016/10/01-2016/11/01' -s ip/bytes -s dstport/bytes
Top 5 IP Addr ordered by bytes:
Date first seen          Duration    Proto        IP Addr    Flows(%)      Packets(%)        Bytes(%) pps    bps  bpp
2016-10-06 23:56:26.395 1669042.445 any       127.0.0.1    4.9 M(17.0)    1.1 G(72.0)    4.4 T(93.1) 650 21.1 M 4056
2016-10-10 23:55:40.001 1669113.643 any  205.186.148.46   18.4 M(63.6)  238.1 M(15.8)  172.4 G( 3.6) 142 826138  724
2016-10-01 23:58:55.136 1668894.058 any  205.186.148.167   5.2 M(18.1)  177.4 M(11.8)  151.3 G( 3.2) 106 725126  852
2016-10-05 00:30:48.938 1642988.710 any   75.101.135.185    524( 0.0)   11.9 M( 0.8)   14.5 G( 0.3)   7  70474 1221

Top 5 Src Port ordered by bytes:
Date first seen          Duration    Proto     Dst Port    Flows(%)      Packets(%)        Bytes(%) pps    bps  bpp
2016-10-02 23:58:52.923 1668895.917 any           3306   1.6 M( 5.5)  560.7 M(37.2)    4.3 T(90.5) 335 20.5 M 7638
2016-10-06 23:58:55.136 1668894.058 any            443   2.9 M(10.1)  119.3 M( 7.9)  150.6 G( 3.2)  71 722078 1262
2016-10-09 23:57:54.858 1668954.925 any             80   3.3 M(11.3)   99.4 M( 6.6)  125.8 G( 2.7)  59 602883 1265
2016-10-10 23:56:40.862 1669052.782 any            993 235208( 0.8)   11.3 M( 0.8)   22.9 G( 0.5)   6 109703 2025
```

SOF-ELK: NetFlow Collector and Analytic Platform

- SOF-ELK is also a NetFlow analysis platform
  - Receives live NetFlow v5 via UDP
  - Loads archived NetFlow parsed to ASCII
- Enriches all IP addresses with GeoIP and ASN
- Includes Kibana NetFlow dashboard
- Extendable via Kibana visualizations

Although command-line NetFlow analysis is a critical skill that we'll use extensively throughout the rest of the class, the sheer volume of data often means a graphical tool can be a benefit to the analyst. The SOF-ELK platform was designed to provide this functionality. As with all other evidence, SOF-ELK will ingest both live and archived NetFlow data. Live NetFlow is consumed from a UDP port while archived NetFlow is read from specifically formatted ASCII text files placed onto its filesystem.

Regardless of the ingest method, SOF-ELK will enrich all NetFlow, including the addition of the ASN and standard geolocation data for each IP address. This opens up the opportunity for both the provided and custom NetFlow dashboards to use mapping visualizations. SOF-ELK's built-in NetFlow dashboard gives the analyst a wealth of insight to data that has been loaded.

SOF-ELK: Loading NetFlow

- Live collector:
  - Open firewall port UDP/9995 on VM
  - Point exporters or traffic splitter(s) to SOF-ELK
- Loading archived NetFlow:
  - Use `nfdump` command with output format string
    - Set exporter IP ("`0.0.0.0`" in example)

```
$ nfdump (-r <input file> | -R <input dir>) -q -N -O tstart -o "fmt:0.0.0.0 %das %dmk %eng %ts %fl 0 %byt
  %pkt %in %da %nh %sa %dp %sp %te %out %pr 0 0 %sas %smk %stos %flg 0" > ⏎
  /cases/for572/netflow-export.txt
$ nfdump (-r <input file> | -R <input dir>) -q -N -O tstart -o "fmt:0.0.0.0 $NFDUMP2SOFELK" > ⏎
  /cases/for572/netflow-export.txt
```

  - Place ASCII file in `/usr/local/logstash-nfarch/`

Loading live NetFlow is very simple and straightforward. First, ensure SOF-ELK has an accessible network interface (e.g., "Bridged" in VMware). Then, open the firewall port using the command below, and finally configure NetFlow exporters or traffic splitters to send their NetFlow to the SOF-ELK's IP address on UDP/9995.

```
$ sudo firewall-cmd --zone=public --add-port=995/udp --permanent
```

Loading archived NetFlow takes a little more work. SOF-ELK requires that the NetFlow data be in a specific ASCII format, which is then loaded from the filesystem. To mirror Logstash's native NetFlow features, the format `nfdump` format string shown here is required. Note that the IP address shown as "`0.0.0.0`" is a field that indicates the NetFlow exporter. It can be set to any value as long as it fits a valid IPv4 address, but it must be present or the data will not load. In the future, loading nfcapd-created NetFlow evidence will be accomplished through an automated process. This will be published to and documented in the SOF-ELK Github repository.[1]

```
$ nfdump (-r <input file> | -R <input dir>) -q -N -O tstart ⏎
  -o "fmt:0.0.0.0 %das %dmk %eng %ts %fl 0 %byt %pkt %in %da %nh %sa %dp %sp
  %te %out %pr 0 0 %sas %smk %stos %flg 0" > /cases/for572/netflow-export.txt
```

On the custom FOR572 version of the SANS SIFT workstation, this complex format string has been simplified by the use of a bash environment variable, NFDUMP2SOFELK. The following command provides the same results from the FOR572 SIFT with fewer opportunities for typos:

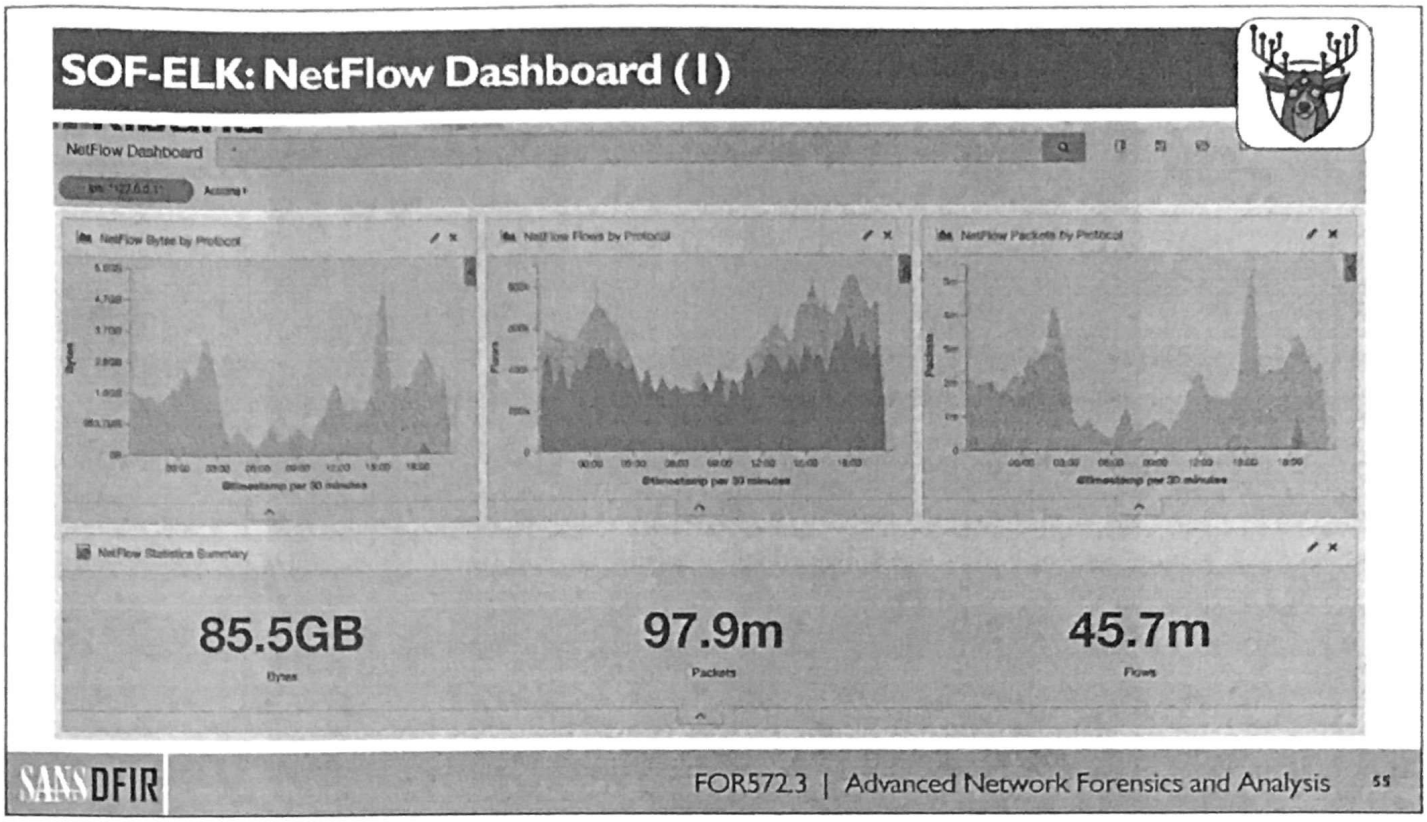
```
$ nfdump (-r <input file> | -R <input dir>) -q -N ⏎
  -O tstart -o "fmt:0.0.0.0 $NFDUMP2SOFELK" > ⏎
  /cases/for572/netflow-export.txt
```

After creating the ASCII file ("`/cases/for572/netflow-export.txt`" in this example), ensure filesystem permissions are properly set and place the file into the "`/usr/local/logstash-nfarch/`" directory on the SOF-ELK VM's filesystem.

Note: The `nfdump` output format includes fields that match Logstash's NetFlow codec. Some of these are not available from `nfdump` and are replaced with hard-coded "`0`" values.

References:

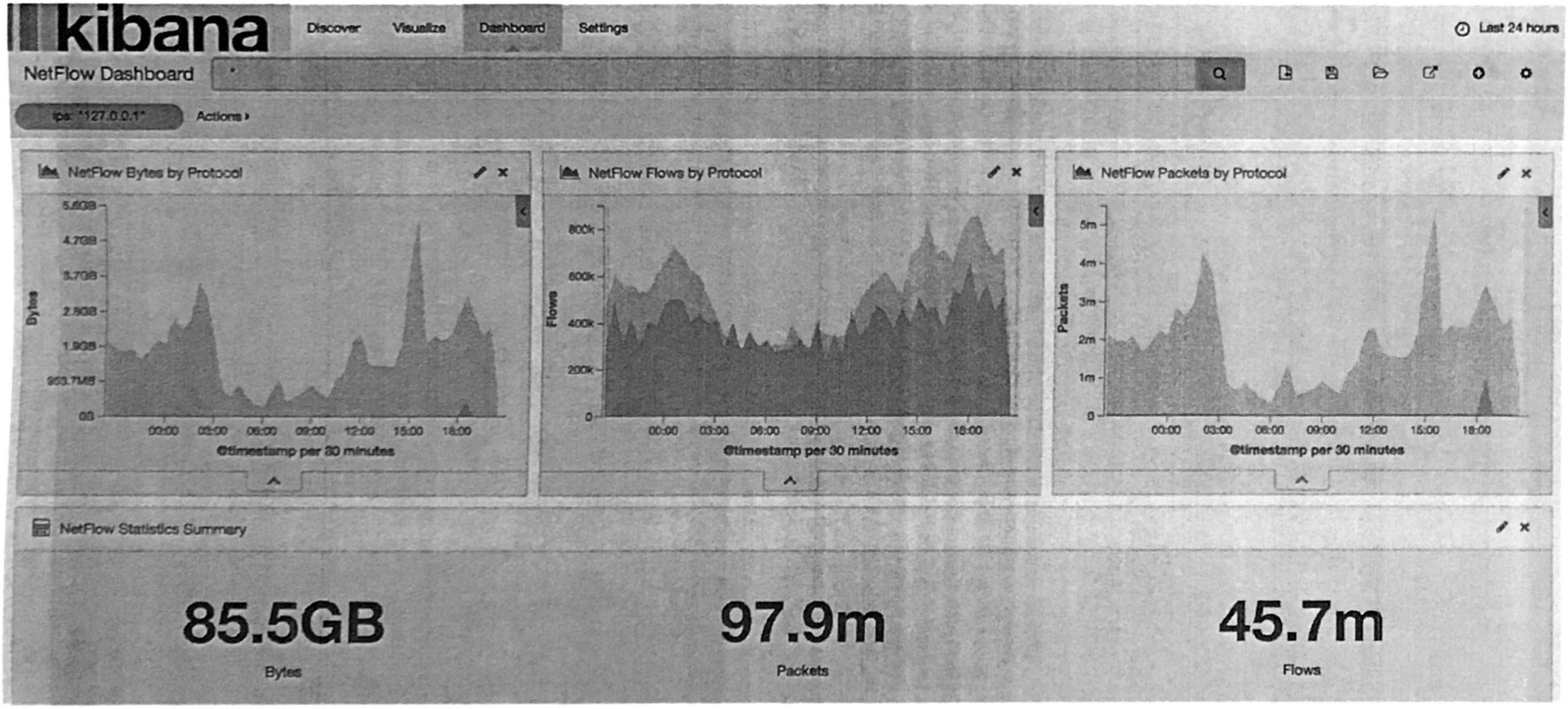
[1] `http://for572.com/sof-elk-git`

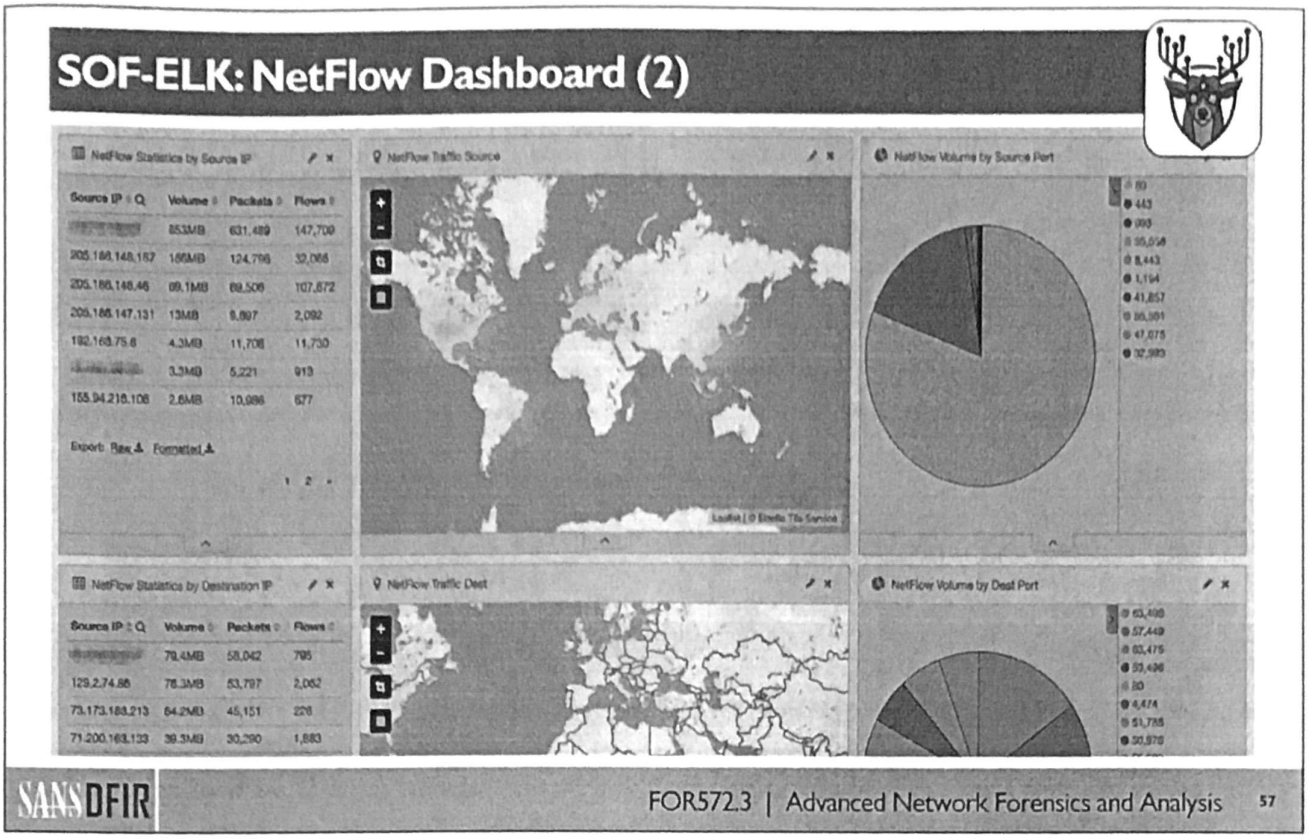


This is the top of SOF-ELK's supplied NetFlow dashboard. Traffic is broken down by byte, packet, and flow count. This mirrors the basic metrics provided by nfdump and many other NetFlow platforms.

The three-pronged approach gives the analyst a visual means of identifying events of interest. For example, beaconing traffic would generally reflect a small byte count and a high packet count as a result of numerous small transmissions. A typical data theft would be reflected by a low flow count and high byte and packet counts.

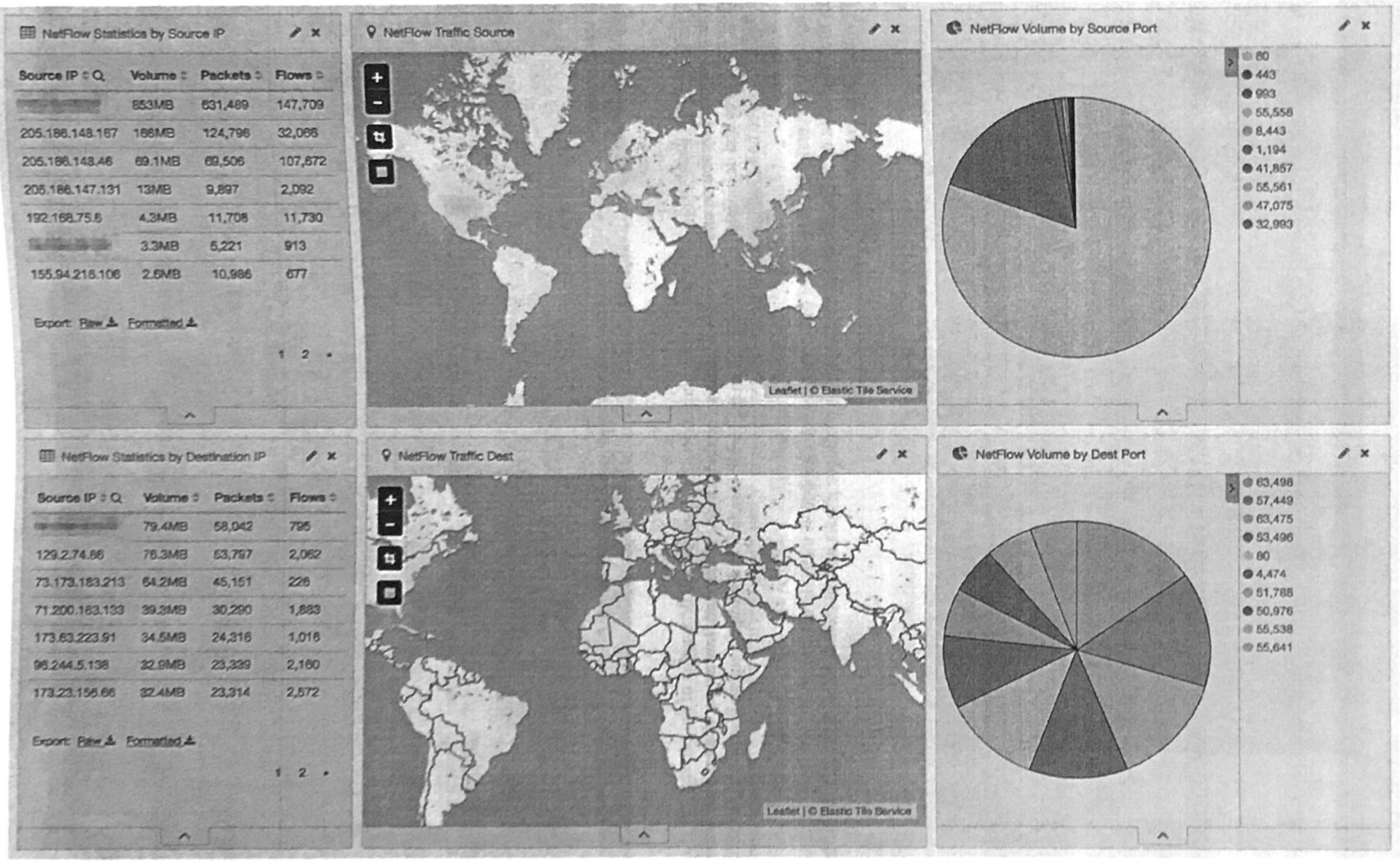
The summary portion of the dashboard shown is a helpful starting point and of course the three panels can be used in click-and-drag fashion to select new timeframes and zero in on spikes or troughs of interest.

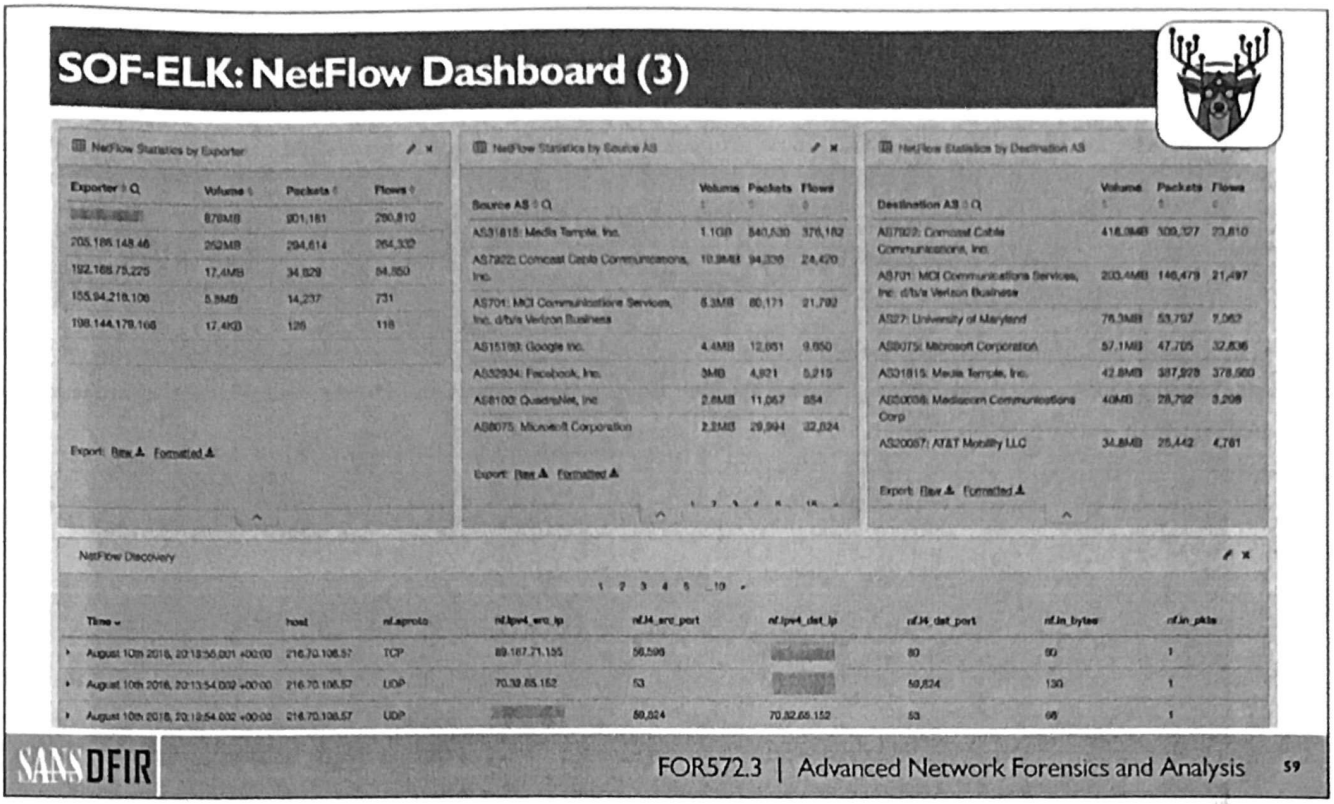




Scrolling down the dashboard, the analyst can see summaries of the source and destination of the currently in scope. Traffic is reported by source and destination IP address, again by byte, packet, and flow count. The heatmap panels reflect the source and destination of traffic, based on the GeoIP enrichment done upon ingest. The pie graphs show the traffic's source and destination port breakdowns.

It's again worth mentioning that each of these visualization elements is interactive. Clicking any row in IP address lists, drawing a box on the heatmaps, or clicking slices on the pie graphs will all create and apply tunable filters to the contents returned from Elasticsearch.

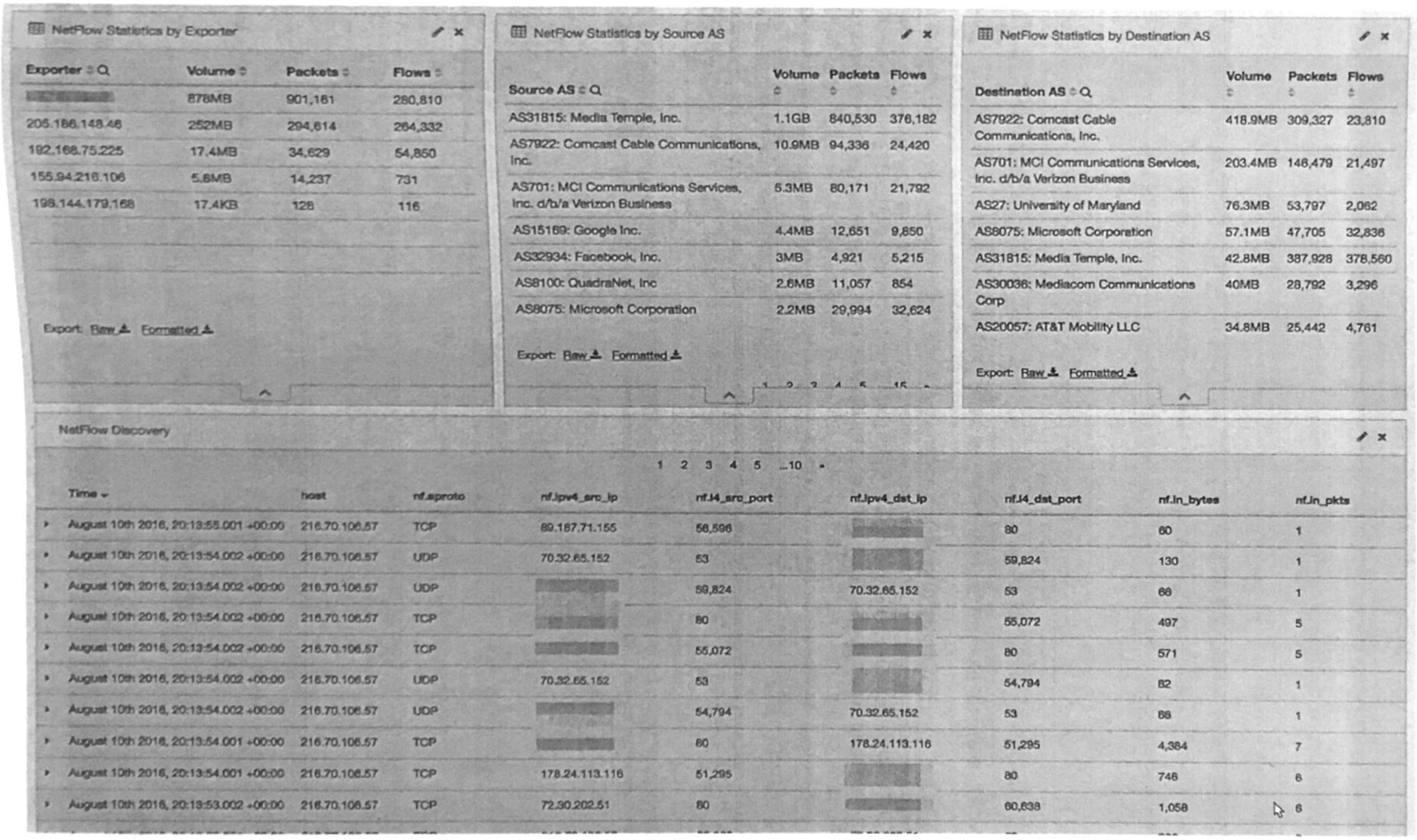




The next row in the SOF-ELK NetFlow dashboard includes several lists, including NetFlow classification by exporter IP address. This is useful because it allows the forensicator to quickly and easily isolate the traffic in view to just one vantage point within the broader evidence collection.

There are also panels for the BGP autonomous system, or AS, of the traffic source and destination. This is added during the Logstash ingest in the same manner as the GeoIP fields. In an investigative context, these values can help to characterize the traffic based on overall ownership for the IP address space. For example, the traffic in the screenshot involves a great deal of data sent from Media Temple, a hosting provider where one of the primary servers being monitored is hosted. However, if an unfamiliar, or known malicious, or otherwise suspicious AS were to appear in the top source or destination AS entries, this would certainly be worth digging into.

Finally, at the bottom of the dashboard is the familiar ELK document listing as we saw previously with the log dashboards.



NetFlow Statistics by Exporter

Exporter	Volume	Packets	Flows
[redacted]	878MB	901,161	280,810
205.186.148.46	252MB	294,614	264,332
192.168.75.225	17.4MB	34,629	54,850
155.94.216.106	5.8MB	14,237	731
198.144.179.168	17.4KB	128	116

Export: Raw Formatted

NetFlow Statistics by Source AS

Source AS	Volume	Packets	Flows
AS31815: Media Temple, Inc.	1.1GB	840,530	376,182
AS7922: Comcast Cable Communications, Inc.	10.9MB	94,336	24,420
AS701: MCI Communications Services, Inc. d/b/a Verizon Business	5.3MB	80,171	21,792
AS15169: Google Inc.	4.4MB	12,651	9,850
AS32934: Facebook, Inc.	3MB	4,921	5,215
AS8100: QuadraNet, Inc	2.6MB	11,057	854
AS8075: Microsoft Corporation	2.2MB	29,994	32,624

Export: Raw Formatted

1 2 3 4 5 ...15

NetFlow Statistics by Destination AS

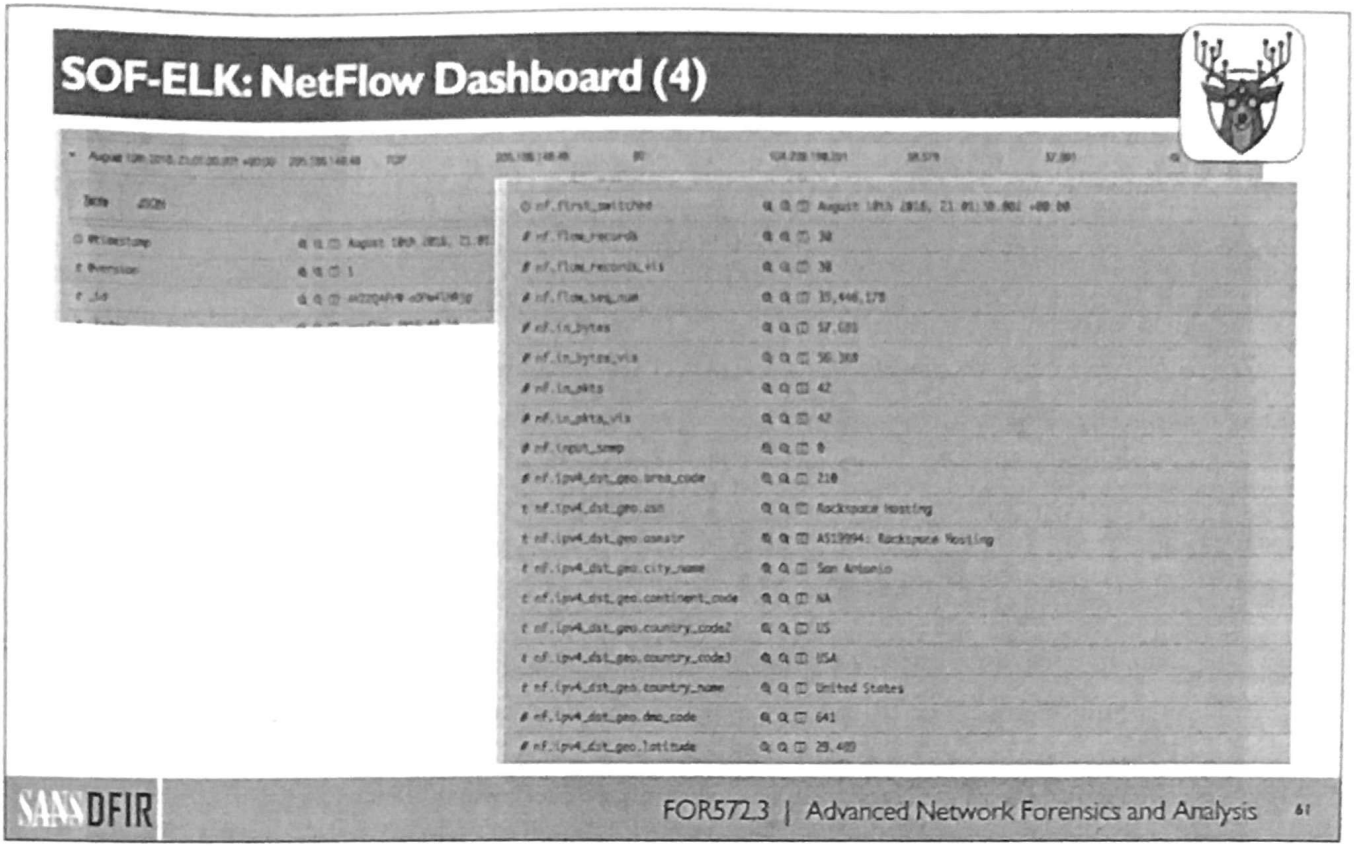
Destination AS	Volume	Packets	Flows
AS7922: Comcast Cable Communications, Inc.	418.9MB	309,327	23,810
AS701: MCI Communications Services, Inc. d/b/a Verizon Business	203.4MB	146,479	21,497
AS27: University of Maryland	76.3MB	53,797	2,062
AS8075: Microsoft Corporation	57.1MB	47,705	32,836
AS31815: Media Temple, Inc.	42.8MB	387,928	378,560
AS30036: Mediacom Communications Corp	40MB	28,792	3,296
AS20057: AT&T Mobility LLC	34.8MB	25,442	4,761

Export: Raw Formatted

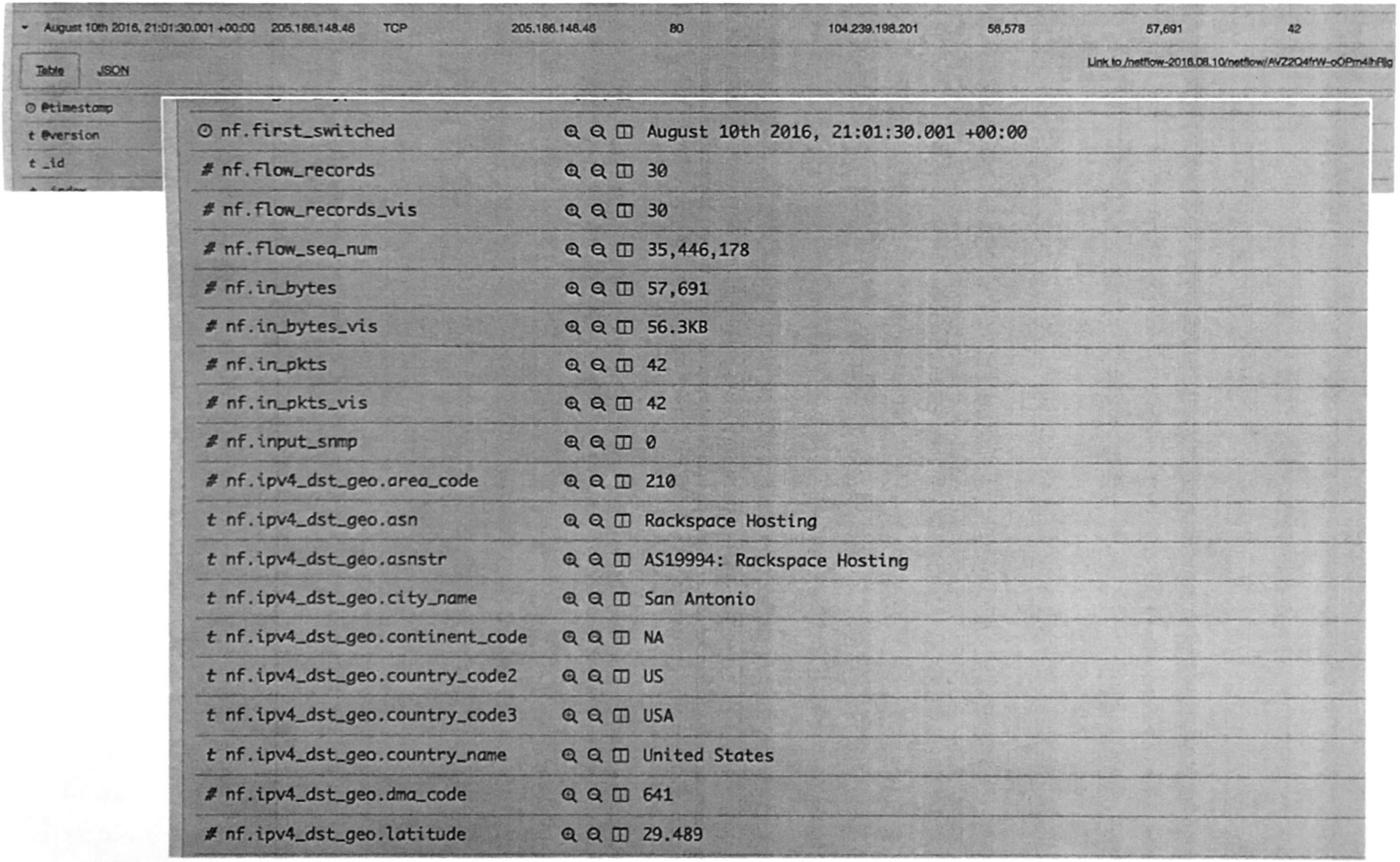
NetFlow Discovery

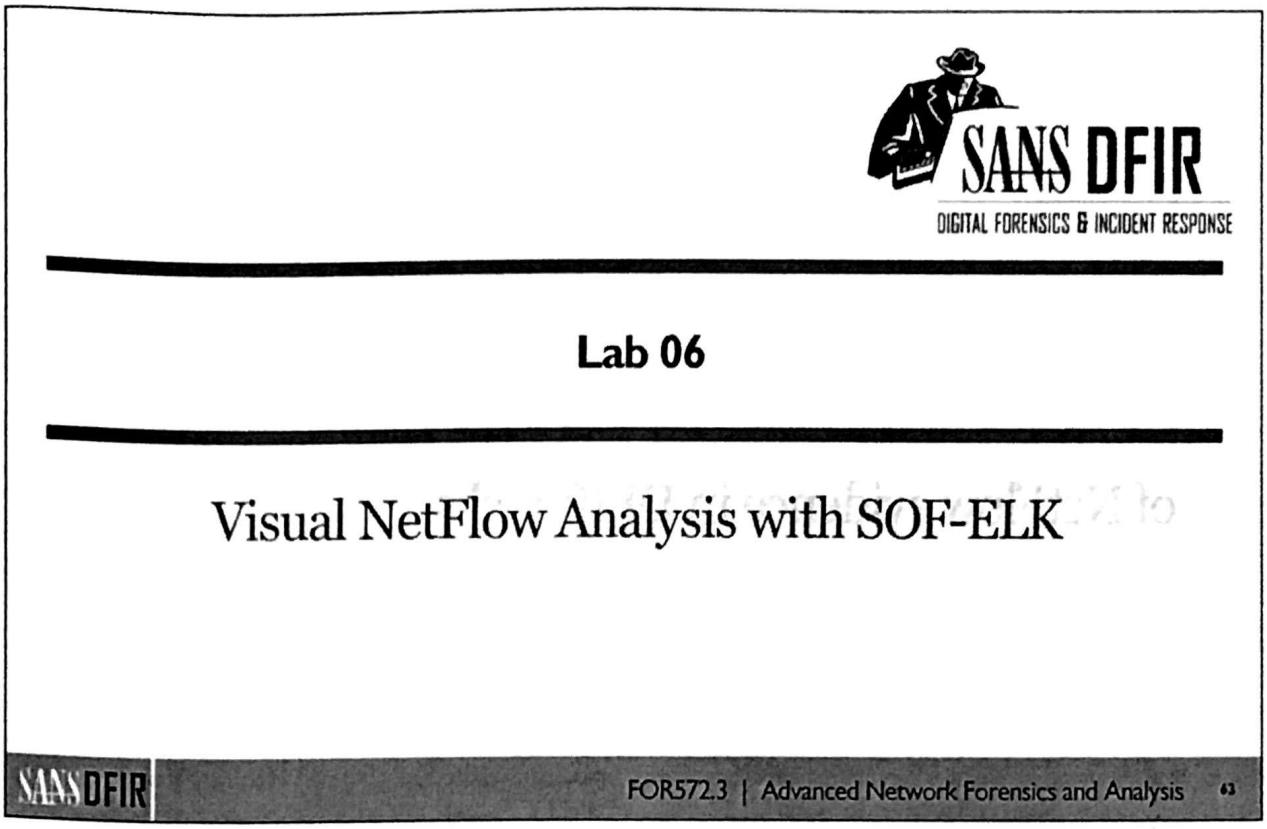
1 2 3 4 5 ...10

Time	host	nf.sproto	nf.ipv4_src_ip	nf.l4_src_port	nf.ipv4_dst_ip	nf.l4_dst_port	nf.in_bytes	nf.in_pkts
August 10th 2016, 20:13:55.001 +00:00	216.70.106.57	TCP	89.187.71.155	58,596	[redacted]	80	60	1
August 10th 2016, 20:13:54.002 +00:00	216.70.106.57	UDP	70.32.65.152	53	[redacted]	59,824	130	1
August 10th 2016, 20:13:54.002 +00:00	216.70.106.57	UDP	[redacted]	59,824	70.32.65.152	53	66	1
August 10th 2016, 20:13:54.002 +00:00	216.70.106.57	TCP	[redacted]	80	[redacted]	55,072	497	5
August 10th 2016, 20:13:54.002 +00:00	216.70.106.57	TCP	[redacted]	55,072	[redacted]	80	571	5
August 10th 2016, 20:13:54.002 +00:00	216.70.106.57	UDP	70.32.65.152	53	[redacted]	54,794	82	1
August 10th 2016, 20:13:54.002 +00:00	216.70.106.57	UDP	[redacted]	54,794	70.32.65.152	53	66	1
August 10th 2016, 20:13:54.001 +00:00	216.70.106.57	TCP	[redacted]	80	178.24.113.116	51,295	4,384	7
August 10th 2016, 20:13:54.001 +00:00	216.70.106.57	TCP	178.24.113.116	51,295	[redacted]	80	746	6
August 10th 2016, 20:13:53.002 +00:00	216.70.106.57	TCP	72.30.202.51	80	[redacted]	60,638	1,058	6



These records are displayed in a spreadsheet-like interface, but each row can be expanded to show all fields present in the underlying data store. The analyst can use these fields to build filters through the magnifying glass icons with the plus/minus signs, as well as add columns to the record listing with the table icon.





This page intentionally left blank.

Lab 06 Objectives: Visual NetFlow Analysis with SOF-ELK

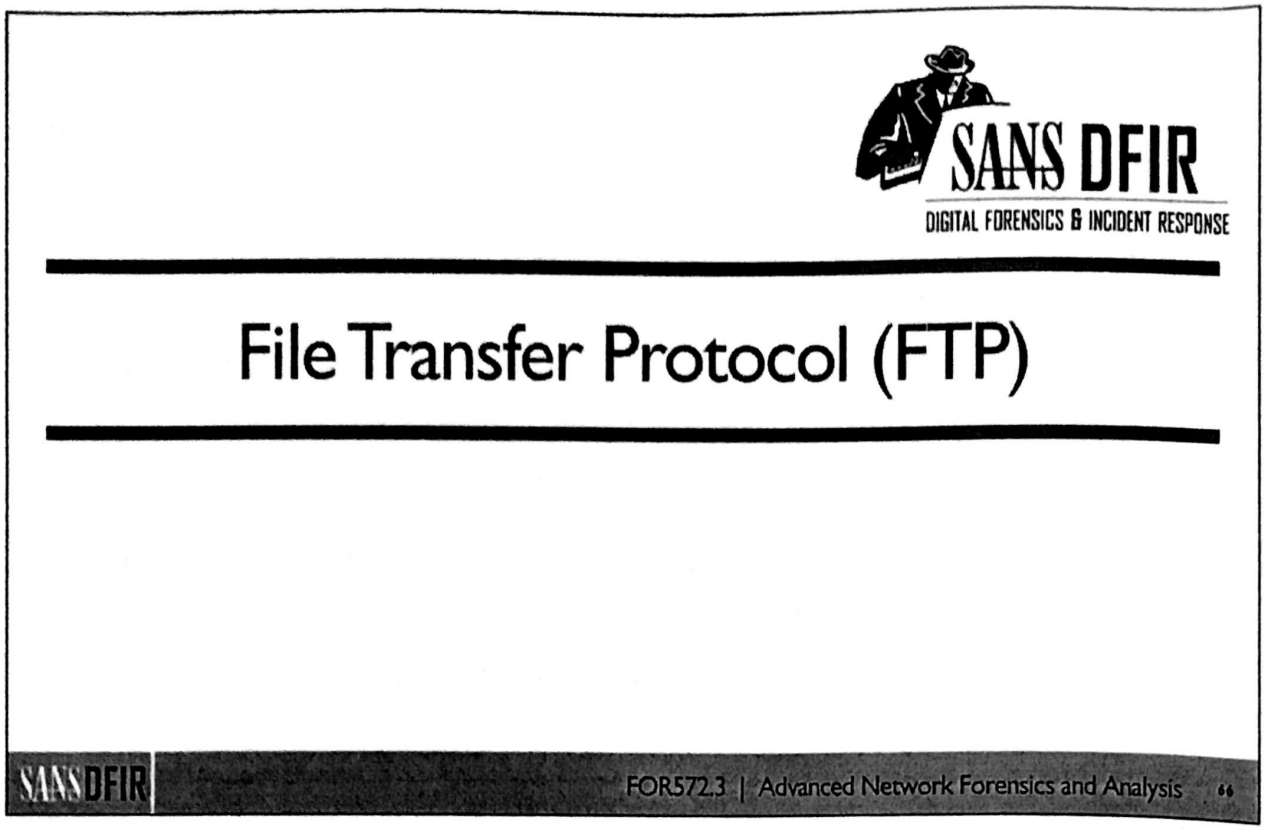
- Distill from `nfcapd` evidence, load to SOF-ELK
- Explore evidence and build leads via dashboard
- Identify key NetFlow findings using the SOF-ELK NetFlow workflow
- Understand the inherent and lead-generating value of NetFlow evidence in DFIR tasks

This page intentionally left blank.

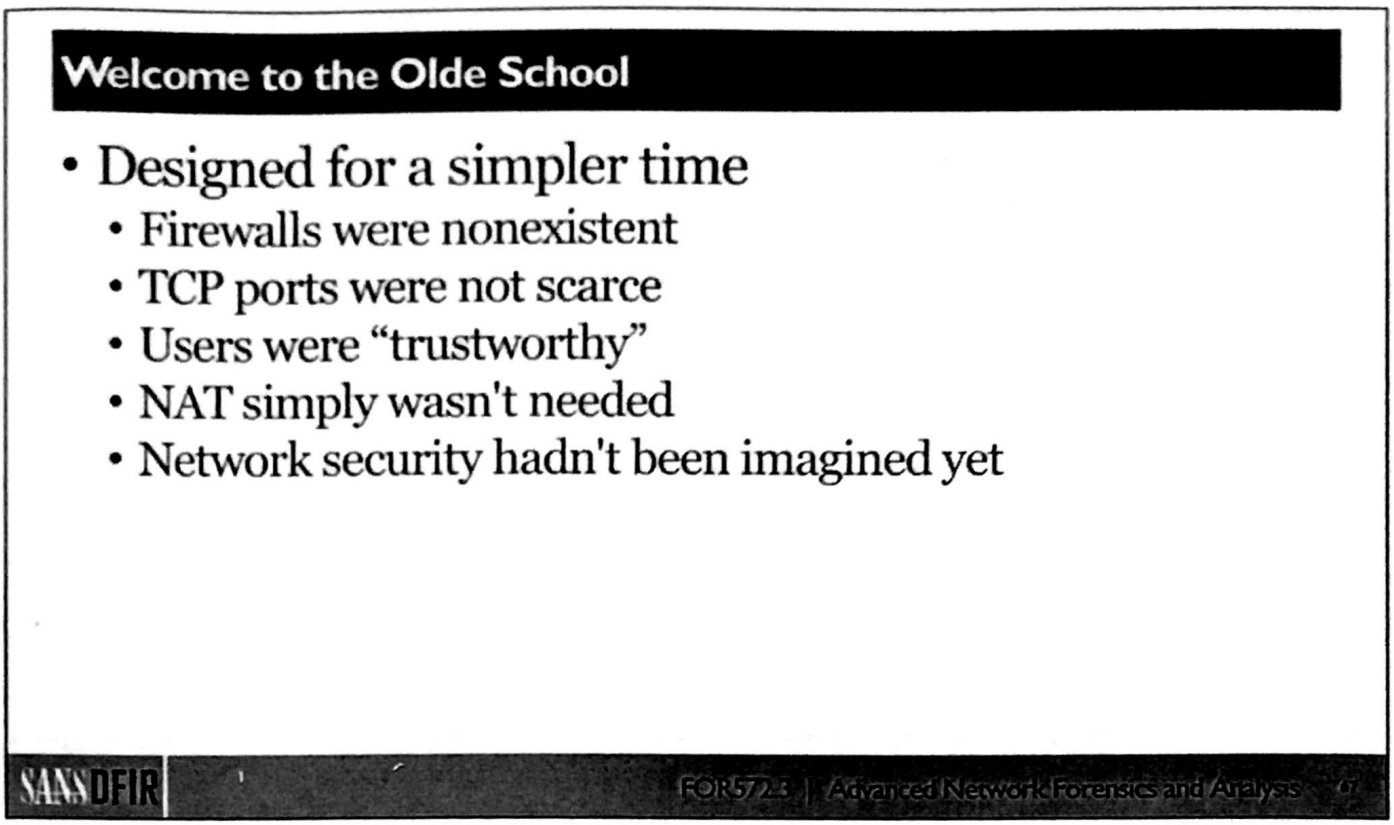
Lab 06 Takeaways: Visual NetFlow Analysis with SOF-ELK

- NetFlow can provide quick overview of traffic
  - Often requires additional confirmation due to lack of packet-level detail and underlying content
  - Typical port-protocol association may be used, but is not absolute until confirmed via layer 7 means
- SOF-ELK's NetFlow Dashboard can help identify traffic patterns that warrant further investigation
  - Visually finding patterns is easier for humans than examining thousands of lines of text output

This page intentionally left blank.



This page intentionally left blank.



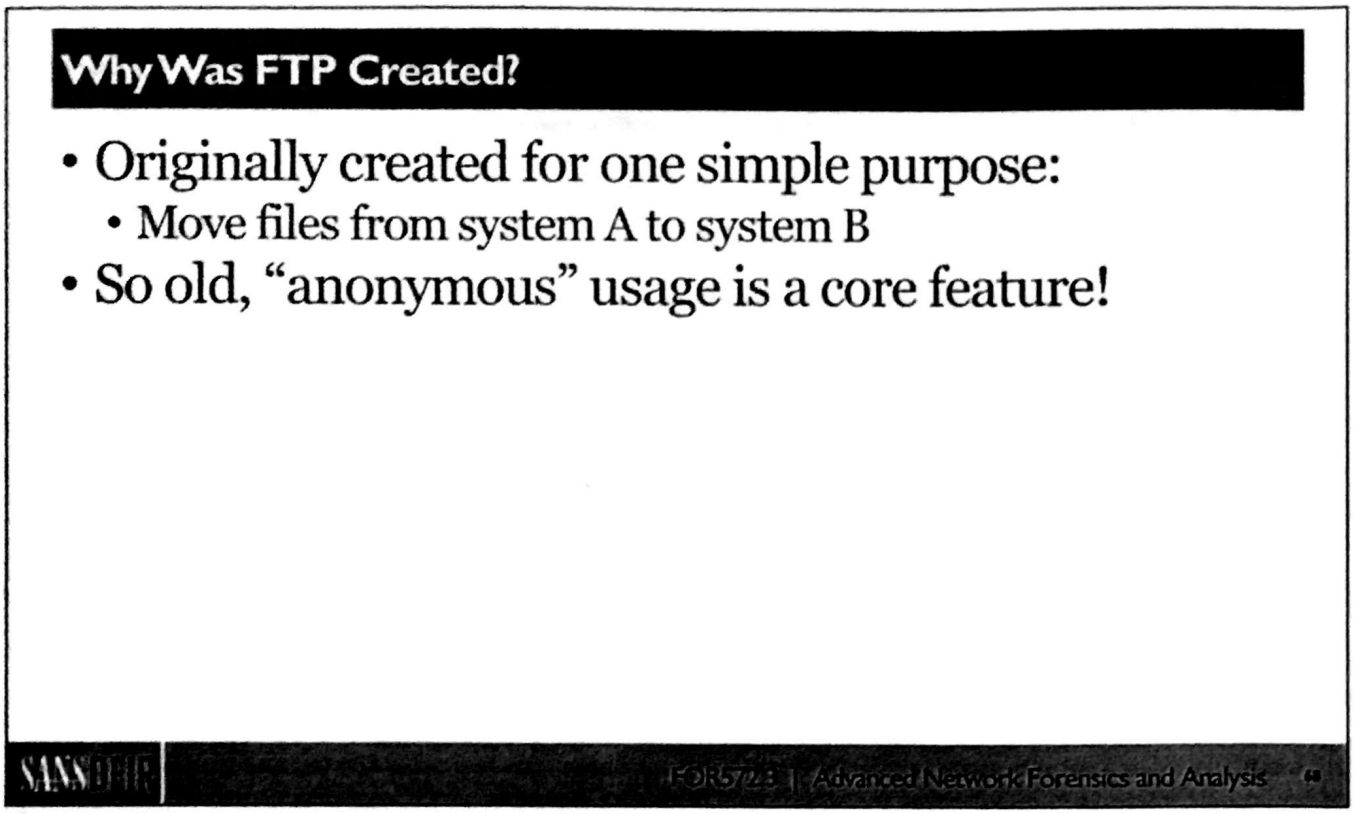
FTP is an old and venerable protocol, but before we dive into some of its unique functionality and features, we need to jump into the Delorean and travel back to a time when the Internet barely resembled what it does today. It all started back in 1971. While at MIT, Abhay Bhushan wrote RFC114, which showed the method he developed to transfer files between a GE 645 and a PDP-10.

For starters, the concept of selectively blocking traffic like a firewall was simply unheard of! Routers were in place to *deliver* packets, not *prevent* them! In that vein, TCP port allocation was still pretty wide open—you didn't have to worry about stepping on anyone's toes when using a few extra ports—as long as they were above 1023, of course. In part, that's because the general unwashed masses didn't know—let alone care—what the Internet was. Anyone online inherently held a level of trust from the other users in such a comparatively small community. For the most part, everyone in the land of the big Internet was assumed to be a benevolent actor, behaving themselves appropriately.

Another alien concept to the founders of the Internet would be Network Address Translation, or NAT. As you should know, NAT allows many endpoints with their own IP addresses to "live" behind another device that masks the endpoints' IP addresses with its own. Almost any home with a broadband connection has this today in the form of the gateway router/wireless access point, or similar device. But at the time the Internet was designed, the idea that there would be more than FOUR BILLION allocated IP addresses was certainly preposterous. It was a very safe assumption that every connected device would have direct access to any other connected device.

That all underscores the simple truth that the Internet was created without the slightest goal of security or scarcity—it was designed to allow data to flow even after a nuclear catastrophe. But let's get back to FTP…

References:
`http://for572.com/5w-b2`



In its more common form, FTP was designed to do one thing and do that one thing well: Transfer Files between endpoints on the network. (Get it? File Transfer Protocol?)

That's it.

Although user authentication was an FTP feature from the start, the fundamental goal of transferring files ran so deep that "anonymous" FTP was widely deployed until HTTP became the preferred authentication-free file distribution protocol.

References:
`http://for572.com/zgqx-`

FTP Today

- Grown into a complex behemoth
  - Bolt-on extensions for modern requirements
  - New commands with varying levels of support
- Decreasing in popularity, but still lives
  - Zombie protocols will never die
- NOT to be confused with SFTP
  - This is an extension to the SSH protocol
  - Similarities to true FTP are cosmetic

As with any simple, elegant solution, FTP has been Franken-protocoled into a complex beast of a system that still lives on, despite a number of viable alternatives coming into existence. FTP has a number of extensions that provide features such as encryption, functionality across firewalls and NAT devices, queries for additional pieces of file metadata, and more. As with any "evolved" protocol, client support for these features can vary widely.

Because FTP use itself is declining, we should not expect a great deal of increased client support for these extensions, and should probably look to newer, more robust solutions to replace that one simple function. But, as investigators, our job is to analyze the traffic that exists, not the traffic that should exist. It's important to realize that FTP is still in use today—for both good and bad purposes—which is why we spend the time here to make sure you're familiar with the more common incarnations of FTP traffic.

Speaking of newer, more robust solutions, you may have heard of SFTP, or Secure File Transfer Protocol. This is not FTP! In fact, it's not even close. SFTP is the term for file transfer over the SSH (secure shell) protocol. Though they share the common goal of transferring files between hosts, they are distant cousins at best. Any similarities between them are most likely the result of the SFTP developers creating a protocol that FTP users could easily adapt to.

FTP Basics

- Multiple-stream protocol
  - Command channel (TCP/21)
  - Data channels (TCP>1023)
- "It's complicated"
  - Command channel contents drive TCP behavior of data channel transfers
- Plaintext
  - New extensions add encryption

Let's talk about the basics—what does FTP look like?

FTP is somewhat unique in that it uses multiple TCP streams—one accommodates commands issued from the client to the server, as well as numerical reply codes in the other direction. We will call this the "command channel". Typically, the server listens for this traffic on TCP port 21. The "data" channel is where things get interesting. This connection is unique for three main reasons.

Here, "data" means results from commands—not just the files that someone uses FTP to transfer in the first place. So when you ask for a directory listing over the command channel, the resulting status code will come back on the command channel but the file listing comes back on the data channel. A new data channel is opened for each reply. Each and every command response is a distinct stream.

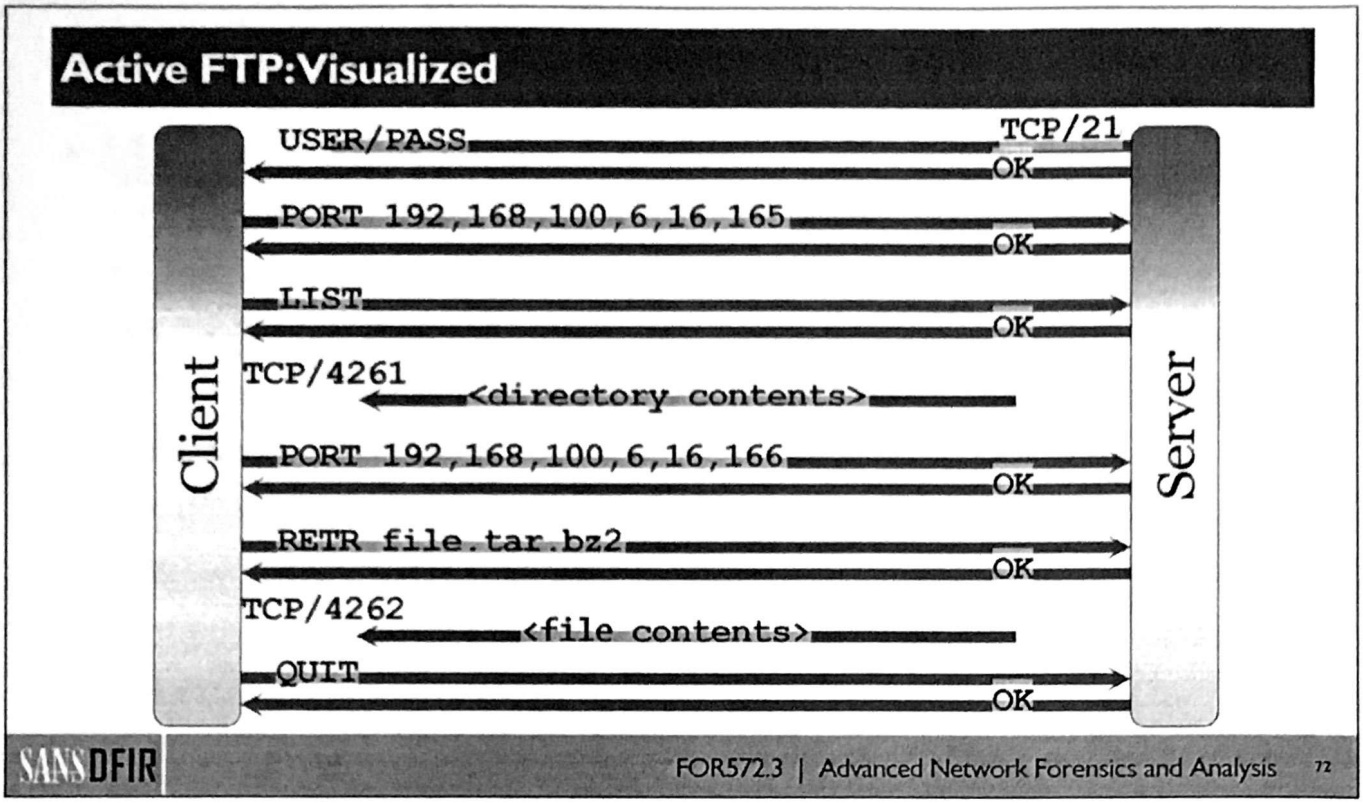
Originally, for each data channel connection, the "server" actually operates as the "client". Yes, that is strangely correct! When it was designed, a download command issued by a human operator on the client machine would also open a listening socket on that "client" system, to which the server would then connect, and blindly deposit the results. (It should be noted that there is a more traditional client-server mode for the data channel as well, but it was not an original feature of the protocol. We will walk through both.)

(We will discuss this in more detail—and with pictures—in a few slides.)

To accomplish this somewhat convoluted protocol, the command channel passes a series of messages containing instructions on how to set up each data channel connection. In an effort to confound SANS students, early FTP developers made these messages into a bit of a math problem. However, these messages tell the server and client what port numbers to use for each data connection.

Finally, FTP is a plaintext protocol. As we mentioned earlier, security was simply not a concern when FTP was developed, so the idea of evil on the wires was still over a decade or two away. Although some extensions provide encryption capabilities, they are relatively recent additions.

(What could possibly be wrong with a plaintext announcement about which TCP port you're about to open with a socket where the system will happily receive any data sent?!)



This slide depicts a typical FTP session, during which the client requested a directory listing and then downloaded a file named "file.tar.bz2".

You can see that the client first connects via the command channel on TCP port 21, then issues a PORT command before requesting a directory list. Upon issuing the PORT command, the **client** binds a listener to TCP port 4261. Then, the **server** connects to this listening port and delivers the directory listing over the data channel.

The client then issues another PORT command before requesting the "file.tar.bz2" file. The client uses TCP port 4262 this time. (These port numbers are often—but not always—sequential.) However, instead of the directory listing, the server delivers the contents of the requested file via the data channel.

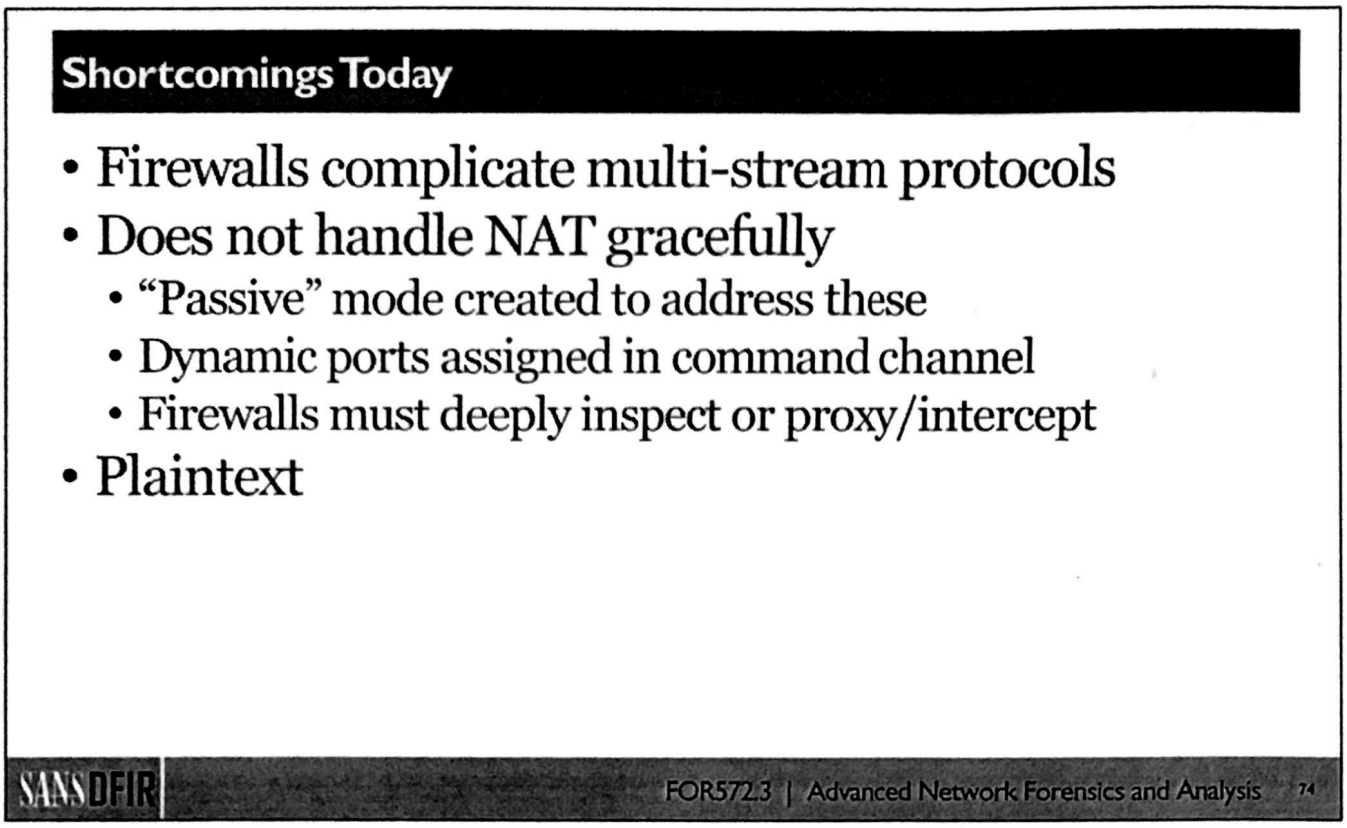
In this example, the client issues the QUIT command, which cleanly closes the command channel.

Active FTP: PORT Command

- `PORT 192,168,100,6,16,165`
- Client opens socket for data receipt
  - IP address 192.168.100.6
  - TCP port 4261
    - (16 * 256) + 165 = 4261

The PORT command requires some additional explanation. As an ASCII protocol, FTP relies on the context of byte-wide octets for the IP address and port, but renders them in their decimal equivalents.

The first four sections are the octets of the IP address—very straightforward. However, the fifth and sixth values represent a 16-bit number, where the first number represents the highest-order eight bits and the second number represents the lowest-order 8 bits. Therefore, the first number is multiplied by 256 and added to the second number. The resulting number must be greater than 1,023. This is because a non-privileged user may be using the FTP client software but cannot typically bind to (aka "listen on") a port between 1 and 1,023.



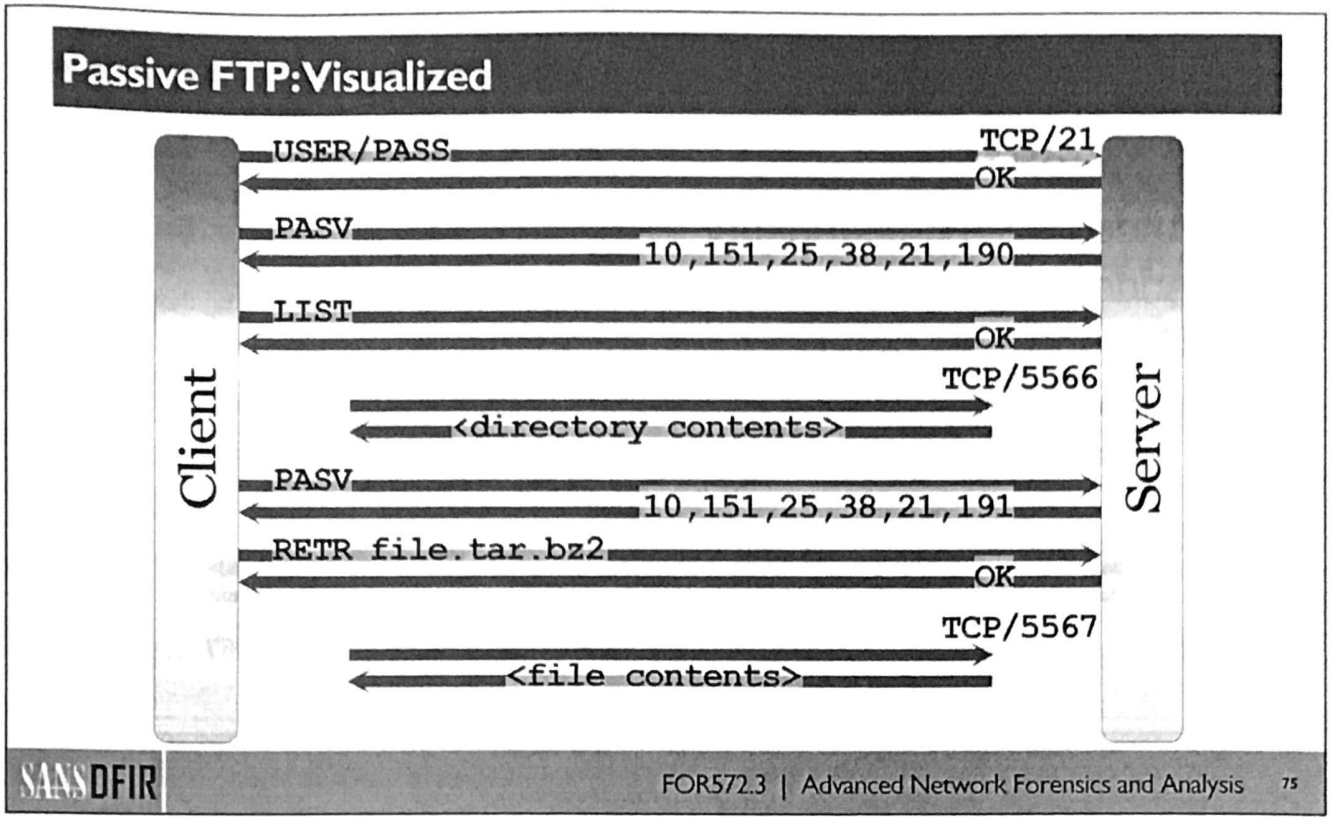
In the typical modern-day network environment, the original incarnation of FTP simply would not be functional. In some ways, even the modern implementations of FTP are not advisable. There are three main reasons for this, two of which involve the multi-stream nature of the protocol.

Firewalls are designed to limit network flows between two endpoints. NAT is a technology that enables a true endpoint's IP address to be altered without either end being aware of the change. FTP was designed with the expectation that traffic between endpoints can pass unimpeded, and the protocol's internals involve endpoints announcing their IP addresses. You should be able to see how these technologies would be inherently at odds with the FTP protocol.

Though "passive" mode was specified in the RFCs as early as 1980 (RFC 765), its use expanded dramatically as firewalls and NAT became more prevalent in operational networks. We will more extensively discuss the differences between the original default "active" mode and "passive" mode in the next slides. The key point is that while active mode requires the client to open a listening socket to receive data or command results from the server, passive mode requires the server to open the socket so the client can connect and receive the data.

However, even in passive mode, the mere fact that many, many TCP sessions are required for what the user perceives as a single FTP "session" complicates things for the firewall. At one time, guidance for firewall administrators was to allow internal clients access to external servers on any TCP port over 1023, or within a range known to be used for FTP transactions! Although it's easy to laugh at such broad and troublesome guidance today, the firewall technology was not sufficient to perform real-time rule modification. To do this, the firewall must deeply inspect command channel packets for the "PORT" command, which designated where a particular client could retrieve the data it just requested. Such features are easy to accomplish today, but it's easy to see how complex a process it is to effectively manage FTP traffic across a firewall.

Finally, FTP is generally plaintext. Although there are extensions that enable encrypted connections, the overwhelming number of FTP servers operate without them. The entire connection—authentication, commands and results, and file transfers—are fully and completely accessible with your friendly neighborhood packet capture solution.



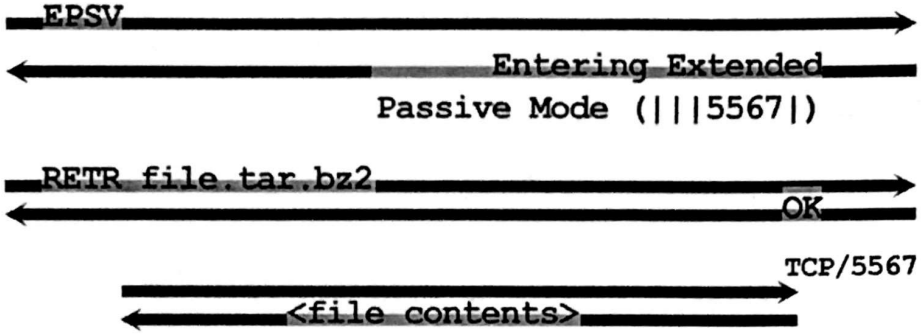
Passive mode FTP sessions generally use a similar command set and sequence of events. However, the client requests passive mode with the PASV command. The server then provides a reply similar to the PORT command we previously saw. The makeup and math behind these numerical values are the same.

The server then opens a listening socket according to the IP and port specification, and the client connects to that listening port to collect results.

This mode is much more resilient to problems with firewalls and NAT because the client system never opens up a listening port. The "server" is always taking on the role of a "server".

Extended Passive FTP

- RFC 2428 (Sept 1998) introduced EPSV
  - Added IPV6 functionality
  - Added more robust NAT handling
  - Uses raw port numbers, not calculations



A slight modification to the concept of passive mode FTP, known as Extended Passive Mode, came along in the late 1990s. This mode was primarily designed to handle IPv6 functionality, but even in IPv4 networks, it is perhaps the mode most widely encountered today.

The typical exchange is nearly identical to the passive mode example on the previous slide, but there is no octet math. Instead, the server simply indicates the port number on which it will be providing results.

At the same time Extended Passive Mode was defined, an updated version of the "PORT" command was established. The "Extended Port" command, or "EPRT", behaves the same as the PORT command described earlier, except that it can handle IPv6 addresses.

References:
`http://for572.com/w9f5y`

Capturing FTP

~~• $ sudo tcpdump -i eth0 ⏎~~
~~'(tcp and port 21)'~~

```
• $ sudo tcpdump -i eth0 '(tcp and (port 21 or ⏎
   (src portrange 1024-65535 and ⏎
    dst portrange 1024-65535)))'
```

- Need to acquire command and data streams
  - Command channel contains only issued commands—no results or file transfers
  - Data channel contains only command results and file transfers—no filenames or other artifacts
- Just capturing TCP/21 leaves huge voids

It is quite straightforward to capture active mode FTP traffic. As long as you remember to acquire both the command and data channels, the session itself can be reconstructed as needed. Remember that while the command channel contains issued commands and their corresponding result codes, the directory listings and file transfers sent from the server traverse a separate data channel. Of course, we're likely interested in what files were transferred, yet a capture of just the command channel would not acquire those files.

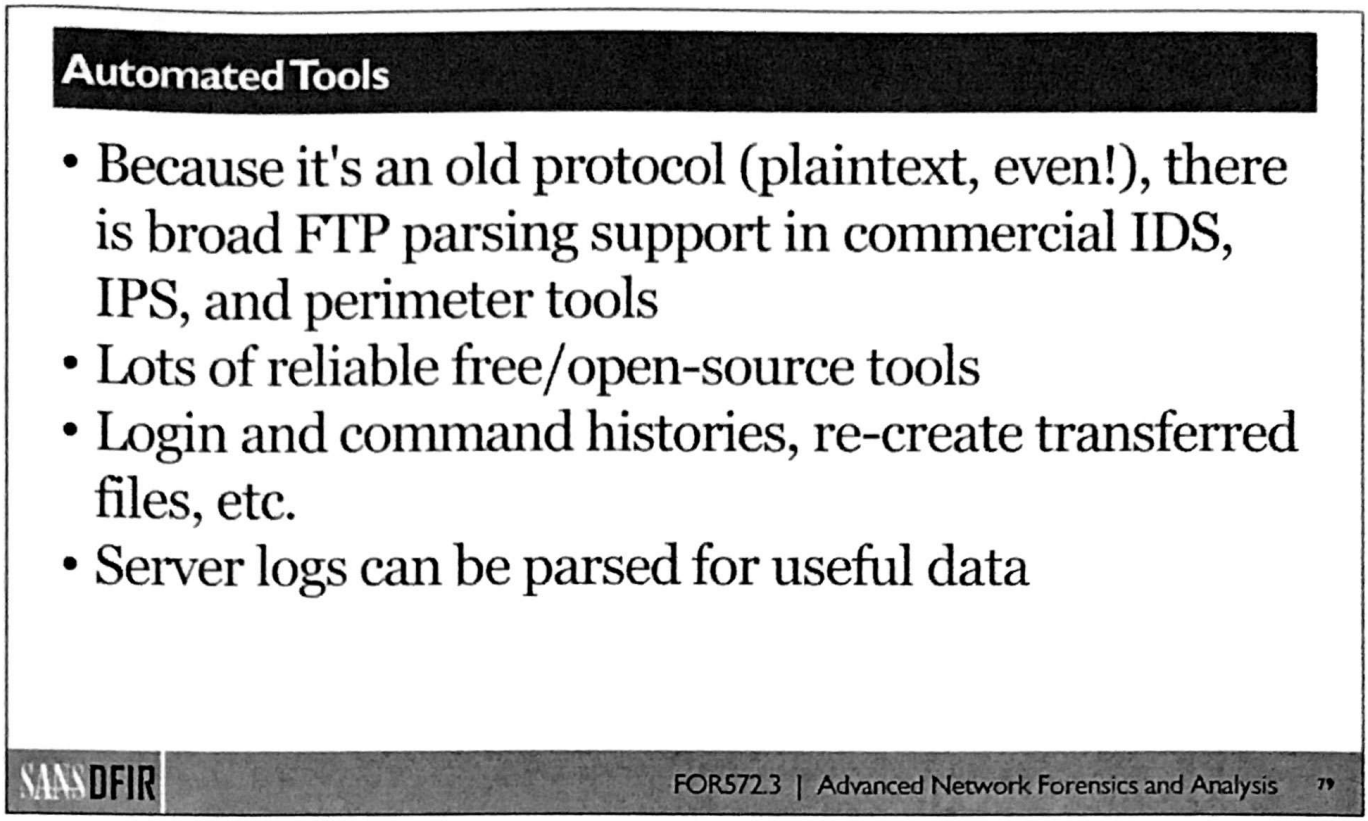
However, the tcpdump command here would not be useful for passive mode. In passive mode transfers, the data channel uses dynamically assigned TCP ports for the data connection. (Remember that typically, only a privileged user can bind ports <=1023.) With passive connections, the FTP server software may designate a range of ports for passive mode data channels, but there is no RFC stipulation for these port assignments. Technically, capturing active and passive FTP traffic with tcpdump would require the following command line:

```
$ sudo tcpdump -i eth0 '(tcp and (port 21 or ⏎
     (src portrange 1024-65535 and ⏎
      dst portrange 1024-65535)))'
```

Analyzing FTP

- Review command channel first
- Extract and convert relevant PORT commands
- Review data channel
  - Write to disk, etc.

To analyze captured FTP traffic, we must first review the command channel—a logical place to start. This characterizes the client's behavior and activities during the FTP session. Within this traffic, we also find the PORT commands, which define the ports used by the data channels. With this information, we can isolate a specific data channel—most likely a file transfer—and save that data to disk. Remember that the data channel contains no headers, footers, or other contaminating data. The data channel passes just the files that we're interested in. It's carving-free file reconstruction!

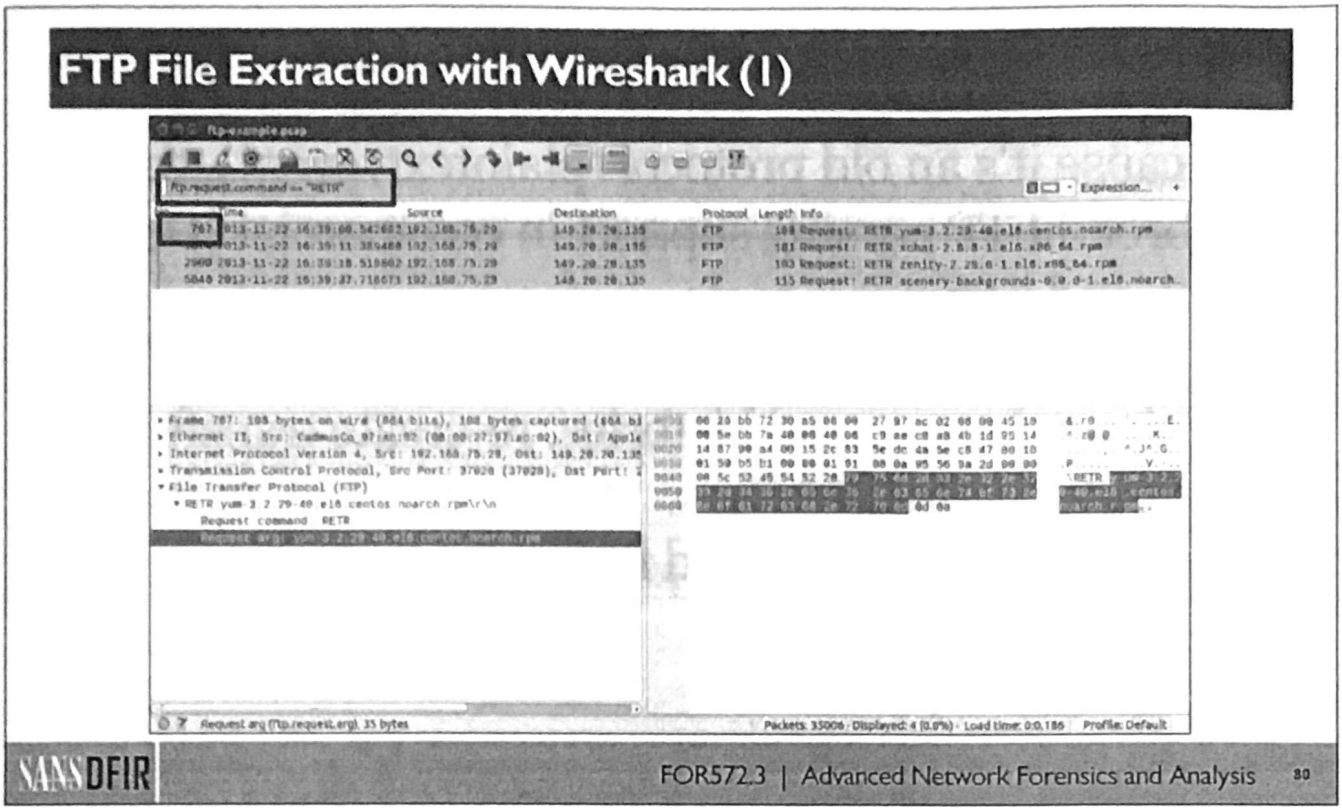


When it comes to FTP, there is a wide variety of tools to help us analyze, extract, and otherwise manage what can be a large volume of data. Owing to this are the facts that the protocol has strong roots back in the early days of ARPAnet and friends, as well as the fact that it's plaintext. For these and other reasons, many free and commercial tools include a robust FTP analysis feature set.

Some tools can extract all files that were transferred within an FTP session. For example, the free and paid versions of NetworkMiner[1] do this particularly well. Others can pull usernames and passwords, or command histories, from a pcap file or live network capture with little or no user intervention. An example of extracting a file transferred via FTP using just Wireshark is on the next slides.

Another useful mechanism to analyze FTP activity is the log files from the FTP server itself. We won't discuss this method in detail here, but it certainly bears mentioning that most FTP servers keep detailed usage accounting in what are typically plaintext files. These files can be analyzed with shell script tools or automated solutions as well. As with most system log files, they are frequently subject to long retention policies and can be a major boost when full packet captures were never created or have lapsed out of rotation.

References:
[1] `http://for572.com/kcv0n`



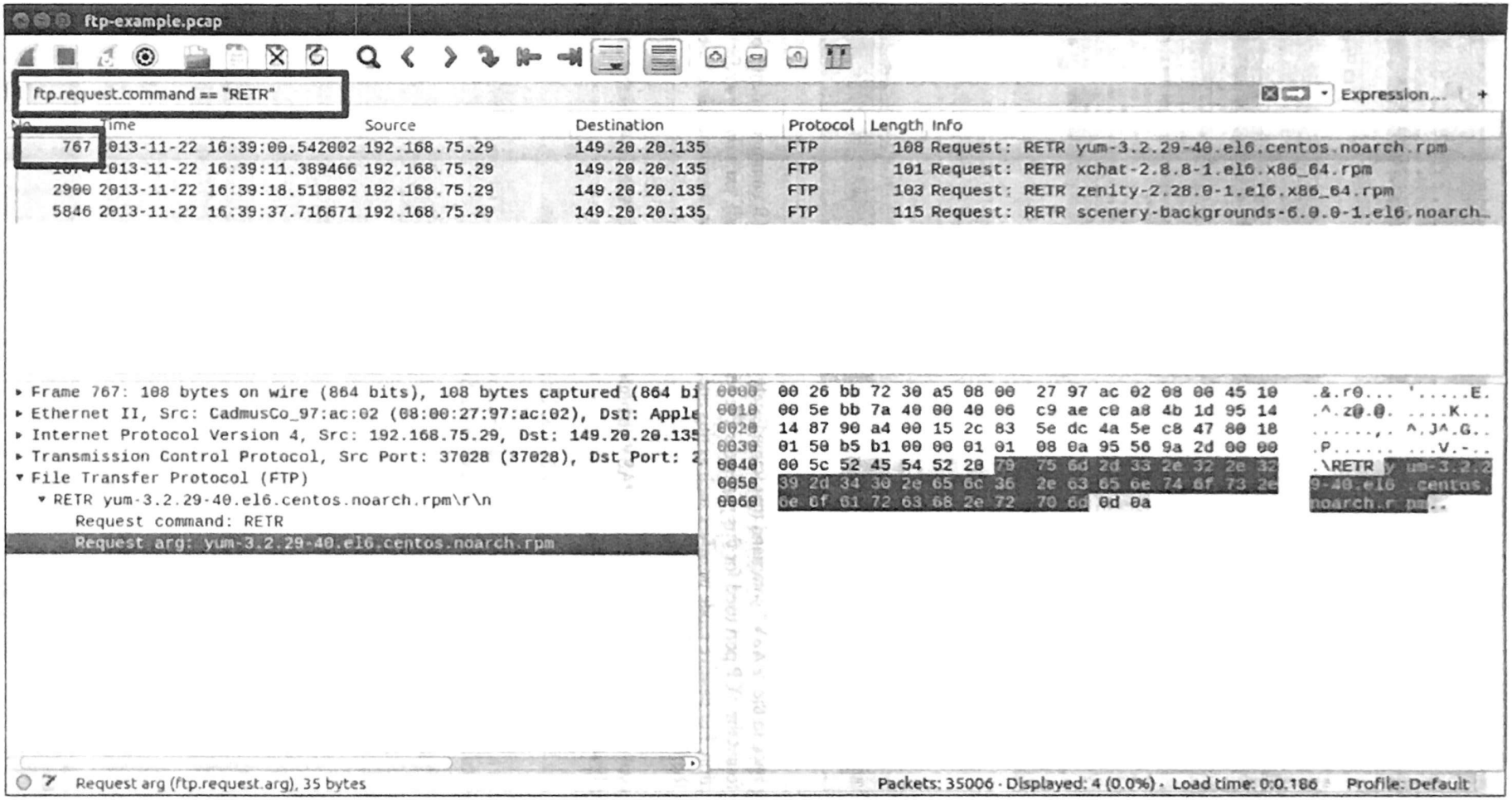
As an example, the next few slides show how to isolate a file transferred via FTP all within Wireshark. Similar steps could be taken to accomplish the same steps in **tshark** or from a custom Python script. Once you are familiar with the protocol itself, the means of pulling relevant data are plentiful.

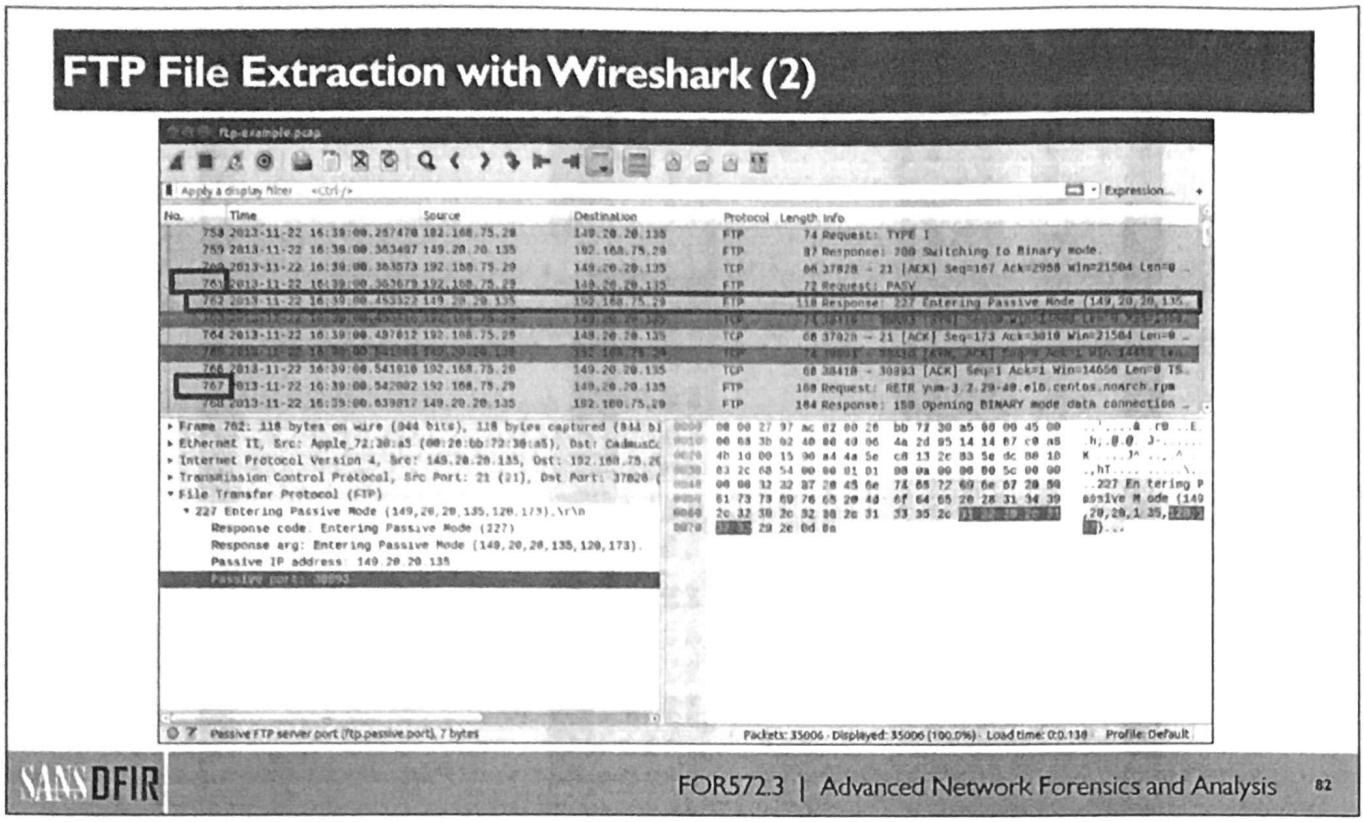
The "`ftp-example.pcap`" file used in this example is also available in the `/sample_pcaps/` directory of your USB drive. Feel free to follow along in class or on your own to experience how the extraction process works.

In this screen, we have used the following display filter to find all "RETR" FTP commands:

```
ftp.request.command == "RETR"
```

We have selected frame number 767, in which the client requested the file "`yum-3.2.29-40.el6.centos.noarch.rpm`".



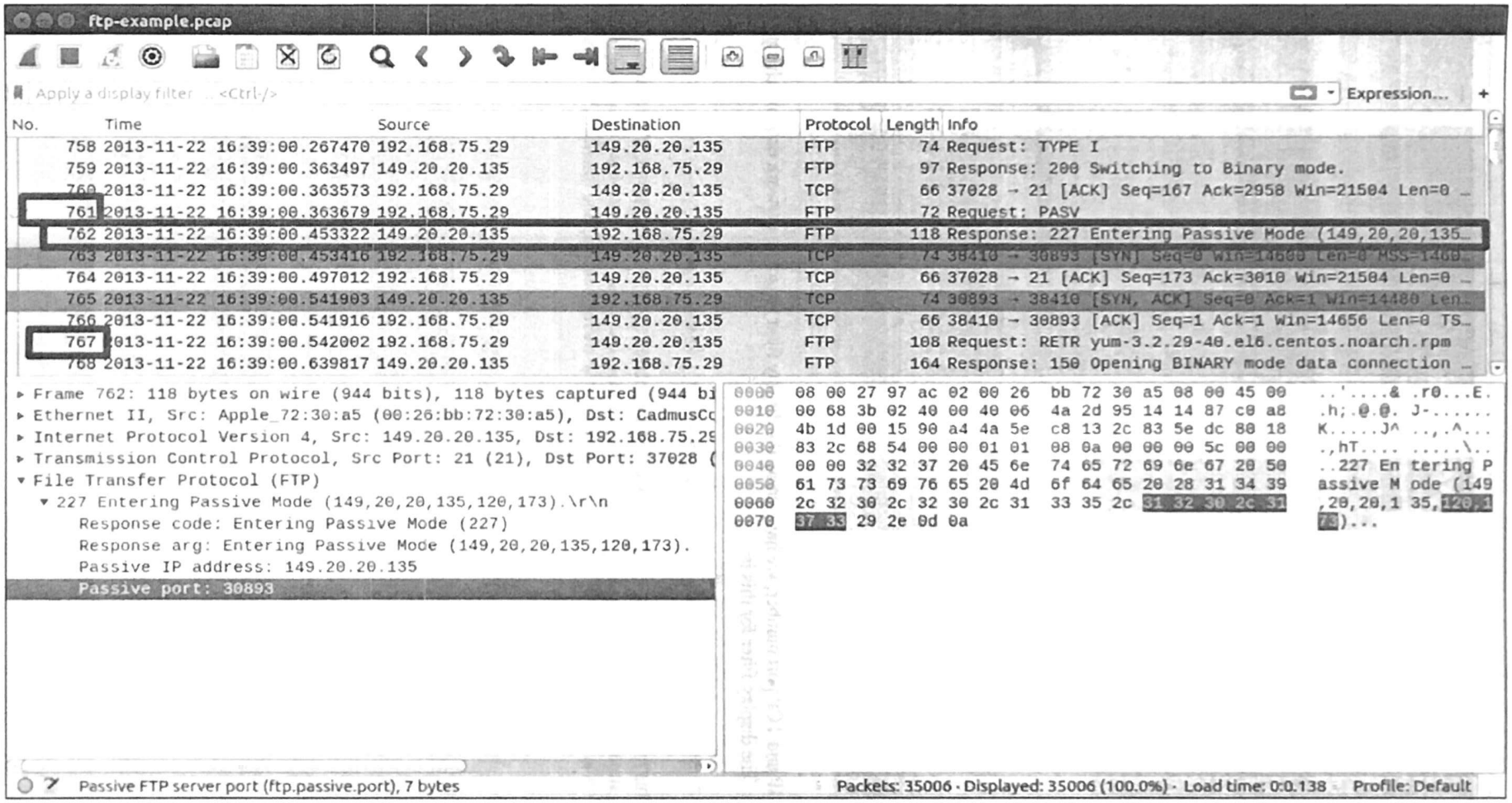


By stepping back to the "PASV" command that immediately preceded the RETR command identified previously, we quickly found the TCP port used for this particular file transfer. As you recall, each data transfer requires a new TCP connection. For passive mode transfers such as this one, the server informs the client side of the port within its response to the PASV command.

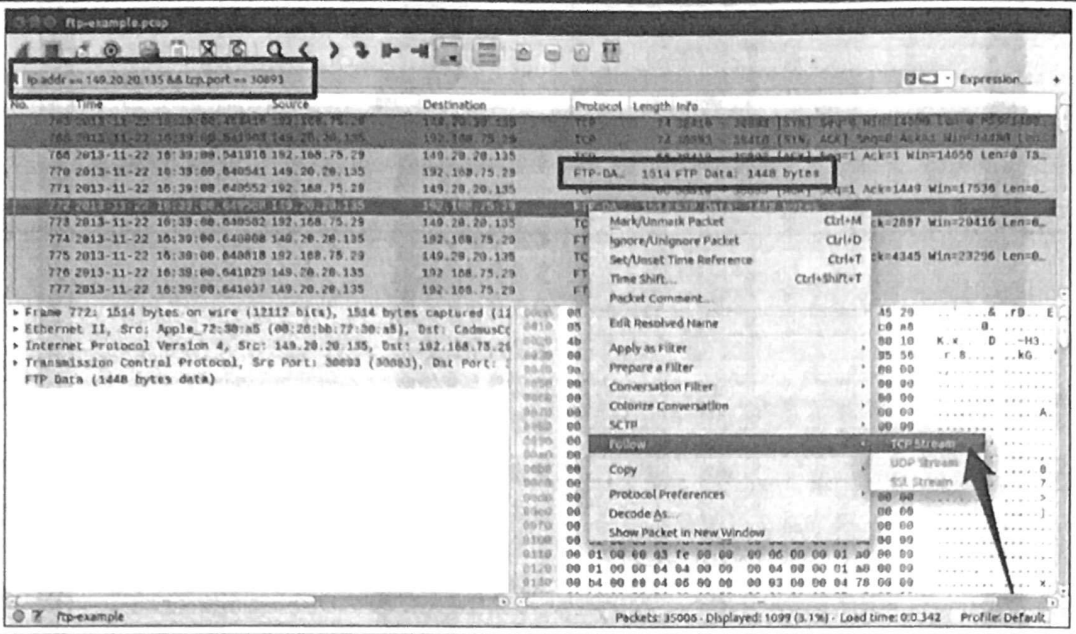
In the case above, you can see that the client's PASV command is in frame 761, with the server's reply in frame 762. The reply includes the following text:

```
227 Entering Passive Mode (149,20,20,135,120,173)\r\n
```

This indicates that the server will have data ready on IP address 149.20.20.135, TCP port 30893. Wireshark has conveniently calculated this for us as well, indicated in the lowermost box.



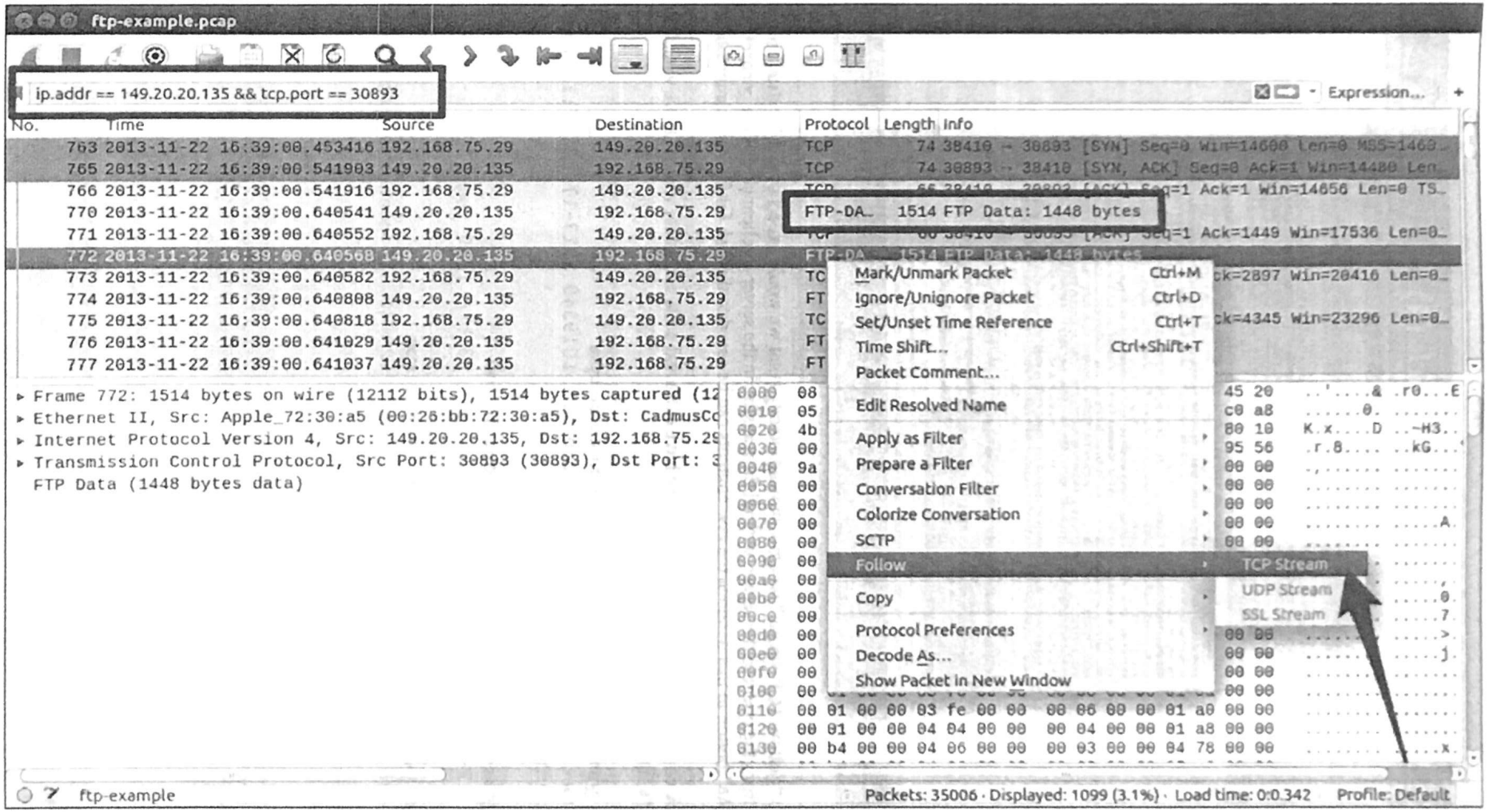
FTP File Extraction with Wireshark (3)



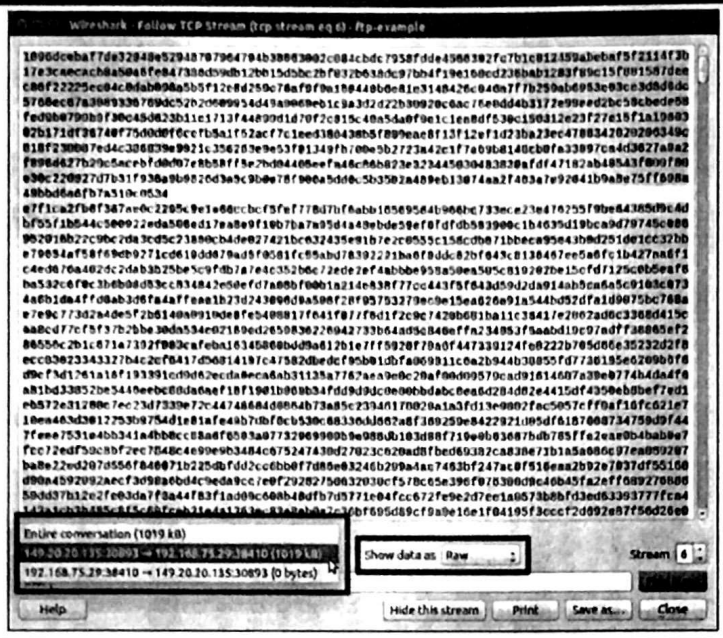
Using this IP and TCP port number, we have created a display filter to isolate network data matching these parameters. The display filter for this is:

```
ip.addr == 149.20.20.135 and tcp.port == 30893
```

Note that this may not uniquely isolate the transfer we are interested in, so at this point, it's important to identify how many streams the display filter has identified. In this case, there was only one stream, so by right-clicking one of its packets and selecting "Follow TCP Stream…", we know we're observing the intended file transfer.



FTP File Extraction with Wireshark (4)



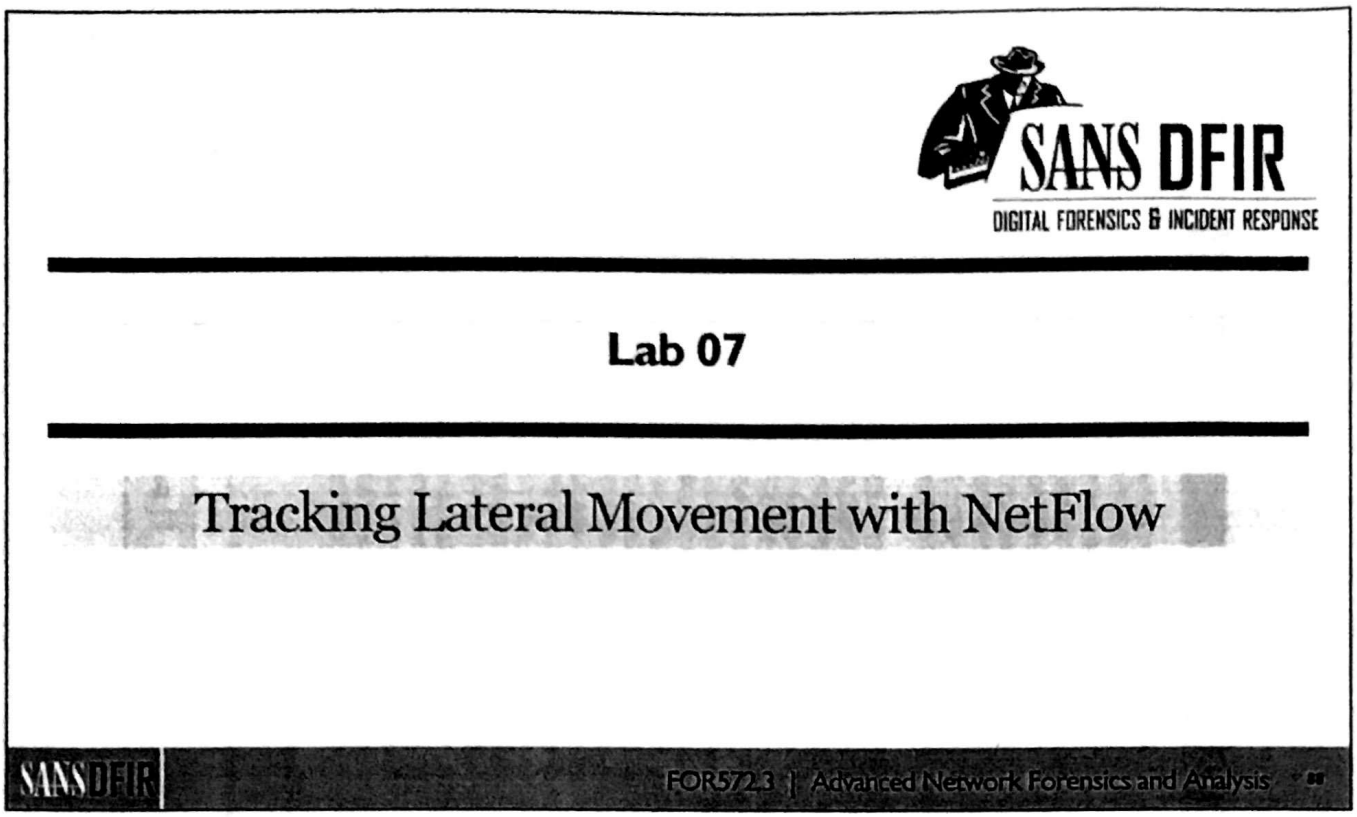
Finally, the "Follow TCP Stream" window showed exactly what we would expect of an FTP data transfer—zero data bytes from the client to the server, with all the transfer on the server-to-client side of the connection. By using the "Save As" button in this window, the raw contents can be saved to the disk for further examination.

A few quick commands in the shell show that we have extracted a file consistent with what we would expect from the transfer's filename.

```
$ ls -l yum-3.2.29-40.el6.centos.noarch.rpm
-rw-rw-rw- 1 sansforensics sansforensics 1019540 2013-11-22 17:11 yum-3.2.29-
40.el6.centos.noarch.rpm

$ file yum-3.2.29-40.el6.centos.noarch.rpm
yum-3.2.29-40.el6.centos.noarch.rpm: RPM v3 bin i386 yum-3.2.29-40.el6.centos

$ md5sum yum-3.2.29-40.el6.centos.noarch.rpm
70710806ef778f84a709b1a2318fd14b  yum-3.2.29-40.el6.centos.noarch.rpm
```



This page intentionally left blank.

Lab 07 Objectives: Tracking Lateral Movement with NetFlow

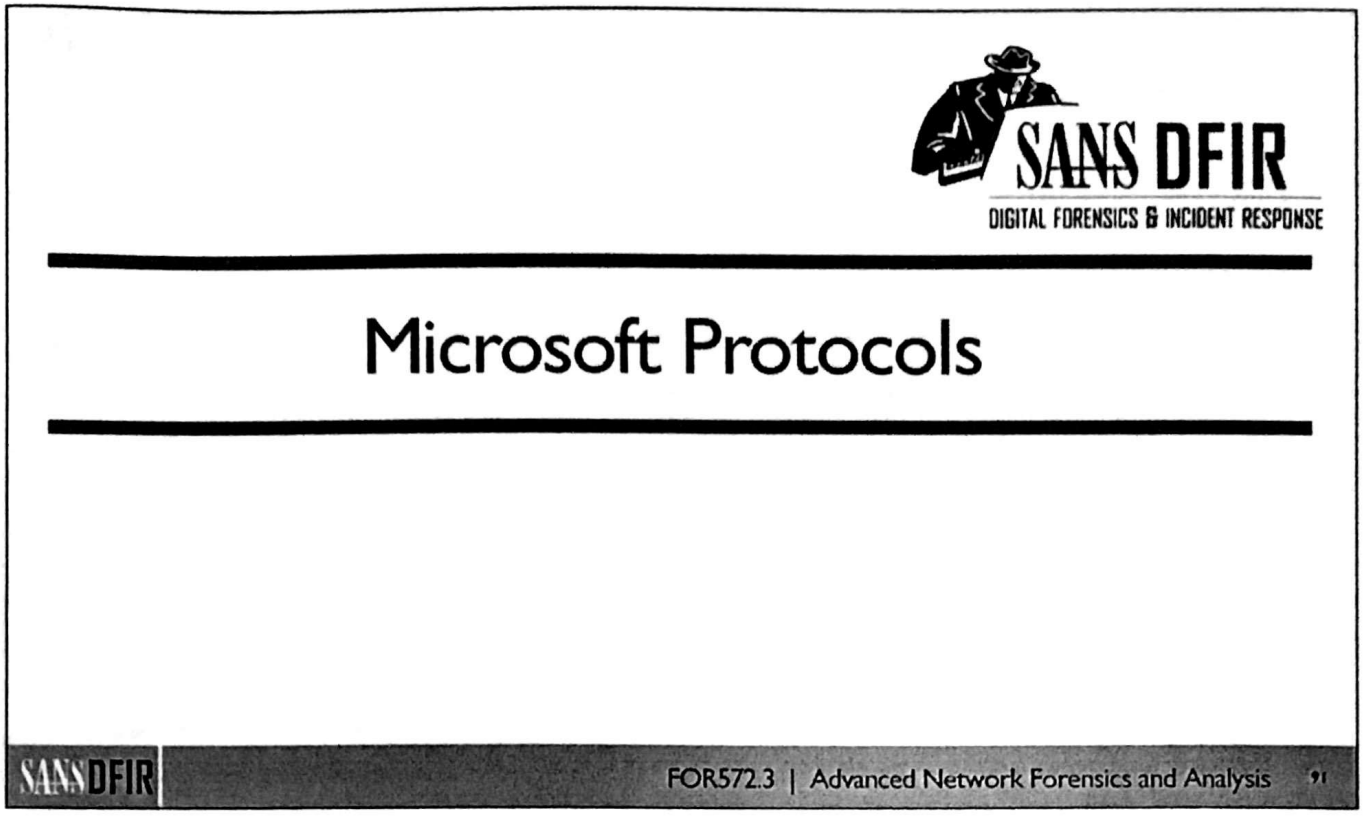
- Identify attacker's lateral movement using only NetFlow evidence
- Establish patterns of activity that help characterize the attacker's behavior profile
- Understand and mitigate shortcomings of high-level evidence such as NetFlow data
- Determine additional sources of evidence that may provide more detailed findings

This page intentionally left blank.

Lab 07 Takeaways: Tracking Lateral Movement with NetFlow

- Even without full content or logs, NetFlow can be an invaluable source of network-based evidence
- Despite its shortfalls, NetFlow findings can quickly identify events that need additional attention
- Observing systems that communicate on protocols and ports that are not consistent with standard client-server behavior is an important finding

This page intentionally left blank.



This page intentionally left blank.

MS Protocols and Network Forensics

- SMB allows for remote access to files
- Advanced attackers use captured credentials to access users regular files
- Attackers use mapped drives and shared folders to compromise data and upload malware/exploits

Advanced and dedicated attackers will use only the minimum effort to compromise a system. Once the attackers have access to either user credentials or user hashes, they will use various tools to see what they can gather from the user's access. This will result in considerable volumes of SMB file and folder access, all of which are currently easy to be read and interpreted. However, the attackers' actions will appear different only if they are in a distinct time zone, and the attackers' actions occur when the user is known to be offline.

Regardless of where the source IP address indicates, the remote attackers often tend to work office hours in their own time zone. Therefore, understanding the time difference between the user and attacker will allow the analyst to understand the predictable behavior of the attackers. In prolonged attacks, cross-checking periods that seem to exhibit a lack of attacker activity against a list of world national holidays[1] may give an indication as to where the attacker lives or their home country.

By interviewing users whose accounts have been accessed by attackers as soon as possible, the analyst will better be able to determine which file accesses the user conducted and which the attacker conducted. At a technical level, the analyst will need to understand the data being accessed by SMB and e-mail flows (if the capture locations permit), but more importantly the analyst must track the user activity, the times the user logged on, the files they accessed, the places they went.

Attackers will often use network shares (over SMB) to move their malware to locations where it will be accessed and run by unsuspecting users. This can be easily detected by observing the users smb2 access to files and folders, and malware uploaded can be reassembled from packet captures (fortunately malware tends to be small in size).

References:

[1] http://for572.com/1pz9u

Windows Architecture

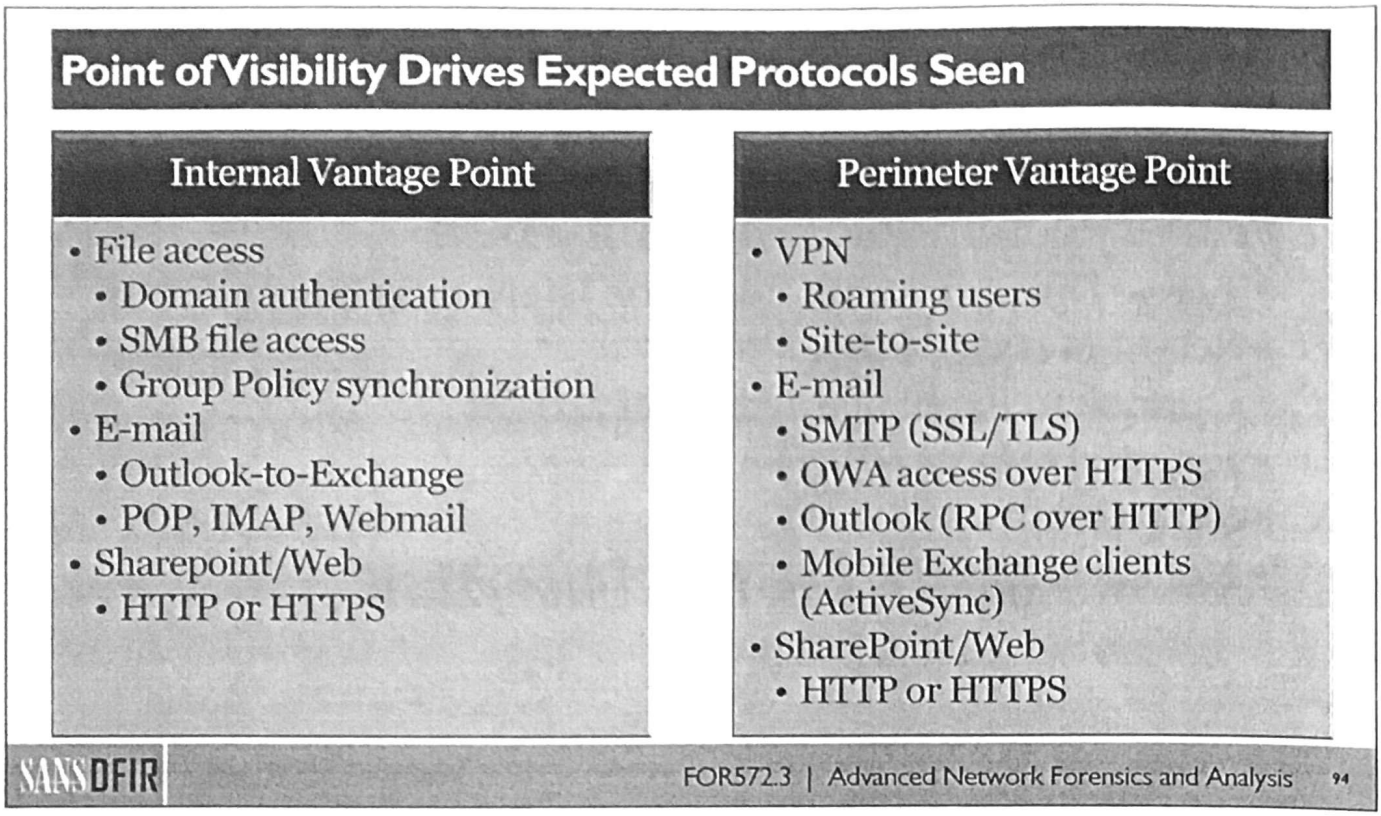
- Client-server deployment
- Primary communication/protocols
  - Active Directory Authentication (Kerberos/NTLM)
  - Server Message Block
  - Outlook to Exchange synchronizations
  - External Clients (VPN)
  - SharePoint
  - Not covering NTP, FTP, DNS, DHCP, TFTP

Microsoft Windows operating systems are normally deployed to corporate environments using a client/server model. This enables centralized administration, audit and accounting functions. However, some smaller, specialized or start-up organizations without a dedicated IT support team will sometimes run the client OS in a workgroup mode. Because all services or resources are network-enabled, they can be accessed without any particular client-side configuration. It is important that analysts know what is supposed to be in the environment so they can more easily identify what should not be present.

For example, in a domain-based client/server environment, it would be unusual to see workgroup systems. An analyst observing workgroup-based SMB messages in this environment should inquire as to where that system is, who owns it, and why it is not part of the domain. It might be a personal laptop or external consultant's device that may not be authorized to connect to the LAN. Worse, the untrusted and unmanaged system might be riddled with malware.

Within the Windows Domain, analysts will observe large volumes of Active Directory-related traffic. Therefore, understanding these contents is key to understanding the environment. Within MS Windows environments, Outlook is the most common e-mail client and Exchange the most common enterprise e-mail/calendar server. There are multiple rings of communication in this arena. We will examine the client/server communications, but also those between the core and edge (or transport) servers and between internal servers. The mail process also involves external mail servers (usually on the Internet), and mobile clients using the ActiveSync protocol. Finally, we will touch upon how a VPN affects such connections when connecting devices outside the physical LAN.

As mentioned on the previous slide, we will cover the different aspects of SharePoint, but not the HTTP/HTTPS foundations. Other protocols handled by the OS such as NTP, FTP, DNS, DHCP, and TFTP are generic networking protocols rather than Microsoft-specific protocols. These will be not be re-addressed in this section, but are commonly seen in MS networking environments.



The location of data capture or observation directly affects the type of data you can expect to see, as well as how it may be encoded.

On the LAN, Active Directory (AD) traffic flows in abundance. Authorized e-mail users (i.e., on domain-administered client systems) most likely use the Outlook client software to connect with an Exchange server. However, internal users may also access external mail servers using the Post Office Protocol (POP) or Internet Mail Access Protocol (IMAP) protocols, or via webmail using the HTTP/HTTPS protocols. These protocols could also be used by non-domain-administered systems to connect to outside mail systems.

The internal LAN will probably have internal web and SharePoint sites for users to exchange information. Often, these are deployed without HTTPS encryption, so all content will be visible on the wire. Furthermore, if the authentication mechanism is poorly implemented on an internal website, the user's AD credentials could also be compromised.

Captures done on the outside of the LAN, such as on exit routes or in DMZs, will see different traffic than that observed on the internal LAN.

E-mail may be transmitted in clear (both inbound and outbound), but the increasing use of SSL/TLS is seen in mail traffic as well. Opportunistic encryption configurations, in which servers first try to use secured channels before falling back to unencrypted communications, are becoming more widely used, but are still far from universal.

Microsoft also provides an excellent resource on the overall mail architecture in an Exchange environment.[1]

There could be some inbound HTTPS to the Exchange server (usually the Client Access Server (CAS)). However, because it would be encrypted, an analyst would be unable to distinguish it from Outlook Web Access (OWA), Outlook Anywhere (Exchange RPC over HTTP), or mobile clients (ActiveSync) unless these all have different URLs or entry routes. Having a full understanding of the environment is critical in characterizing such activity.

When observing Microsoft-based web traffic, a good deal of the legitimate traffic will be HTTP and some will be HTTPS. However, most of today's malware tends to avoid HTTPS usage, relying instead on obfuscation, "proprietary" APIs, and application-level data encryption (such as encrypted RAR archives).

References:

[1] http://for572.com/ob26a

Outlook-to-Exchange Email Traffic

- Uses RPC in domain configuration
  - XOR with '0xA5'
  - Captured synchronization traffic can reveal some clear text
- Other protocols:
  - IMAP/POP3 – Usually use SSL/TLS
  - SMTP – Needed to send email with IMAP/POP3 accounts
    - Increasingly uses SSL/TLS

In a normal domain configuration, the Outlook client uses RPC to access the Exchange server's information store and retrieve its messages. This RPC communication is encoded in places, but considerable chunks of e-mail data can be extracted by XORing a pcap file's payload with the '0xA5' byte. Even without a full understanding of the fields and data structures, the content of the e-mail can be read with the human eye.

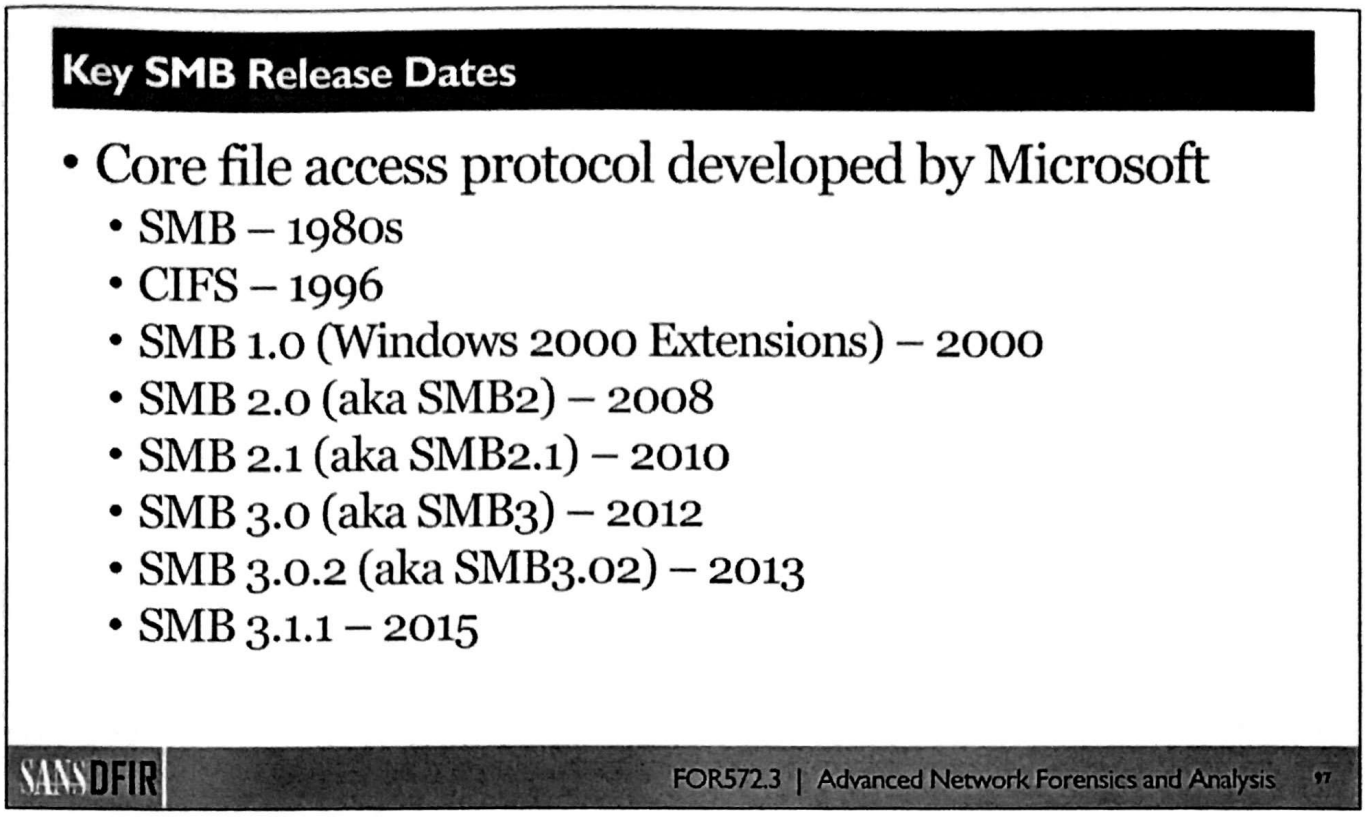
As a multipurpose e-mail client, Outlook can be configured to simultaneously access multiple e-mail servers using various protocols. Non-Exchange access methods include POP and IMAP. Although both POP and IMAP user credentials are passed in clear plaintext, both can be sent over SSL/TLS connections. However, this is not always the case and thus user credentials can often be captured over the wire.

Such credential interception presents a significant risk when a user has manually synchronized his e-mail and domain passwords. As a means of SPAM-prevention, most SMTP servers require authentication. This may be with Secure Password Authentication (SPA), which is an NTLM-based method, or other authentication methods.

For further reading, Microsoft's specification for this Wire Format Protocol is available from MSDN.[1]

References:

[1] http://for572.com/msup0



Server Message Block (SMB) has been the core file, folder, printer, and Inter Process Communication (IPC) protocol of the Windows Operating System family since Windows NT. If you have ever browsed your local system's shares (for a Windows OS in the default configuration), you may remember a share named "IPC$".

Introduced to allow multiple clients to access remote files as if they were on the local system, the protocol was a significant selling point in the networking of systems. Touted as a benefit over sneaker-net (where the user walked from machine to machine in their sneakers), SMB and its newer derivative CIFS provides a command set that allows a user local to create, edit, modify and deleted remote files and folders.

The CIFS terminology was introduced when MS submitted the protocol to the IETF, and became the version of SMB that shipped with MS NT4.

Since CIFS, MS has expanded and extended the protocol, integrating Kerberos authentication, SMB signing, and shadow copying.

Microsoft has frequently updated the SMB protocol stack over the years, the list below outlines the key milestones and years. For the security architect's consideration, when asked to implement CIFS or SMB1 it is worth considering the levels of LAN/OS security that were prevalent at that time; the CIFS specification is dated 1996!

References:

`http://for572.com/jetd5`

`http://for572.com/tv9zn`

`http://for572.com/8yz0d`

Version Comparison

- Early SMB used various protocols and ports
  - NetBIOS over IPX (called NBIPX)
  - NetBIOS over TCP (TCP ports 137, 138, 139)
- Current OSes support SMB 2.0 and SMB 3.x
  - TCP (Port 445 without NetBIOS)
- Windows 10 and Server 2016 use 3.1.1
  - TCP (Port 445 without NetBIOS)
- Newer versions may still use legacy ports for backward compatibility

SMB has used several transport protocols over the years, and in legacy environments these may still be present:

CIFS (i.e., SMB1) NetBIOS over IPX/SPX – CIFS NetBIOS used the Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX) transport protocol suite provided by Novell.
CIFS (i.e., SMB1) NetBIOS over TCP – Using TCP for transport ports 137, 138, and 139 open for IPC$ and Null Sessions that both utilized CIFS SMB2 however, uses only TCP through port 445.

After SMB and CIFS, Microsoft released updates to the SMB protocol in line with the other operating system improvements.

SMB2.1 was released with the update of Windows Server 2008 R2 and release of Windows 7, with some minor protocol performance improvements.

SMB 3.0 was introduced with Windows 8 and was designed to operate with Windows Server 2012. This latest version provides a significant update from SMB 2.1 and many new features have been added with its release. For network forensic analysts, the most important is that of session encryption.

Windows 8.1 and Windows Server 2012 R2 introduced support for SMB 3.0.2, and Windows 10 and the preview releases for Windows Server 2016 natively use SMB 3.1.1.

Identify SMB Clients by Version

Operating System	Win XP/2k, Svr 2003	Win Vista, Svr 2008	Win 7, Svr 2008 R2	Win 8, Svr 2012	Win 8.1, Svr 2012 R2	Win 10, Svr 2016
Win XP/2k, Svr 2003	SMB 1.0	SMB 1.0	SMB 1.0	SMB 1.0	SMB 1.0	SMB 1.0
Win Vista, Svr 2008	SMB 1.0	SMB 2.0.2	SMB 2.0.2	SMB 2.0.2	SMB 2.0.2	SMB 2.0.2
Win 7, Svr 2008 R2	SMB 1.0	SMB 2.0.2	SMB 2.1	SMB 2.1	SMB 2.1	SMB 2.1
Win 8, Svr 2012	SMB 1.0	SMB 2.0.2	SMB 2.1	SMB 3.0	SMB 3.0	SMB 3.0
Win 8.1, Svr 2012 R2	SMB 1.0	SMB 2.0.2	SMB 2.1	SMB 3.0	SMB 3.0.2	SMB 3.0.2
Win 10, Svr 2016	SMB 1.0	SMB 2.0.2	SMB 2.1	SMB 3.0	SMB 3.0.2	SMB 3.1.1

This table shows what "dialect" (SMB terminology for its version) should be selected between the various permutations of connections between Microsoft Windows hosts. Notice how the inclusion of XP or 2003 reduces the dialect to SMB1 which has a significant network overhead over later more efficient versions.

The investigator can identify the end client system simply by looking at the SMB version it is broadcasting. When read in conjunction with the next slide, the forensicator can identify if an end system is a Windows client or something pretending to be a client (i.e. a macOS client or a Linux OS with a SAMBA client installed).

References:
`http://for572.com/2v-o0`
`http://for572.com/jhtzw`

Who Else Uses What Version?

Vendor	Product	SMB Version
Apple	macOS 10.12 Sierra	SMB 3.0
Apple	OS X 10.11 El Capitan, 10.10 Yosemite	SMB 3.0
Apple	OS X 10.9 Mavericks	SMB 2.0 (Replacing AFP!)
Apple	OS X 10.7–10.8 Lion/Mountain Lion	SMB 1.0 (Via Apple SMBX)
EMC	VNX, Isilon 7.1.1	SMB 3.0
EMC	Isilon 7.0	SMB 2.1
EMC	Isilon 6.5	SMB 2
EMC	Older versions	CIFS/SMB 1.x
NetApp	Data ONTAP 8.2	SMB 3.0
NetApp	Data ONTAP 8.1	SMB 2.1
NetApp	Data ONTAP 7.3.1	SMB 2.0
NetApp	Older versions	CIFS/SMB 1.0
Samba	Samba 4.1, 4.2	SMB 3.0
Samba	Samba 3.6	SMB 2.0 (with some 2.1)
Samba	Older versions	CIFS/SMB 1.0

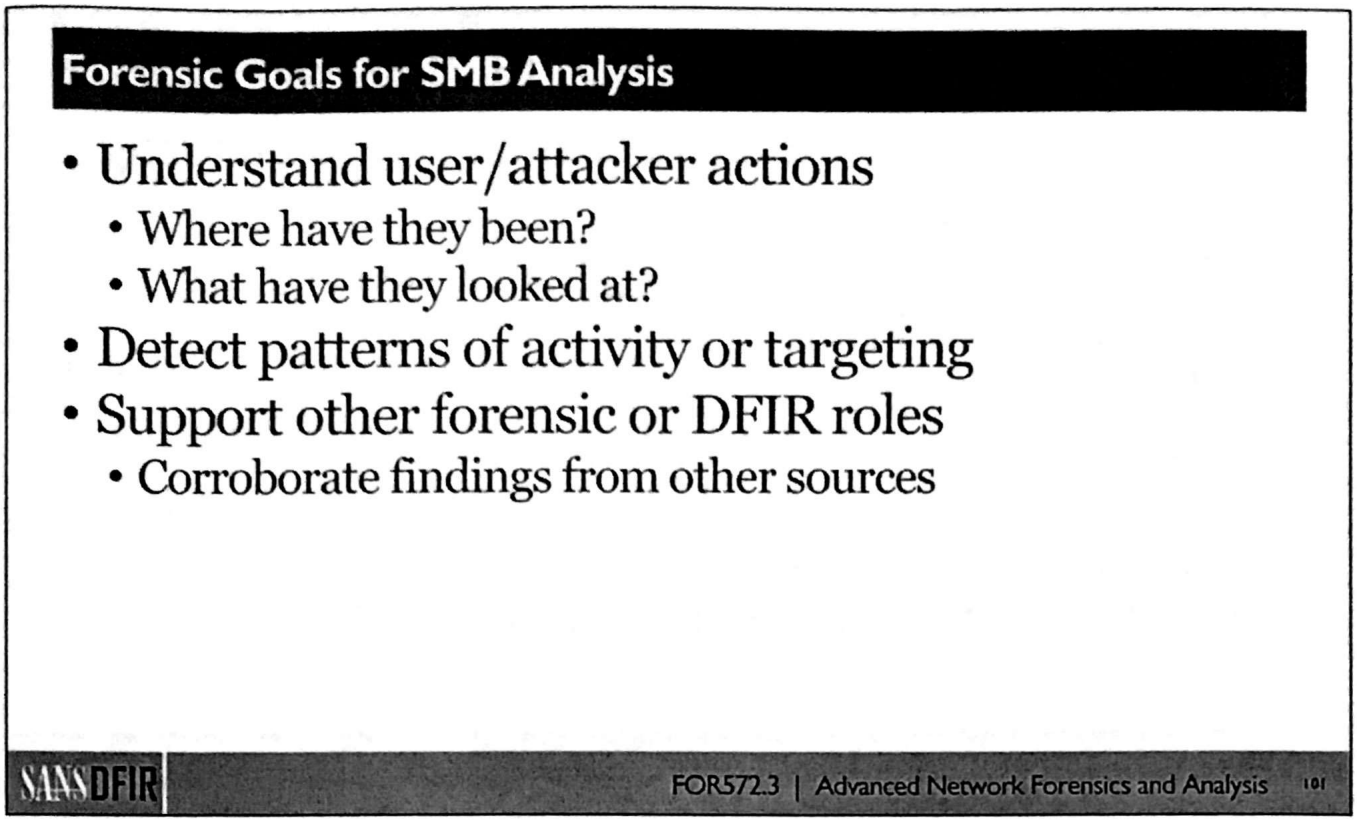
As can be seen from the table above, if you have a multi-platform environment, you will probably have old and insecure (non-SMB3) versions of SMB on your network.

If you operate in an all-Windows 8+ environment, SMB3 may be all that is in use. However, if other operating systems or network level storage solutions are in use it is common for security to go to the lowest common denominator and the weaker protocol versions will be in play.

From a network forensic perspective, we can leverage the insecure aspects of SMB1 to better characterize a user's activity. Of course, so could an attacker.

Correct as of November 2015 (source: SNIA SMB3 Remote File Protocol Presentation by Jose Barreto of Microsoft (author-approved reproduction)), augmented for FOR572.

References:
`http://for572.com/8yz0d`



When examining SMB traffic, an analyst can provide extremely valuable insight to the files and other domain-based transactions that occur within the environment. Because SMB is a filesystem-focused protocol, these insights often mirror those from the filesystem and even memory DFIR sub-disciplines.

For example, it is possible to determine what files an attacker looked at, opened, or copied to another location on the network. Additionally, after an attacker-controlled account has been identified, searching for all activity performed with that username can quickly identify the scope of an attacker's footprint in the victim's environment.

When examining the directory and file paths an attacker traverses, it may be possible to characterize their intent. For example, an attacker who searches an entire environment for a share named "`/finance_documents/`" would have a different operational goal than one looking for "`/blueprints/`". This different targeting profile may lead to different investigative approaches, inclinations for law enforcement involvement, or engagement of counter-intelligence resources.

Network-based SMB analysis is also suited to supplement system-based DFIR roles. This is because we see artifacts such as filesystem paths, sizes, MACB times, and host-based process IDs in the SMB traffic. We can pass these findings to team members performing the disk, memory, or log file analysis tasks, and vice versa.

Don't Over-Tech the Attack

- APT methods are simple but effective—often using inherent SMB functions
  - Log in to the user's workstation from another system (internal or via VPN)
  - Read the user's files
  - Surf to internal websites and access data
  - Copy and paste from file shares or network storage to attacker-controlled locations
  - Exfiltrate data by any means available

Remember that attackers will use the simplest, easiest, and most effective ways of accessing data; with the greatest weight towards easiest.

Attackers will use malware if they have to, exploits if they must, but will continue to access and use regular accounts if this will allow them to achieve their mission. Put simply: No professional attacker will throw around their expensive 0-day or rootkit unless it's absolutely needed. And in most cases, it's not at all needed.

Therefore, when dealing with smart attackers, we often look for "normal" activity in unusual places or at unusual times. Sometimes, that unusual activity is simply mounting a remote drive or using RDP to access a system with the user's own credentials.

Data harvesting attacks can be simple reading of confidential internal websites, saving internal data or stealing documents that the user can regularly access. The exfiltration methods can be equally simple, including e-mail, website forum uploads, and cloud storage solutions (e.g., Dropbox/Sky Drive/Google Drive).

So don't overlook the obvious attack just because it is not special or technical; hackers are lazy, too. In fact, the most advanced attackers' tactics rely on using the least elegant methods possible to access a target. Why would they burn a sensitive and/or expensive exploit or technique if it's not needed? Although we'll cover OPSEC in greater detail later in the course, remember that good attackers use sound OPSEC as well.

Filter and Review SMB

- Filter larger captures for just SMB traffic
- Segment to smaller subset of traffic by time or host
- Load into Wireshark for initial characterization
- Apply a display filter
  - Hide GPO sync, SMB announcements, browser elections
- Read users'/attackers' actions in "Info" column
  - Tweak possible to do this with `tshark`
- Pivot to broader data set with `tshark`, bash scripts

A simple way to analyze SMB activity is to take the capture and apply the '`smb or smb2`' display filter.

A clean-up can be conducted by filtering up some of the GPO replication traffic by progressively adding more restrictive filters for the "`gpt.ini`" and "`sysvol`" files.

With this in place, it is a matter of looking at the info column. The Wireshark GUI's "Info" column, however, requires a tweak to display from `tshark`. The following command will display the frame number and "Info" column contents for each frame[1]:

```
$ tshark -o column.format:'"No.", "%m", "Info", "%i"' ...
```

`tshark` can be used to filter traffic down to packets that are associated (source or destination) with ports 137, 138, 139, and 445. Note: Applying a read filter of '`smb or smb2`' in `tshark` will not capture other traffic that uses SMB for transport (such as remote registry access, SAMR, LSARPC, DCE/RPC, and more). To minimize traffic to that which will likely be the most relevant to the analytic process, remember to test your filters on sample captures first.

References:

[1] `http://for572.com/db3fg`

SMB2 Commands

- Protocol negotiation
  - NegotiateProtocol, SessionSetup, SessionLogoff, TreeConnect, TreeDisconnect
- File, directory, and volume access
  - Create, Close, Flush, Read, Write, Lock, Ioctl, Cancel, Find, Notify, GetInfo, SetInfo
- Other
  - KeepAlive, Break

The SMB2 command set consists of these 19 commands,[1] which represent a significant streamlining compared to SMB1.[2] However, some of these commands can perform multiple functions depending on the status of a resource at the time the command is issued.

References:

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

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

"Typical" SMB2 File Access Session

- Negotiate Protocol
- Session Setup AndX
- Tree Connect AndX
- Trans2 (Multi-use)
  - Query File Basic Info
- NT Create AndX
  - Open **or** create!
- Locking AndX (lock)
- Read AndX
  - May require multiple exchanges per file
- Locking AndX (unlock)
- Close
- Tree Disconnect
- Logoff AndX

Many different paths through SMB operations!

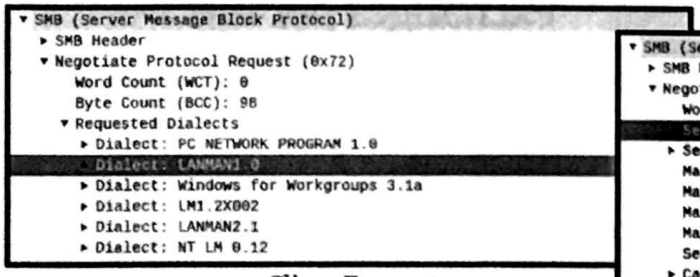
The extensive nature of the SMB2 protocol allows for thousands of different usage sequences. To illustrate a common set of commands that provide file access functionality, though, we will examine the steps above in greater detail. The upcoming exercise will also use this sequence, which is often seen in operational environments as well.

In operational usage, each command above consists of a request/response message pair.

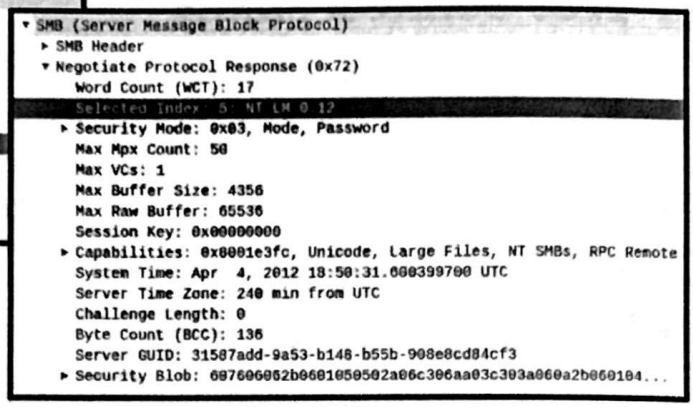
It is important to note that this is not a rigid sequence of events for a file access session. Abrupt actions might cause a session to end without unlocking a file, for example. Additionally, network conditions might cause some messages to occur out of sequence. The walk-through in this section serves as a general guideline rather than an absolute process map.

SMB: Protocol Negotiation

- Protocol Negotiation Request/Response
- Wireshark display filter: "smb.cmd==0x72"



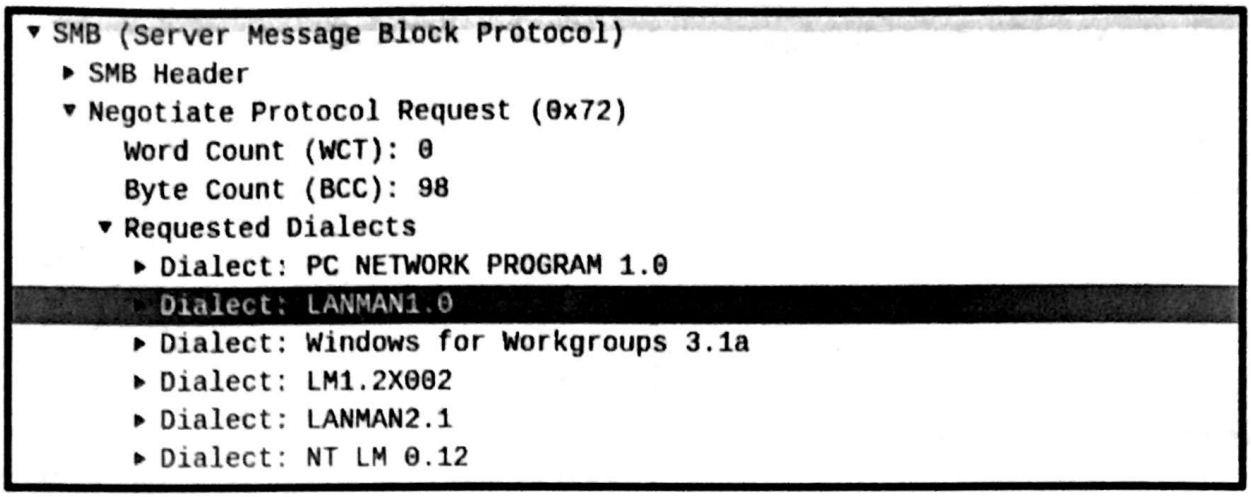
Client Request



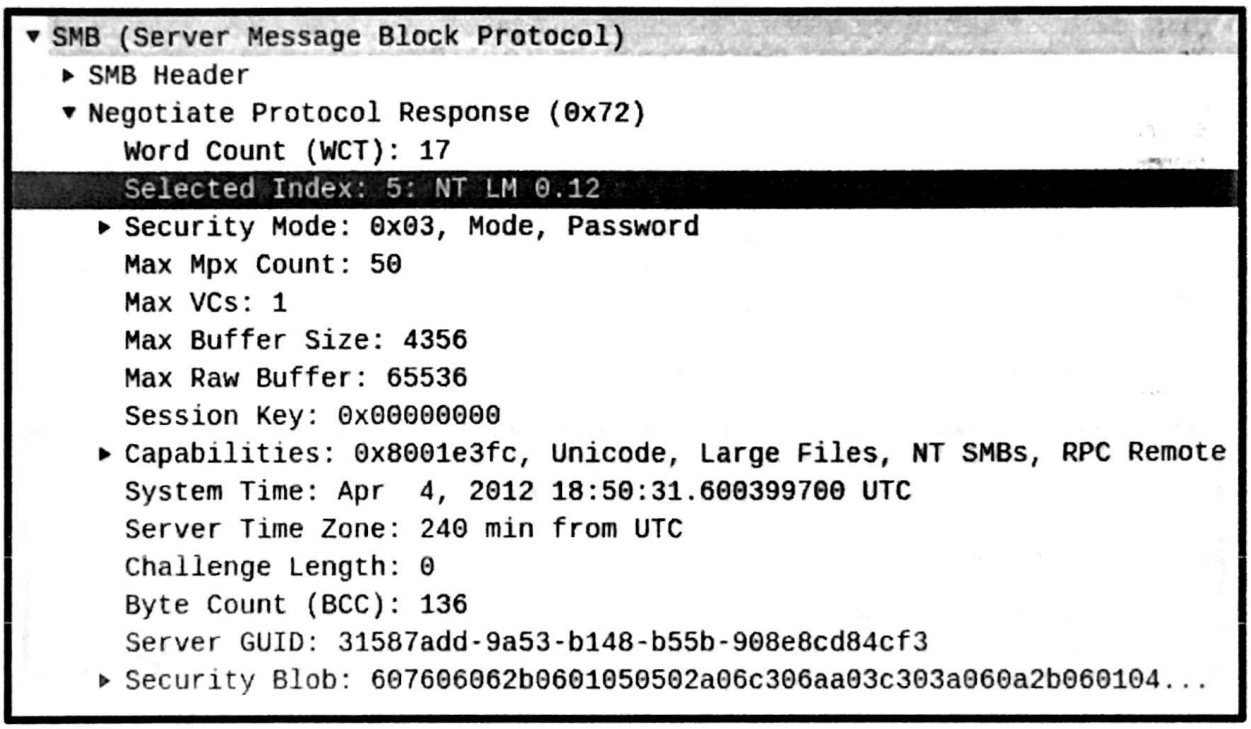
Server Response

These commands initiate the connection and involve the client (initiator) sending a list of the supported dialects. In this case, the client will happily negotiate dialects that date back several decades because the system has not been configured to prefer a more recent version. This enables backward compatibility because the server is supposed to select the strongest common dialect. The screenshot shows these fields in Wireshark's packet details and packet bytes panes. Because the client system in this case is running Windows XP SP3, it should not be a surprise to see it requesting older, predominantly LANMAN-based protocols.

The server's response packet includes several important bit flag fields, which identify the parameters the server will allow for the remainder of this session. These include the selected SMB dialect and the server's current date/time and time zone.



Protocol Negotiation: Client Request



Protocol Negotiation: Server Response

SMB: Session Establishment

- Session Setup AndX Request/Response

```
▾ GSS-API Generic Security Service Application Program Interface
  ▾ Simple Protected Negotiation
    ▾ negTokenTarg
        responseToken: 4e544c4d535350000300000001800180a0000008180
      ▾ NTLM Secure Service Provider
          NTLMSSP identifier: NTLMSSP
          NTLM Message Type: NTLMSSP_AUTH (0x00000003)
        ▸ Lan Manager Response: b275d2e7ca442baa0000000000000000000
          LMv2 Client Challenge: b275d2e7ca442baa
        ▸ NTLM Response: 9c670e19cef29dd42fbd29fe1a01236962b07042
        ▸ Domain name: shieldbase
          User name: vibranium
        ▸ Host name: WKS-WINXP32BIT
          Session Key: Empty
        ▸ Negotiate Flags: 0xa2888205, Negotiate 56, Negotiate 128
        ▸ Version 5.1 (Build 2600); NTLM Current Revision 15
Native OS: Windows 2002 Service Pack 3 2600
Native LAN Manager: Windows 2002 5.1
Primary Domain:
```

Client Request

- Display filter: "smb.cmd == 0x73"
- User authentication
- Client sends Process ID (PID); server assigns User ID (UID)

```
▾ GSS-API Generic Security Service Application Program Interface
  ▾ Simple Protected Negotiation
    ▾ negTokenTarg
        negResult: accept-completed (0)
Native OS: Windows 7 Ultimate 7601 Service Pack 1
Native LAN Manager: Windows 7 Ultimate 6.1
```

Server Response

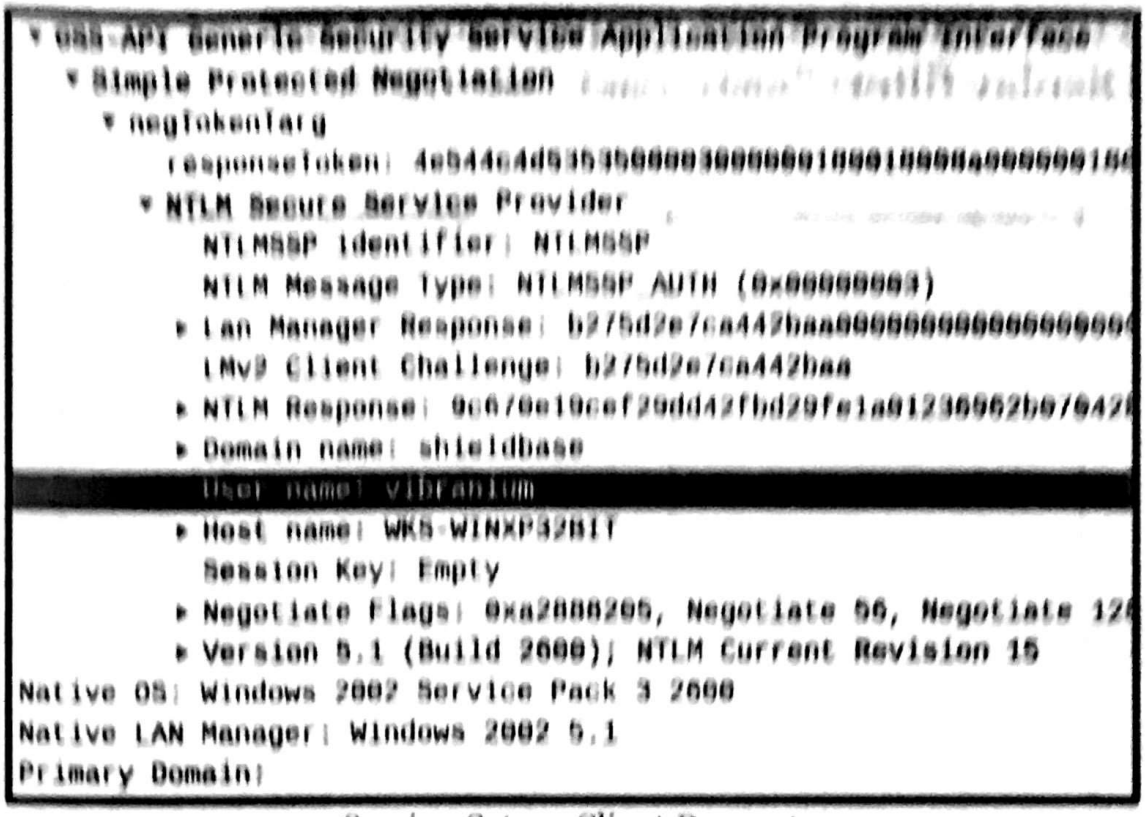
After negotiating the communication parameters, the client sends login credentials according to the most preferable common method agreed upon above. This step uses the "Session Setup AndX" command. Microsoft added the "AndX" extension to an early release of the SMB specification to reduce the number of packets sent across the network. In earlier revisions, the session had to be fully established before the client could request any services. However, the updated SMBv1 specification allowed these to be compounded and SMBv2 allows them to be stacked together. Because the session setup usually requires several TCP packets to be exchanged, the decreased packet count is significant in any reasonably busy Windows environment.

The request message contains a number of data points that can be useful for an investigation—particularly the domain and username to be used for the session. Another useful item is the process ID number from the client system. Typically, low numbers typically indicate core operating system-level processes, while those over 1000 usually indicate a user-initiated request. This data point can prove very helpful in correlating network-based activity with disk- or memory-based forensic examinations or event log entries.

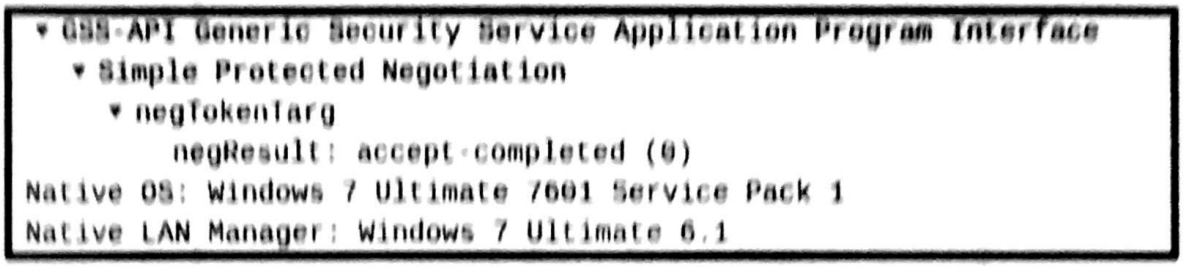
Because SMB activity may create many concurrent sessions between hosts, all requests are uniquely identified by their Multiplex ID value. This is a unique and random value that the client sets, and is generally a multiple of eight. With this method, the client can effectively pair a response packet with an outstanding request, continuing on to process the communication already underway. Understanding the nuances of this multiplexing/de-multiplexing process is critical when examining SMB traffic in any real-world environment.

This exchange also includes the Generic Security Service – Application Program Interface (GSS-API) data, which handles client authentication through the use of the Simple Protected Negotiation (SPNEGO) protocol. Several SPNEGO fields contain ASCII content, similar to those you previously observed in the "Follow TCP Stream" pop-up dialog box. The SPNEGO fields also include various hash values and tokens that permit domain authentication to occur.

The "Session Setup AndX Request" command includes additional data points, such as the operating system version and build, LANMAN protocol version, and sender's domain (if it has been established). In a typical session setup process, there are multiple exchanges of this message type, each of which covers a different phase of the setup. Not all artifacts are present in each message.



Session Setup: Client Request



Session Setup: Server Response

SMB: Accessing Services

- Tree Connect AndX Request/Response
- Display filter: "`smb.cmd == 0x75`"
- Server assigns Tree ID (TID)

```
    Reserved: 0000
    Tree ID: 0
    Process ID: 65279
    User ID: 2049
    Multiplex ID: 384
▾ Tree Connect AndX Request (0x75)
    Word Count (WCT): 4
    AndXCommand: No further commands (0xff)
    Reserved: 00
    AndXOffset: 80
  ▸ Flags: 0x0008, Extended Response
    Password Length: 1
    Byte Count (BCC): 37
    Password: 00
    Path: \\10.3.58.5\C$
    Service: ?????
```

Client Request

```
  ▾ Tree ID: 2049  (\\10.3.58.5\C$)
      [Path: \\10.3.58.5\C$]
      [Mapped in: 14]
    Process ID: 65279
    User ID: 2049
    Multiplex ID: 384
▾ Tree Connect AndX Response (0x75)
    Word Count (WCT): 7
    AndXCommand: No further commands (0xff)
    Reserved: 00
    AndXOffset: 62
  ▸ Optional Support: 0x0001, Search Bits, CS
  ▸ Maximal Share Access Rights
  ▸ Guest Maximal Share Access Rights
    Byte Count (BCC): 13
```

Server Response

After successfully establishing an SMB session, the client requests access to one of the server's services. For example, in this screenshot, the client requests to access the "C$" resource, akin to mapping a shared drive. Notice that the client includes the same UID value already received. Additionally, note the "Path" value under the "Tree ID" assignment in the server's response. Wireshark shows this in square brackets, which is an indicator that this literal value does not appear in the packet itself. However, Wireshark's SMB parser still provides the value in a field that can be addressed in the same way as content that is directly in the packet. This enables the analyst to use them in building display filters or printing tshark fields.

```
    Reserved: 0000
    Tree ID: 0
    Process ID: 65279
    User ID: 2049
    Multiplex ID: 384
▼ Tree Connect AndX Request (0x75)
    Word Count (WCT): 4
    AndXCommand: No further commands (0xff)
    Reserved: 00
    AndXOffset: 80
  ▶ Flags: 0x0008, Extended Response
    Password Length: 1
    Byte Count (BCC): 37
    Password: 00
    Path: \\10.3.58.5\C$
    Service: ?????
```

Tree Connect: Client Request

```
  ▼ Tree ID: 2049  (\\10.3.58.5\C$)
      [Path: \\10.3.58.5\C$]
      [Mapped in: 14]
    Process ID: 65279
    User ID: 2049
    Multiplex ID: 384
▼ Tree Connect AndX Response (0x75)
    Word Count (WCT): 7
    AndXCommand: No further commands (0xff)
    Reserved: 00
    AndXOffset: 62
  ▶ Optional Support: 0x0001, Search Bits, CS
  ▶ Maximal Share Access Rights
  ▶ Guest Maximal Share Access Rights
    Byte Count (BCC): 13
```

Tree Connect: Server Response

SMB: Obtain Directory/File Metadata

- Trans2 Request/Response
- Display filter: "`smb.cmd == 0x32 && ⏎`
  `smb.trans2.cmd == 0x0005 && ⏎`
  `smb.qpi_loi == 1004`"

```
Subcommand: QUERY_PATH_INFO (0x0005)
Byte Count (BCC): 91
Padding: 000000
▾ QUERY_PATH_INFO Parameters
    Level of Interest: Query File Basic Info (1004)
    Reserved: 00000000
    File Name: \users\nromanoff\documents\outlook files
```

Client Request

```
▾ QUERY_PATH_INFO Data
    Created: Nov 10, 2010 11:03:56.854290000 UTC
    Last Access: Apr  5, 2012 14:54:47.406319800 UTC
    Last Write: Apr  5, 2012 14:54:47.406319800 UTC
    Change: Apr  5, 2012 14:54:47.406319800 UTC
  ▾ File Attributes: 0x00000010
      .... .... .... .... .... .... .... ...0 = Read Only: NOT read only
      .... .... .... .... .... .... .... ..0. = Hidden: NOT hidden
      .... .... .... .... .... .... .... .0.. = System: NOT a system file/dir
      .... .... .... .... .... .... .... 0... = Volume ID: NOT a volume ID
      .... .... .... .... .... .... ...1 .... = Directory: DIRECTORY
      .... .... .... .... .... .... ..0. .... = Archive: Has NOT been modified sinc
      .... .... .... .... .... .... .0.. .... = Device: NOT a device
      .... .... .... .... .... .... 0... .... = Normal: Has some attribute set
      .... .... .... .... .... ...0 .... .... = Temporary: NOT a temporary file
      .... .... .... .... .... ..0. .... .... = Sparse: NOT a sparse file
```

Server Response (Directory)

In this phase of the session, the client requests information about the "`\users\nromanoff\documents\outlook files\`" directory. The server's response contains the familiar MACB data for the directory, which can be of particular use while tracking the state of a filesystem resource over time. The response also includes a bit field detailing the type of file object and its characteristics.

If the requested object had been a file, the data returned by the server would also include the size of the file on disk as the "End of File" value—an example of this is included on the following page in the book.

```
  Subcommand: QUERY_PATH_INFO (0x0005)
  Byte Count (BCC): 91
  Padding: 000000
▼ QUERY_PATH_INFO Parameters
    Level of Interest: Query File Basic Info (1004)
    Reserved: 00000000
    File Name: \users\nromanoff\documents\outlook files
```

Directory Metadata: Client Request

```
▼ QUERY_PATH_INFO Data
    Created: Nov 10, 2010 11:03:56.854290000 UTC
    Last Access: Apr  5, 2012 14:54:47.406319800 UTC
    Last Write: Apr  5, 2012 14:54:47.406319800 UTC
    Change: Apr  5, 2012 14:54:47.406319800 UTC
  ▼ File Attributes: 0x00000010
      .... .... .... .... .... .... .... ...0 = Read Only: NOT read only
      .... .... .... .... .... .... .... ..0. = Hidden: NOT hidden
      .... .... .... .... .... .... .... .0.. = System: NOT a system file/dir
      .... .... .... .... .... .... .... 0... = Volume ID: NOT a volume ID
      .... .... .... .... .... .... ...1 .... = Directory: DIRECTORY
      .... .... .... .... .... .... ..0. .... = Archive: Has NOT been modified sinc
      .... .... .... .... .... .... .0.. .... = Device: NOT a device
      .... .... .... .... .... .... 0... .... = Normal: Has some attribute set
      .... .... .... .... .... ...0 .... .... = Temporary: NOT a temporary file
      .... .... .... .... .... ..0. .... .... = Sparse: NOT a sparse file
```

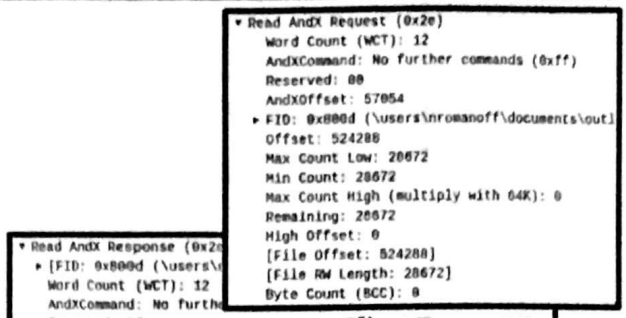
Directory Metadata: Server Response

```
▼ Find File Both Directory Info File: nromanoff@stark-resea
    Next Entry Offset: 168
    File Index: 0
    Created: Nov 10, 2010 11:03:57.010540000 UTC
    Last Access: Nov 10, 2010 11:03:57.010540000 UTC
    Last Write: Apr  5, 2012 15:31:25.421944800 UTC
    Change: Apr  5, 2012 15:31:25.421944800 UTC
    End Of File: 59442176
    Allocation Size: 59445248
  ▸ File Attributes: 0x00002020
    File Name Len: 74
    EA List Length: 0
    Short File Name Len: 24
    Reserved: 00
    Short File Name: NROMAN~1.PST
    File Name: nromanoff@stark-research-labs.com.pst
```

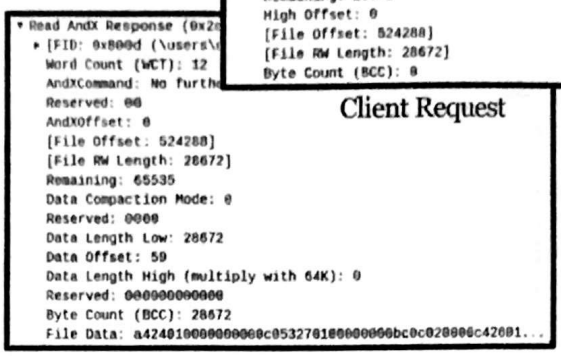
File Metadata: Server Response

SMB: Reading from a File

- Read AndX Request/Response
- Display filter: "`smb.cmd == 0x2e`"
- Partial file reads with byte offsets
  - Cannot "Follow TCP Stream" to get original file



Client Request



Server Response

To read file contents, the client supplies the FID, requested number of bytes, and offset at which to start reading. The server's corresponding response contains the file data itself, among other fields.

Because the allowable packet size is smaller than many files, a complete access might require hundreds or thousands of Read AndX Request/Response message pairs. Each message will contain the relevant SMB command codes, user/file/etc. ID number, and other related header data. For this reason, the Wireshark "Follow TCP Stream" function and other protocol-ignorant stream reconstruction tools cannot be used to extract a file from the SMB transaction. An SMB-aware carving process must be used instead.

```
▼ Read AndX Request (0x2e)
    Word Count (WCT): 12
    AndXCommand: No further commands (0xff)
    Reserved: 00
    AndXOffset: 57054
  ▶ FID: 0x800d (\users\nromanoff\documents\outl
    Offset: 524288
    Max Count Low: 28672
    Min Count: 28672
    Max Count High (multiply with 64K): 0
    Remaining: 28672
    High Offset: 0
    [File Offset: 524288]
    [File RW Length: 28672]
    Byte Count (BCC): 0
```

Reading File: Client Request

```
▼ Read AndX Response (0x2e)
  ▶ [FID: 0x800d (\users\nromanoff\documents\outlook files\nromanof
    Word Count (WCT): 12
    AndXCommand: No further commands (0xff)
    Reserved: 00
    AndXOffset: 0
    [File Offset: 524288]
    [File RW Length: 28672]
    Remaining: 65535
    Data Compaction Mode: 0
    Reserved: 0000
    Data Length Low: 28672
    Data Offset: 59
    Data Length High (multiply with 64K): 0
    Reserved: 000000000000
    Byte Count (BCC): 28672
    File Data: a424010000000000c053270100000000bc0c020000c42601...
```

Reading File: Server Response

SMB: Unlocking a File

- Locking AndX Request/Response
- Display filter: "`smb.cmd == 0x24`"
- Same operation as the lock request
- Action depends on locked/unlocked status at time of request

The client unlocks the file after it has completed its read operation, allowing other processes with appropriate permissions to modify or delete the resource. This is accomplished using the same command as the locking operation, so differentiating a "lock" rather than an "unlock" request depends on the resource's lock status at the time of the request.

SMB: Closing a File

- Close AndX Request/Response
- Display filter: "`smb.cmd == 0x04`"
- De-assigns FID value

After the file is unlocked, the client requests that the server close the file handle. The ephemeral FID value is de-assigned from the server's pool at this stage.

SMB: Tree Disconnection

- Tree Disconnect Request
- Display filter: "`smb.cmd == 0x71`"
- Signals the end of the SMB session on a per-resource basis
- De-assigns TID value

At session termination, each tree attachment requires an explicit disconnection request. These messages close the SMB session, and any future SMB activity will require the full negotiation, session setup, and tree connect message exchange to establish new values for the session-based PID and UID values, and the per-resource Tree ID value.

SMB: Logoff

- Logoff AndX Request
- Display filter: "`smb.cmd == 0x74`"
- Don't occur for each file access
  - Client can keep session open for additional activity if desired
- Signals de-assignment of UID and PID values

These messages terminate the SMB session if it will no longer be needed.

Each of these messages contains the session's established UID and PID, which are used to de-authenticate the client and free the server's connection pool for future SMB sessions.

Other Good SMB Filters

- The following are useful display filters:
  - `smb2.boot_time` (when the system booted)
  - `smb.file` (maps the filename to fileId)
  - `atsvc.opnum == 0`, `atsvc.atsvc_JobInfo.command`
    - Remote job scheduler—common and effective lateral spread methodology!
- These can contain clutter
  - Apply other filters to remove `gpt.ini`
  - File update requests (filename: `browser`)

There are over 100 Wireshark SMB- and SMB2-based filters. These often provide quick leads and clear indications of attacker actions. The "`smb2.boot_time`" field can show when a system was last booted, which can sometimes show abnormal boot or reboot activity. The "`smb.file`" field is extremely useful, as it shows the filename a client requested to access. When an attacker is attempting to purloin sensitive data, this field can show very clear patterns of their attempts to acquire data. Another common means attackers use to spread within an environment is by scheduling remote jobs using Microsoft's "ATSVC" scheduler. This is a payload inside of SMB traffic. The "`atsvc.opnum`" field indicates the type of ATSVC request. The possible values for this field are listed below:[1]

- 0 Add scheduled job
- 1 Delete scheduled job
- 2 Enumerate (list) scheduled jobs
- 3 Get information about a specific scheduled job

Even a first-time SMB reviewer will see there is a great deal of "typical" traffic that doesn't generally provide much investigative benefit. Notably, the group policy file synchronization and SMB file-update detection (aka "browser") traffic occurs quite frequently, and it can be distracting when looking for user- or attacker-driven traffic. The following display filter can be invaluable in hiding those events from view.

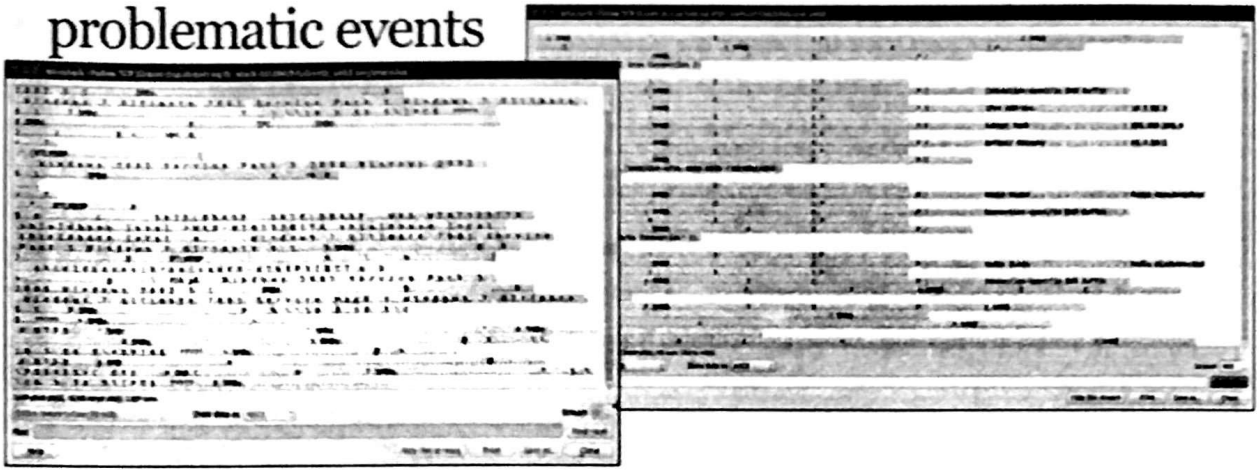
```
((smb2.filename) && !(smb2.filename == "browser")) && !(smb2.filename
contains "{31B2F340-016D-11D2-945F-00C04FB984F9}\\gpt.ini")
```

References:

[1] `http://for572.com/kb59j`

Spotting Oddities Through UTF-16

- All SMB 2+ traffic uses UTF-16 encoding
- "Follow TCP Stream" can be helpful to spot problematic events



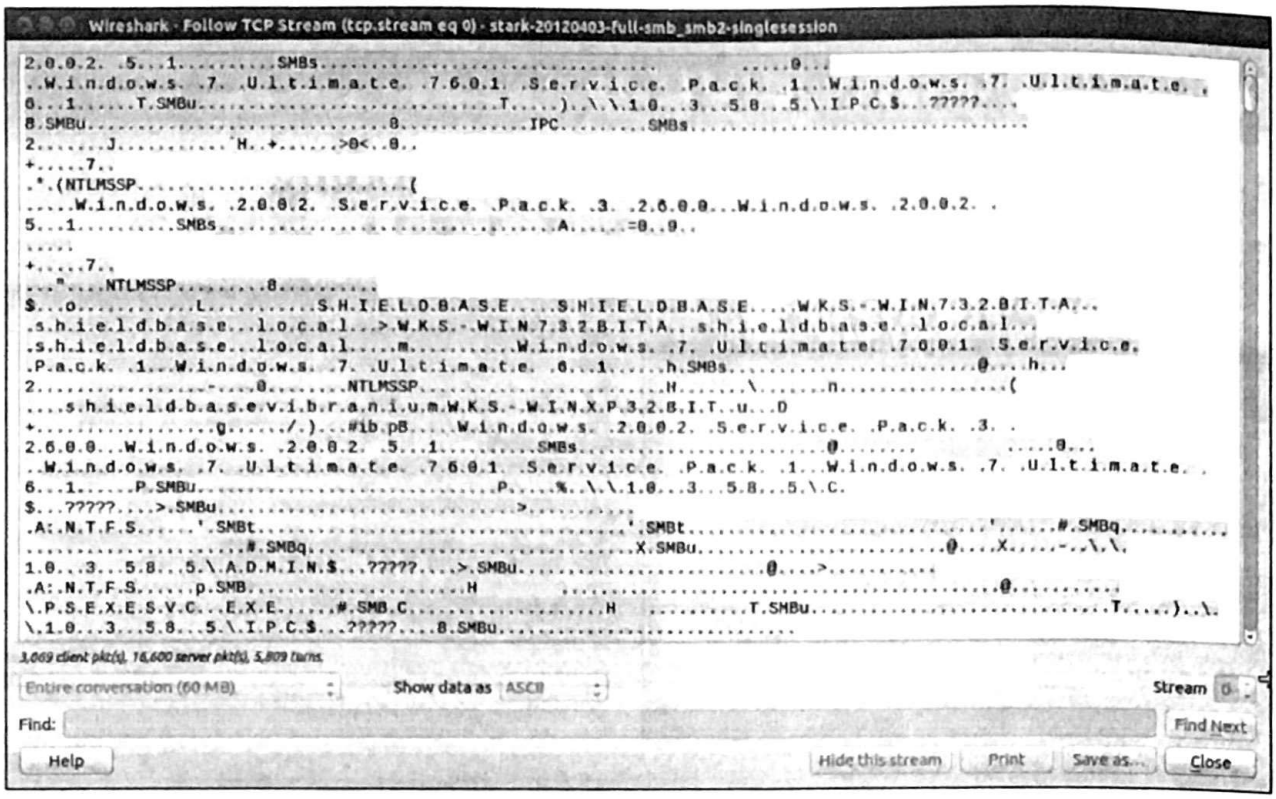
Using Wireshark's "Follow TCP Stream" feature, we can often find artifacts that warrant further investigation. Because SMB uses UTF-16 encoding, ASCII-printable characters are interspersed with NULL bytes (0x00), represented as the "." character in the stream view.

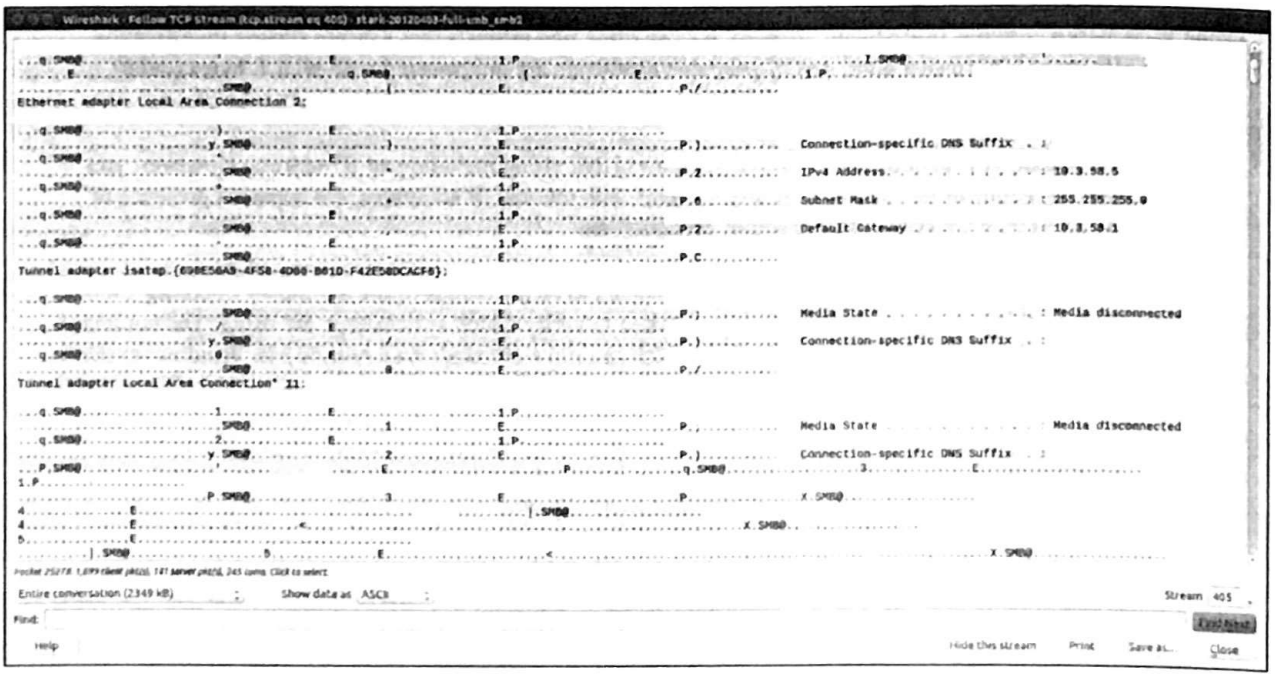
In the first screenshot, you should quickly spot a Windows UNC string including an IP address. However, you knew this SMB communication was between two systems with internal IP addresses, the apparent presence of "`\\10.3.58.5\IPC$`", a system management pseudo-share.

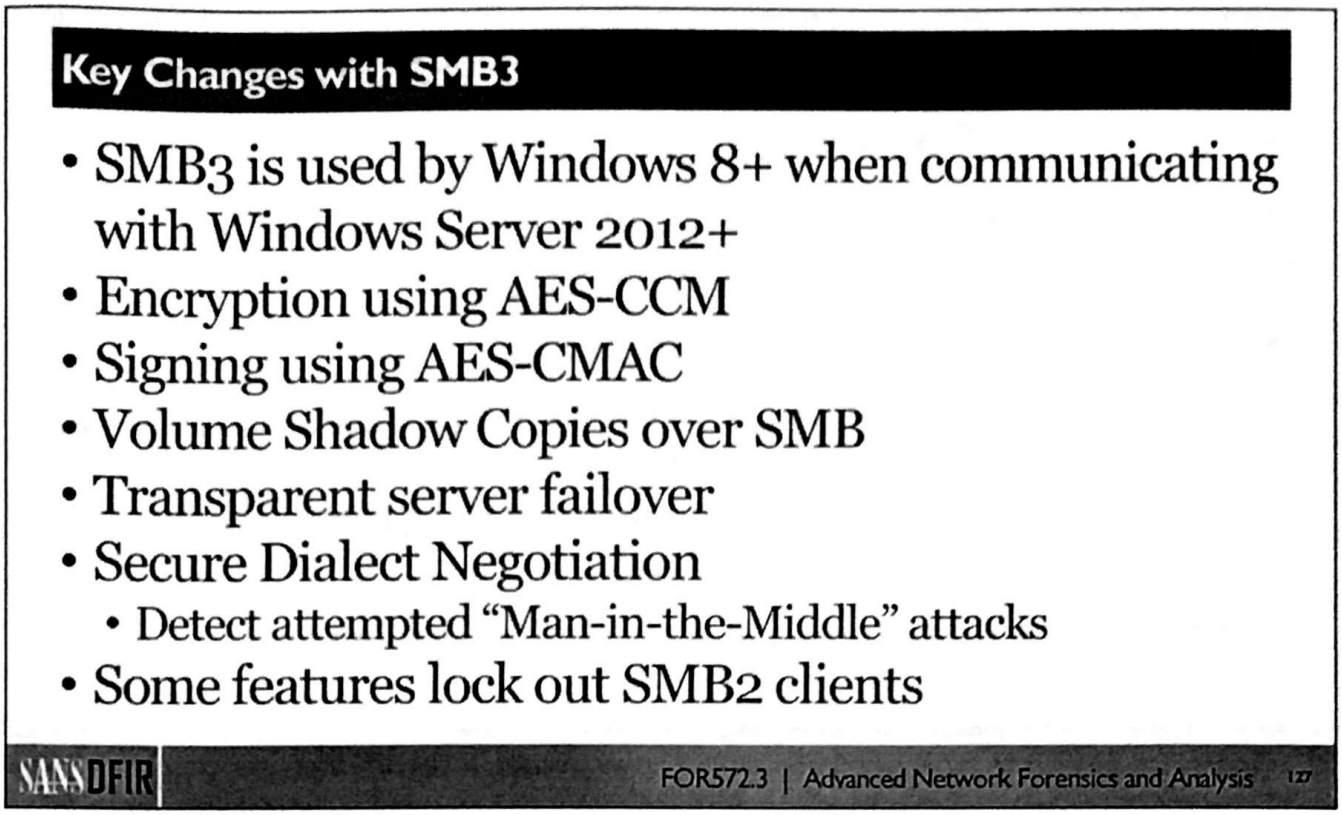
In the second screenshot, there are references to what appears to be the results of the Windows "ipconfig", too. Attackers often use this command when accessing a system to verify which system they are using. The command here was run via the Sysinternals[1][2] `psexec` tool, which can run commands on a remote MS Windows system.

References:

[1] `http://for572.com/t9p7c`
[2] `http://for572.com/-no3m`







SMB3 introduces optional encryption on a per-session basis, which uses AES-CCM (Advanced Encryption Standard (AES) in Counter with CBC-MAC (CCM) mode). This will protect file transfers and file details from being viewed by attackers and network analysts alike. Although not implemented by default at the time of this writing, the trend toward more heavily encrypted communications is easy to see in many commercial products.

The signing algorithm is changed from HMAC SHA-256 in previous SMB versions to AES-CMAC (Advanced Encryption Standard – Cipher-based Message Authentication Code) in SMBv3.[1]

The Secure Dialect Negotiation protects the establishment of sessions preventing attackers from conducting MITM attacks and forcing the downgrade of a SMB3 client to SMB1. The attacker's aim when forcing a session downgrade is for the client and server to use non-encrypted and less secure versions allowing for the interception of traffic and data.

Although SMB3 introduces a new command set, the general session process and overall means of analyzing the traffic are largely unchanged from what is discussed in this section and covered in the next lab. However, an outdated analysis platform will not provide the same visibility for this newer protocol variant. Additionally, the variations in available artifacts between SMB2 and SMB3 mean that many investigative processes will need to be updated as the new version becomes more commonly used in typical network environments.

References:

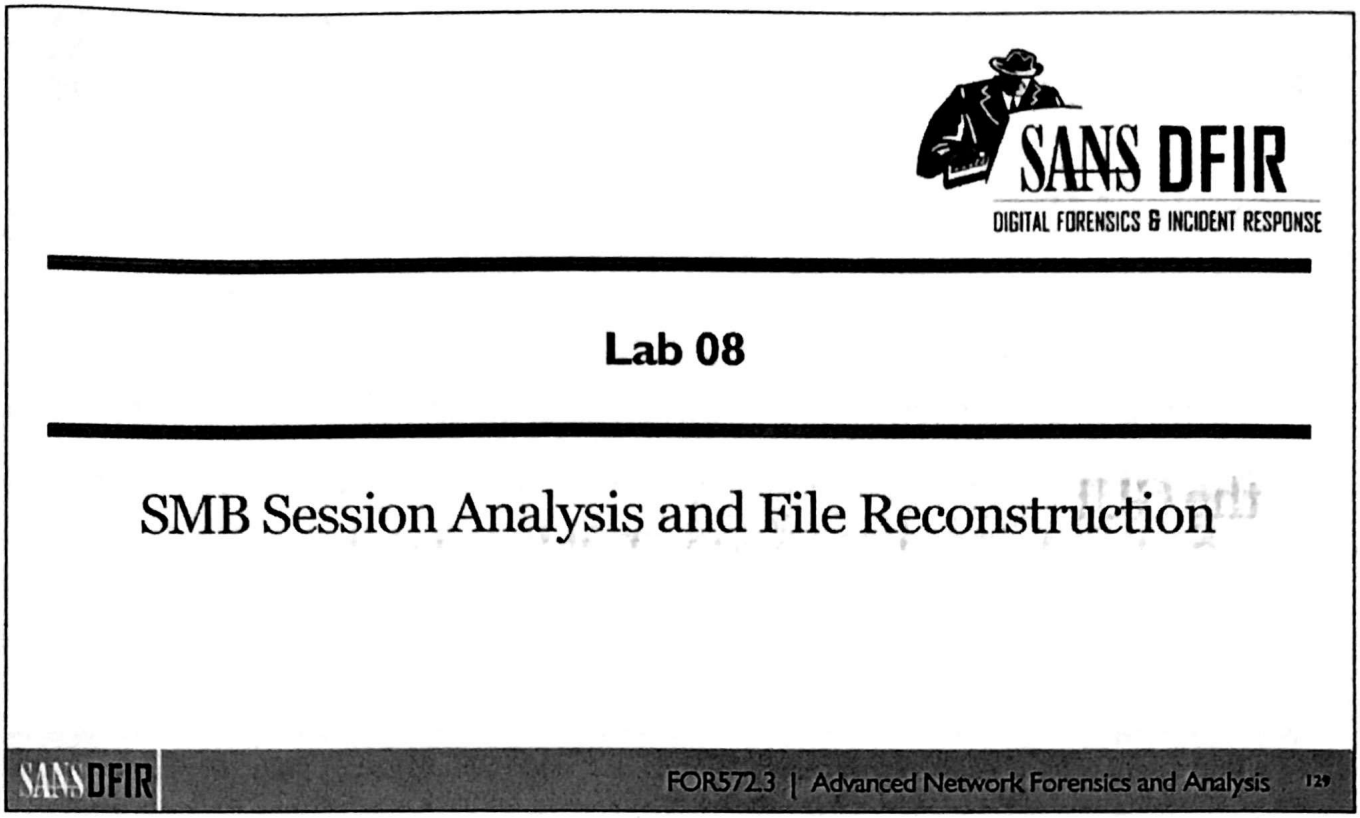
[1] `http://for572.com/wqdmv`

Behavioral Attack Indicators

- Classic attacker activity is to attack servers
- In advanced attacks, they then focus on users' systems and associated file shares
- Indicators of compromise include:
  - Client-to-client file shares over SMB
  - Client-to-server access at unusual hours
  - File share access by same account from different machines, LANs, and even countries
  - Enumeration of file shares and large copy actions

Attackers often target servers as they provide access to account management and centralized security. During an APT-style attack or an attack in which Intellectual Property data is the target, the attackers seek to harvest data stored on users' systems. This means we must rely more heavily on behavioral indicators that direct the investigator's attention to suspicious activity. These behaviors may include:

- Client-to-client SMB file access is unusual as users don't usually connect to other users' systems. File shares are generally on file servers, not client systems.
- Clients who access file shares at unusual times or at times when the user cannot account for their actions.
- In an organization where users cannot simultaneously log into two systems, identifying where a hash has been used from a system different to the primary client system can indicate pass-the-hash attacks.
- Users that browse through every directory on large SMB file shares can be detected by looking at the numbers of fileId references returned to a single IP address. However, this is open to false positives through antivirus scanning the files or users hunting for lost files.



This page intentionally left blank.

Lab 08 Objectives: SMB Session Analysis and File Reconstruction

- Understand typical sequence of events associated with SMB network file accesses
- Identify relevant SMB fields to generate helpful leads during an investigation
- Develop analytic processes with a small dataset in the GUI
  - Scale to larger data set with shell-based tools

This page intentionally left blank.

Lab 08 Takeaways: SMB Session Analysis and File Reconstruction

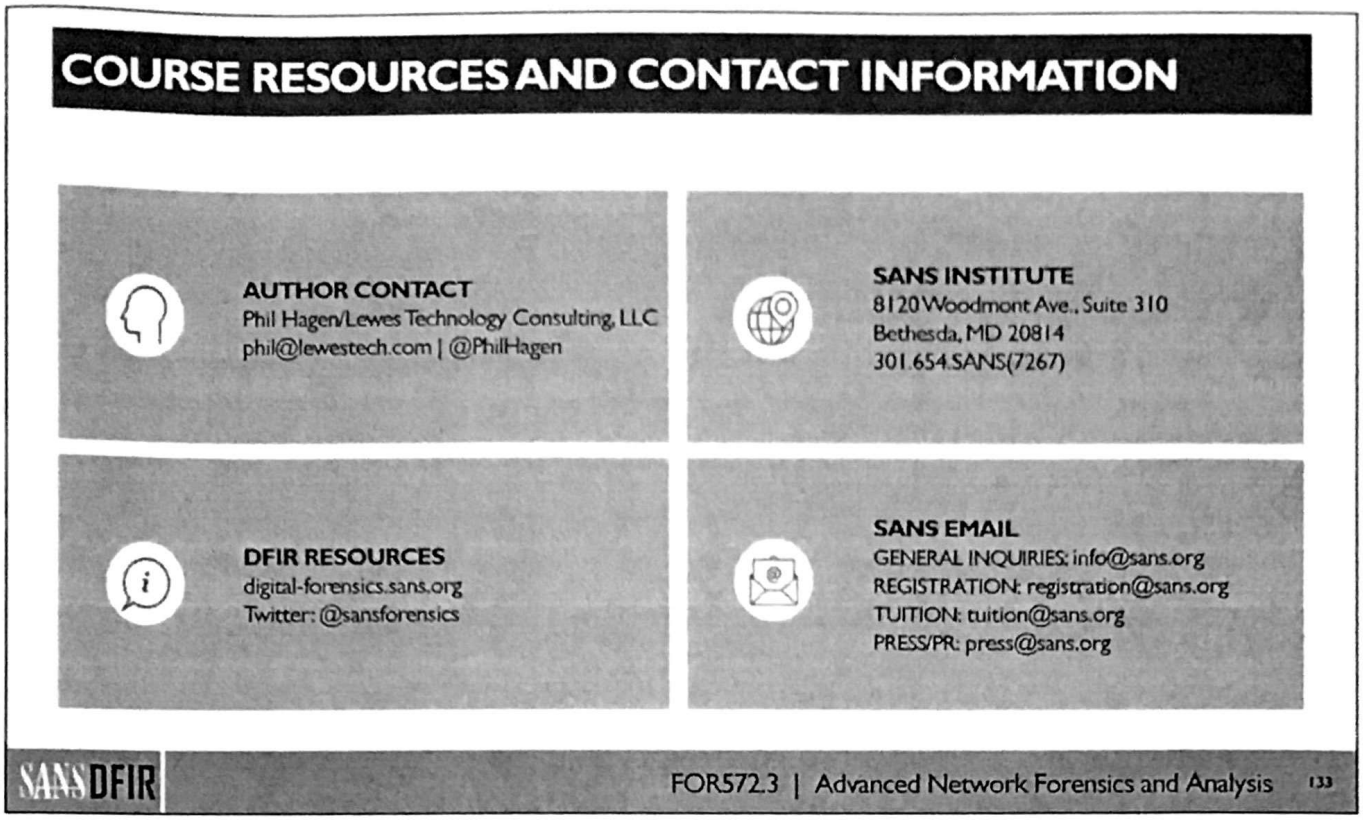
- SMB is a complex and comprehensive protocol
  - Hybrid ASCII/binary nature
  - Wireshark SMB parsers provide critical insight to these complex communications
- Backward compatibility will keep old/weak protocols in use for many years

This page intentionally left blank.

FOR572.3 | Advanced Network Forensics and Analysis

NetFlow and File Access Protocols

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



This page intentionally left blank.

"As usual, SANS courses pay for themselves by Day 2. By Day 3, you are itching to get back to the office to use what you've learned."
Ken Evans, Hewlett Packard Enterprise - Digital Investigation Services

SANS Programs
sans.org/programs

GIAC Certifications
Graduate Degree Programs
NetWars & CyberCity Ranges
Cyber Guardian
Security Awareness Training
CyberTalent Management
Group/Enterprise Purchase Arrangements
DoDD 8140
Community of Interest for NetSec
Cybersecurity Innovation Awards

Search SANSInstitute

SANS Free Resources
sans.org/security-resources

- E-Newsletters
  NewsBites: Bi-weekly digest of top news
  OUCH!: Monthly security awareness newsletter
  @RISK: Weekly summary of threats & mitigations
- Internet Storm Center
- CIS Critical Security Controls
- Blogs
- Security Posters
- Webcasts
- InfoSec Reading Room
- Top 25 Software Errors
- Security Policies
- Intrusion Detection FAQ
- Tip of the Day
- 20 Coolest Careers
- Security Glossary

SANS Institute
8120 Woodmont Avenue | Suite 310
Bethesda, MD 20814
301.654.SANS(7267)
info@sans.org