

572.3

NetFlow and File Access Protocols

<https://t.me/learningnets>

SANS

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

Copyright © 2019, Lewes Technology Consulting, LLC. All rights reserved to Lewes Technology Consulting, LLC and/or SANS Institute.

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

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

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

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

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

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

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

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

PMP and PMBOK are registered marks of PMI.

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

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

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



NetFlow and File Access Protocols



©2019 Lewes Technology Consulting, LLC | All Rights Reserved | Version # FOR572_E01_02

Authors:

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

<http://twitter.com/sansforensics>

SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

FOR500 Windows Forensics
GCFE

FOR508 Advanced Incident Response and Threat Hunting
GCFA

FOR518 Mac and iOS Forensic Analysis and Incident Response
SANS FORENSIC MAC FORENSIC ANALYSIS

FOR526 Advanced Memory Forensics & Threat Detection
SANS FORENSIC MEMORY FORENSICS

FOR585 Smartphone Forensic Analysis In-Depth
GASF

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

FOR578 Cyber Threat Intelligence
GCTI

FOR610 REM: Malware Analysis
GREM

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

OPERATING SYSTEM & DEVICE IN-DEPTH

INCIDENT RESPONSE & THREAT HUNTING

REM MASTER

504

[@sansforensics](#) [sansforensics](#) [dfir.to/DFIRCast](#) [dfir.to/gplus-sansforensics](#) [dfir.to/MAIL-LIST](#)

This page intentionally left blank.

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

OPERATING
SYSTEM &
DEVICE
IN-DEPTH

INCIDENT
RESPONSE
& THREAT
HUNTING



FOR500
Windows Forensics
GCFE



FOR518
Mac and iOS
Forensic Analysis and
Incident Response



FOR526
Advanced
Memory Forensics
& Threat Detection



FOR585
Smartphone Forensic
Analysis In-Depth
GASF



FOR508
Advanced Incident Response
and Threat Hunting
GCFA



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



FOR578
Cyber Threat Intelligence
GCTI



FOR610
REM: Malware Analysis
GREM



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



@sansforensics



sansforensics



dfir.to/DFIRCast



dfir.to/gplus-sansforensics



dfir.to/MAIL-LIST

TABLE OF CONTENTS	PAGE
NetFlow Collection and Analysis	5
Open-Source Flow Tools	29
Lab 3.1: Visual NetFlow Analysis with SOF-ELK®	58
File Transfer Protocol (FTP)	61
Lab 3.2: Tracking Lateral Movement with NetFlow	83
Microsoft Protocols	86
Lab 3.3: SMB Session Analysis & Reconstruction	119

SANS DFIR FOR572.3 | Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response 4

This page intentionally left blank.

NetFlow Collection and Analysis

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
- Powerful hardware and storage are 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 several 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 data sets is nearly impossible without 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 a statistical summary of traffic observed at a point in the environment which shares common attributes. These common attributes allow the packets to be grouped into a “Flow” record, which can then be examined at a later time. Because we understand how sessions and connections are established, we can form a 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/1023) to the server’s web service port (generally TCP/443). During the establishment of this session, a NetFlow sensor is monitoring a link on which the connection is made. The sensor notes that a new flow event has started. The sensor sees the source and destination IP addresses, Layer 4 protocol, and source and destination ports in the initial packet. The sensor then observes the source and destination systems exchange packets with the same source and destination ports—which will not change without a new three-way handshake. Therefore, the sensor can note the following facts regarding the session:

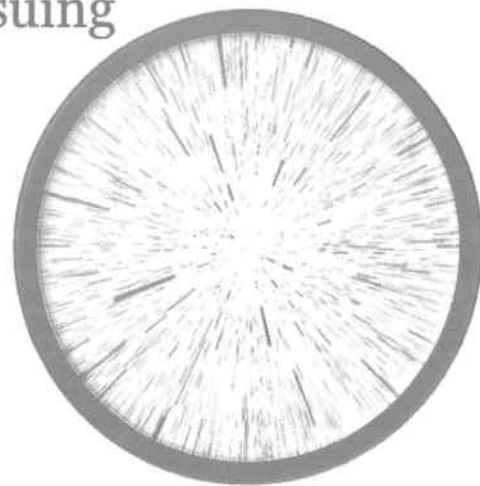
- At $\text{Time}_{\text{Open}}$ system A connected to system B
- The port used by system A was Port_A and the port used by system B was Port_B
- A volume of Data_{A-B} was passed from system A to system B
- A volume of Data_{B-A} was passed from system B to system A
- The connection closed at $\text{Time}_{\text{Closed}}$

The summary of this communication event can be framed into a NetFlow record.

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: Threat Hunting, Analysis, and Incident Response

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 4 and below rather than allocating more resources for full-packet capture or Layer 7 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 TLS 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 Origins and Derivations



- 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 J-Flow.
- 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]. The IETF finalized IPFIX (aka NetFlow v10) via RFC 7011^[2].

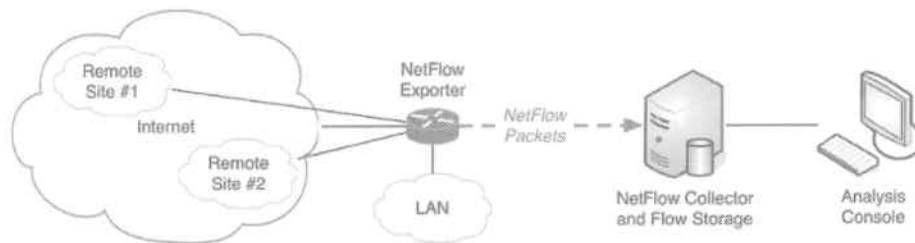
Most modern routing hardware and server OSes support a variety of these standards. Cisco generally supports versions 5 and 9, and most non-Cisco equipment supports versions 5 and 9, as well as IPFIX. VMware ESX can natively export IPFIX. Hosted platforms provide their own equivalent flow data, such as Amazon EC2's VPC-flow^[3] and Azure's Network Watcher^[4].

References:

- [1] <http://for572.com/hz1dw>
- [2] <http://for572.com/gyxsw>
- [3] <http://for572.com/t-e0r>
- [4] <http://for572.com/l0wfd>



- 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:

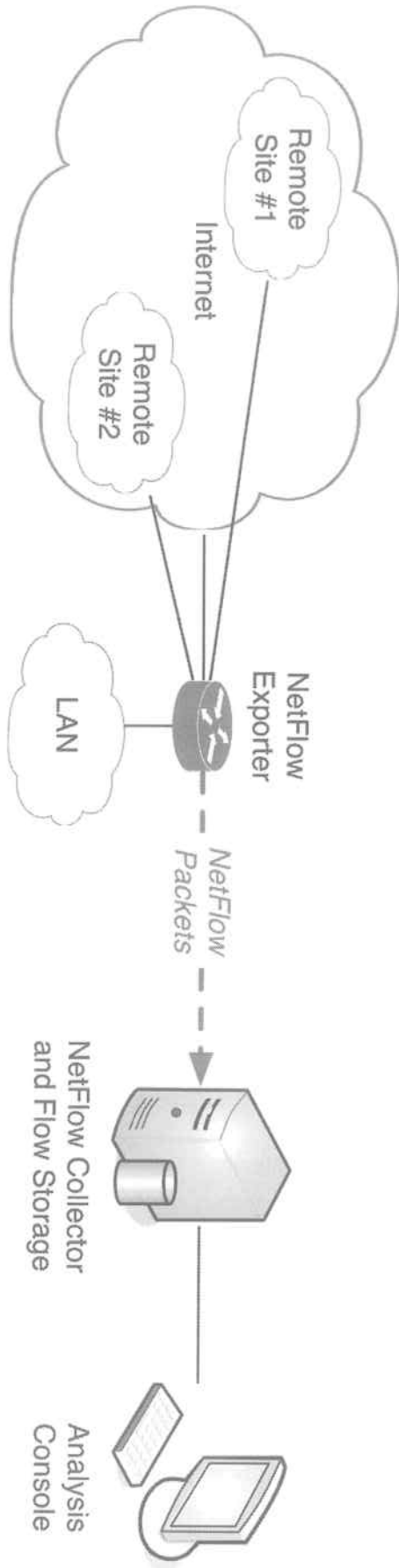
- **Exporter** – The device capable of generating NetFlow records, then exporting them to a Collector.
- **Collector** – The device that receives the exported flows from one or more Exporters.
- **Storage** – Where the exported flows are stored, in a format that is indexed and ready for processing.
- **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.



5



- 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	Mask	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.

<u>Bytes</u>	<u>Contents</u>	<u>Description</u>
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 3 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.



- 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 than 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 worth of data can be exported in the same packet by including multiple flowset IDs, each at the offset defined by the current flowset length value.)

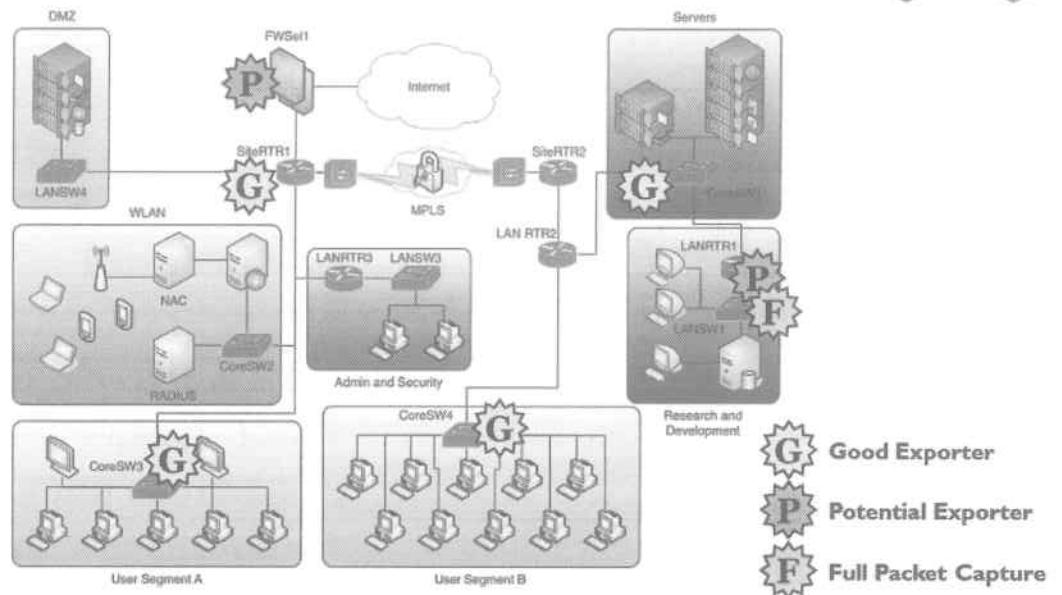
The record fields in each data flowset reflect the values defined in the corresponding NetFlow template. Because each field's type and length are advertised in the template, the collector can "walk" the data per those instructions.

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>

Identify Choke and Critical Points



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 the 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 and 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 provide 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 and 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 desktop systems of the administrators that control that data. 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 so it must be important” or “that’s a file server so it must be important” and is then thinking about what is actually important in terms of protection. However, critical data centers of gravity include different things, depending on the role and industry of the organization. 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.

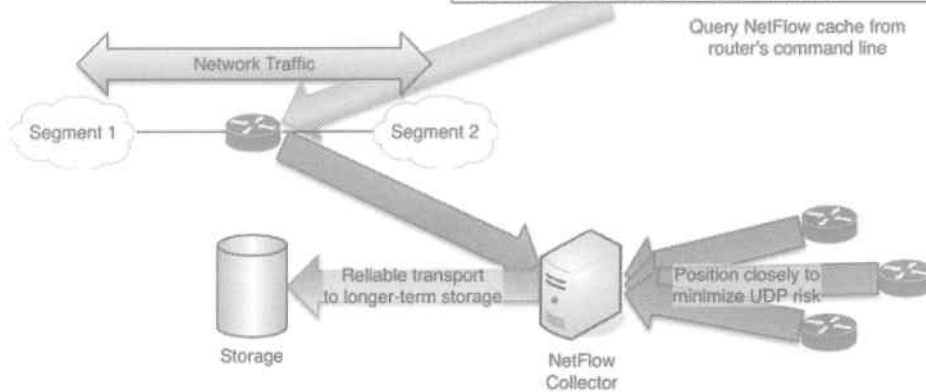
8: Confirm legal compliance

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.

Exporter Basics and Positioning



```
## show ip cache flow
IP packet size distribution (469 total packets):
1-32 64 96 128 160 192 224 256 288 320 352 384 416 448 480
.000 .968 .000 .031 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000
512 544 576 1024 1536 2048 2560 3072 3584 4096 4608
.000 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000
IP Flow Switching Cache, 378544 bytes
7 active, 4089 inactive, 261 added.....
```



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 is sent to the appropriate collector(s). This uses UDP transport by default, so it's easy to see the risk associated with the 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 are 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>

Analyst's Console



- Generally, a thin client—browser-based
- Server usually is co-located with storage to reduce lag and network impact
- Concurrent users limited by performance and license on system
- Highest spec needs of all NetFlow components

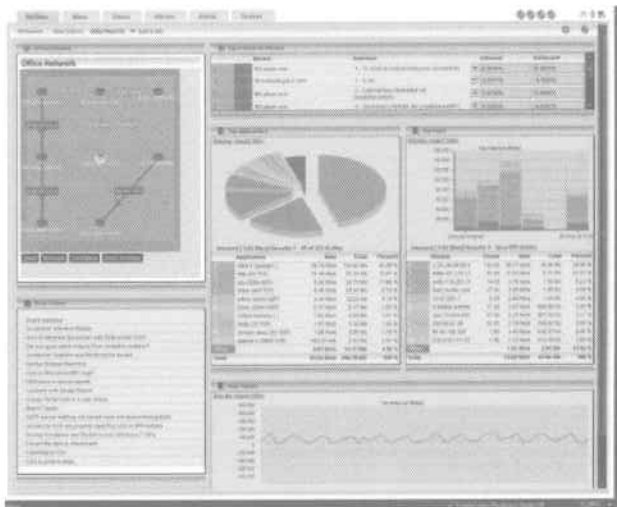
The analysis consoles are usually browser-based or a similar “app” that’s often just a re-skinned or framed browser with some authentication and branding on top.

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 a 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.

Storage Options and Formats Vary Between Collectors



- Sized to suit network
- Security, network teams have different needs!
 - Network team = 1mo?
 - Security team = forever!
- Various storage formats
 - Database
 - Binary files
 - ASCII files

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 (2018),^[1] which stated that the median time between a compromise and its detection was over 100 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.

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.^[2]

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 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.

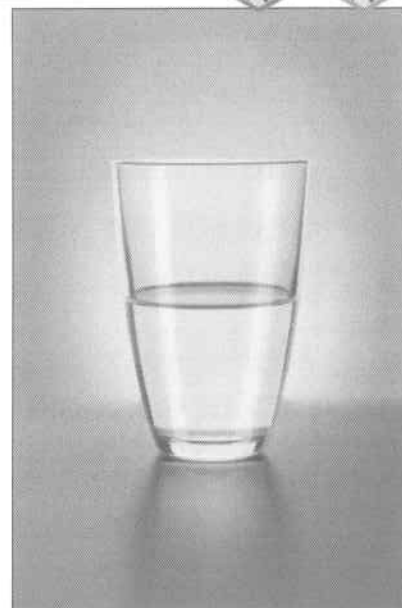
With file-based storage, accessibility is relatively easy—regardless of the format. Therefore, for some organizations, the decision of what tool to implement is more driven by the user's requirements to further mine the NetFlow data using other tools rather than just the initial collection and presentation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

References:

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

[2] <http://for572.com/h6ldq>

- pcap is a challenge due to volume...
...limited view from NetFlow is as well
- 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 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: 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.

Multiple-Use Protocols

- **SSH**
 - Used for remote management of *NIX-based systems
 - File transfer via `scp` or `sftp` utilities
 - Can set up port-based or SOCKS tunnel
- **TLS**
 - Typically used to secure payload of historically plaintext protocols such as HTTP, SMTP, etc.
 - Can establish a VPN between endpoints via SSTP
- **NetFlow** tells volume, duration, throughput, etc.

A number of other protocols are commonly used for encrypting network traffic, including both TLS and SSH. Similar to IPsec, TLS 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 that can proxy all TCP connections to an arbitrary IP address. If analyzing SSH-based activity from NetFlow, the difference between these use cases may be apparent only from the average throughput and duration of the connection.

TLS is commonly used by attackers for custom backdoors since TLS encryption is a well-documented standard and fairly easy for even an entry-level criminal software developer to implement. Although TLS 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.

In any of these cases, NetFlow can still be helpful in characterizing the connections based on their observed behaviors. An SSH session running a command shell will generally be "low and slow", while a file transfer will be an immediate ramp-up and plateau of throughput. Normal HTTPS traffic is typically not going to use an hours-long connection, but an SSTP connection could—giving the analyst a way to possibly differentiate the two connections, even if they both used TCP/443 for their transport.

References:

[1] <http://for572.com/9f8n6>

NetFlow and Encrypted Traffic Provide Similar Challenges



- Malicious activity can blend in with benign traffic
- Requires knowledge of other artifacts

	Apparent Protocol	Client Volume	Server Volume	Connection Duration	Potential Activity
Is DNS name suspicious, malicious, newly-observed?	TCP/80: HTTP	Small	Large	Short	Typical HTTP download
Is source or destination IP malicious?	TCP/443: HTTPS	Large	Small	Various	HTTP POST upload (Exfil?)
	TCP/443: HTTPS	Large	Large	Long	TLS VPN
Is TLS certificate or Certificate Authority compromised?	TCP/22: SSH	Small	Large	Various	SCP download
	TCP/22: SSH	Various	Various	Long	Command shell
	TCP/53: DNS	Large	Large	Long	Tunnel? Definitely suspicious!

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.

When analyzing any traffic without content (NetFlow 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 NetFlow typically 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, TLS tunnels typically do not send as much synchronization information. Therefore, a significant amount of TLS 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.

Open-Source Flow Tools

This page intentionally left blank.



- Collection of tools developed by the CERT Network Situational Awareness Team
- Supports NetFlow v5 and now NetFlow v9 natively
 - Still unidirectional due to legacy architecture
- Designed for **big** systems:
 - Backbones
 - Borders of large and distributed enterprises
 - Mid-sized Internet Service Providers
- Command-line based

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:

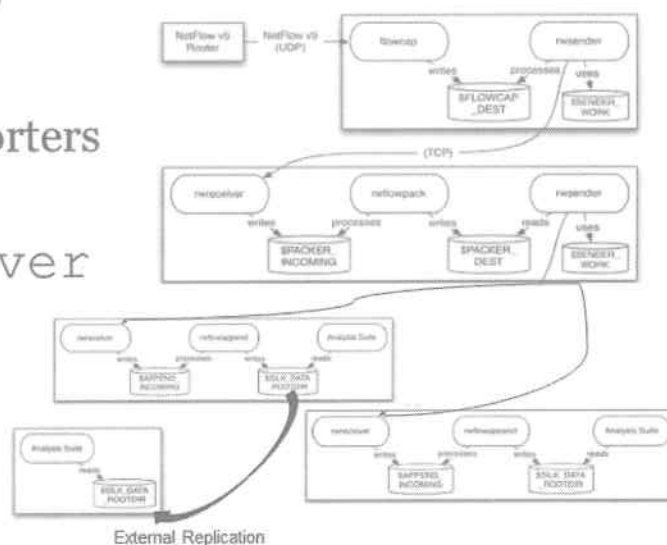
[1] <http://for572.com/vgk01>

[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.

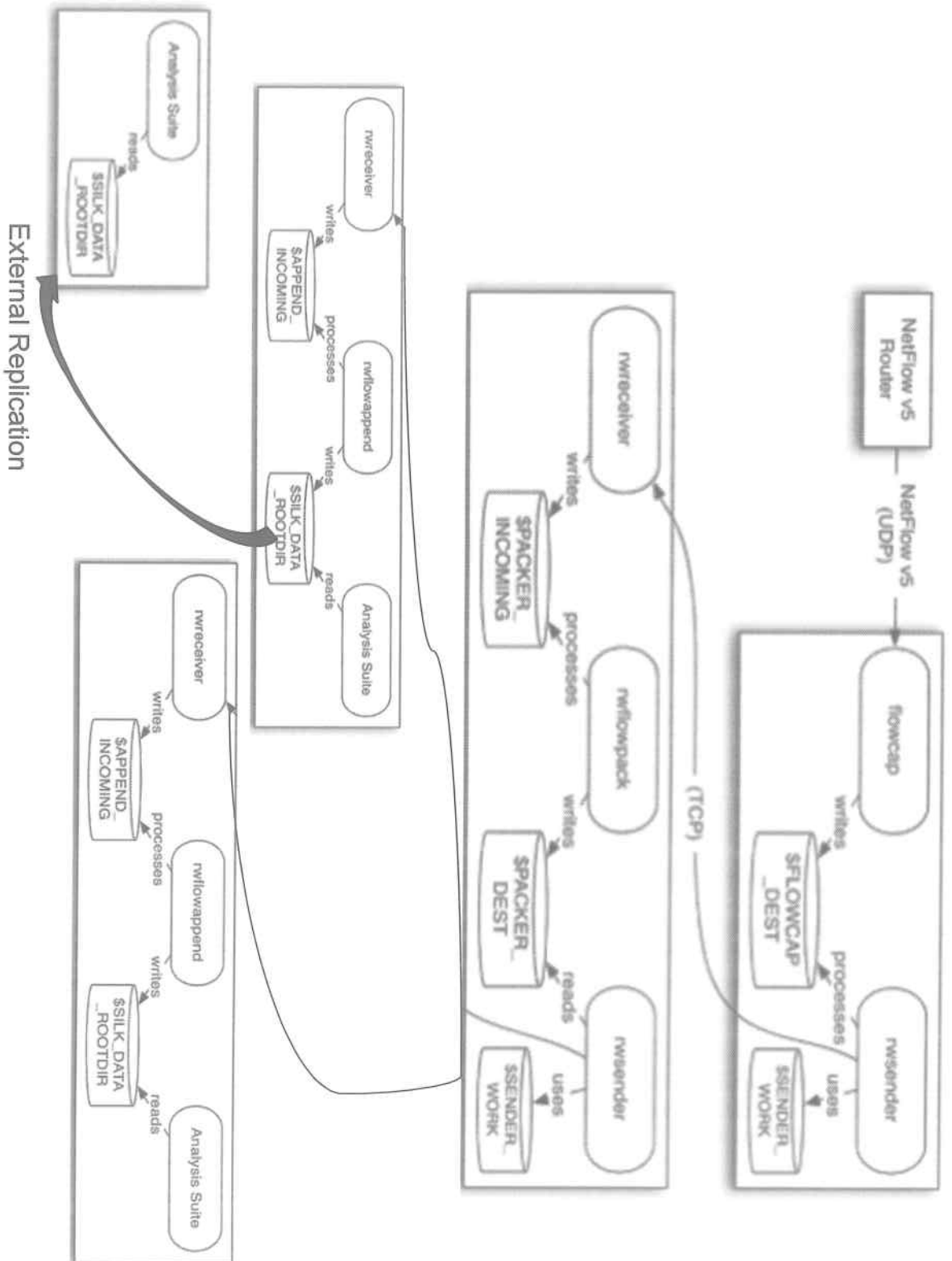


One excellent aspect of SiLK is that it transmits data from collectors to other nodes within the architecture. This mitigates the traditional problems associated with using the unreliable UDP transport mechanism. Although SiLK still captures data via UDP from the source exporter (using `flowcap`), it 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 filesystem 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 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
 - 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 filesystem 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 five 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 example below, 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.

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



- 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 `nfpicapd`, 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



- Read from files or directories of nfcapd data
- Read from single file

```
$ nfdump -r /var/netflow/site-rtr-1/2018/11/08/nfcapd.201811080635 ...
```

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

```
$ nfdump -R /var/netflow/site-rtr-1/2018/11/ ...
```

This slide shows two of the input options that nfdump provides. The tool can read from a single file or from a directory full of files. The latter mode is helpful because nfdump recursively navigates 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, 2018, from 06:35 to 06:40 (collector local time).

```
$ nfdump -r /var/netflow/site-rtr-1/2018/11/08/nfcapd.201811080635 ...
```

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” year/month/day 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 2018.

```
$ nfdump -R /var/netflow/site-rtr-1/2018/11/ ...
```



- Numerous filters available:
 - Protocol: `proto (tcp | udp | icmp | gre | ah | 132)`
 - IP addresses: `host 1.2.3.4 or net 1.2.3.0/24`
 - Autonomous systems: `as 31835`
 - Port: `port 443`
 - TCP flags: `flags S and not flags AFRPU`
 - Session length (ms): `duration < 5000`
 - Volume (bytes): `bytes > 1000000`
- Directionality: `src port, dst host, dst net`
- Standard logic to combine: `and, or, not, ()`

The filters that `nfdump` provides are 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. Of course, directly-observed fields such as Layer 3 or 4 protocols, IP addresses (on a per-IP or CIDR block basis), and Layer 4 ports are available. TCP flag queries can be issued, but the syntax for these will match any flow records where the specified flag is set – so creating a query for a single exclusive flag requires ruling out all of the undesired flags as shown above. Ancillary fields such as BGP autonomous system numbers (ASNs) could be available if the exporter sends them. However, these values will be zero if the exporter is not ASN-aware or if the IP address is an RFC 1918 private IP address. Such values can also be added later via an enrichment process outside of the exporter/collector/query tool chain.

Several derived values are also available as query parameters. These include the length of a session, which is simply calculated at query time as the delta between the start and end time of the flow record. Volume can be another useful parameter when searching for very large or very small flows. Remember that the recorded byte count is the sum total of bytes observed during the flow event.

Address filters at Layers 3 and 4 and ASNs can be directionally qualified with 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`”, as well as using parentheses to enforce the order of filter operations, analysts 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 172.16.4.10) between noon (`nfcapd` local time!) on July 12, 2018 and noon on July 13, 2018.

```
$ nfdump -R /var/local/flows/router1/2018/07/ -t '2018/07/12.12:00:00-2018/07/13.12:00:00' -c 10 'proto tcp and (dst port 80 or dst port 443) and not src host 172.16.4.10'
```



- “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
2018-08-30 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
2018-08-30 06:53:53.370	63.545	TCP	113.138.32.152:25	-> 222.33.70.124:3575	.AP.SF	0	62	3512	1
2018-08-30 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
2018-08-30 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	
2018-08-30 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	

If an output format is not specified with the “-o” command-line parameter, nfdump attempts to display data based on the type of query issued. These examples show the three predefined formats.

The “line” output format provides several basic pieces of information for each flow event that matches any supplied filter.

Specifying the “long” output format adds TCP flags and Type of Service values. The flags represent the union of any flag seen at least once during the flow. The TCP example shown also highlights that each line of output includes a single unidirectional NetFlow event—so a bidirectional communication channel like this one is reflected by two flow records.

The “extended” format adds derived values for the packets per second, bits per second, and bytes per packet values. These are not stored as part of the flow record, but are derived at query time by simply dividing the appropriate stored values. Such statistical values can be useful because they reflect the data rate for the connection and may suggest a bulk download or mostly-dormant command shell, for example.

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



- 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 top 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
2018-03-12 23:58:59.386	2018-03-12 23:59:00.933	1.547	205.186.148.46	157.55.39.191	35449
2018-03-12 23:58:59.599	2018-03-12 23:59:00.322	0.723	205.186.148.46	72.78.204.217	50071
2018-03-12 23:59:00.116	2018-03-12 23:59:00.995	0.879	127.0.0.1	127.0.0.1	59783
2018-03-12 23:59:00.813	2018-03-12 23:59:01.220	0.407	127.0.0.1	127.0.0.1	53414
2018-03-12 23:59:01.851	2018-03-12 23:59:01.851	0.000	205.186.148.46	70.32.65.152	53
2018-03-12 23:59:02.351	2018-03-12 23:59:02.351	0.000	70.32.65.152	205.186.148.46	56879

Fortunately, nfdump also permits the use of custom output formats to tailor results to the analyst's requirements. This functionality involves the use of numerous format strings, which are fully documented on the man page.

These format strings allow the analyst to craft specific results that can be fed into any number of downstream processes. For example, shell scripts and text-parsing utilities could be used to locate unique IP addresses or conversations. Custom CSV output could be fed into spreadsheet or visualization tools such as Microsoft Excel or Paterva's Maltego. As you'll see in this section, the SOF-ELK® ingest process for archived NetFlow uses an extensive custom format string built into a shell script when reading data created by nfcapd or nfpcapd.

```
$ 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
2018-08-30 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
2018-08-30 06:53:53.370      63.545 TCP      113.138.32.152:25 -> 222.33.70.124:3575 .AP.SF 0      62 3512 1
2018-08-30 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 Bpp
Flows
2018-08-30 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
2018-08-30 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 -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
2018-03-12 23:58:59.386 2018-03-12 23:59:00.933 1.547 205.186.148.46 157.55.39.191 35449
2018-03-12 23:58:59.599 2018-03-12 23:59:00.322 0.723 205.186.148.46 72.78.204.217 50071
2018-03-12 23:59:00.116 2018-03-12 23:59:00.995 0.879 127.0.0.1 127.0.0.1 59783
2018-03-12 23:59:00.813 2018-03-12 23:59:01.220 0.407 127.0.0.1 127.0.0.1 53414
2018-03-12 23:59:01.851 2018-03-12 23:59:01.851 0.000 205.186.148.46 70.32.65.152 53
2018-03-12 23:59:02.351 2018-03-12 23:59:02.351 0.000 70.32.65.152 205.186.148.46 56879
```

nfdump: Example Scenario (I)



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

```
$ nfdump -r /var/local/flows/nfcapd.201803120000 -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
2018-03-12 00:09:01.356  0.000 UDP    10.3.58.6:45524  -> 174.37.46.52:8765    1      78    1
2018-03-12 00:23:01.356  0.000 UDP    10.3.59.76:33597  -> 174.37.46.52:8765    1      76    1
2018-03-12 01:18:01.746  0.000 UDP    10.3.58.85:57002  -> 174.37.46.52:8765    1      78    1
2018-03-12 01:36:01.851  0.000 UDP    10.3.58.7:38306   -> 174.37.46.52:8765    1      53    1
2018-03-12 02:12:02.256  0.000 UDP    10.3.58.4:32833   -> 174.37.46.52:8765    1      53    1
```

- First one overnight (1800-0900 UTC):

```
$ nfdump -R /var/local/flows/ -t '2018/03/11.18:00:00-2018/03/12.09:00:00' -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
2018-03-11 22:04:49.411  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 port UDP/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 NetFlow data set starting March 12, 2018 at 00:00 collector local time, which is UTC, as it should be. (This time frame is suggested by the filename and confirmed by examining the file’s contents). In this case, the file is a combined 24-hour chunk of NetFlow records. He asks for the first five hosts that sent packets on the suspected protocol and port.

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 the entire overnight period from 1800 to 0900 the next morning.

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 '2018/03/11.18:00:00-2018/03/12.09:00:00' -O tstart 'host 10.3.58.6'
Date first seen      Duration Proto  Src IP Addr:Port  Dst IP Addr:Port  Packets  Bytes  Flows
2018-03-11 22:00:16.532 24.581 TCP    10.3.58.6:10581  -> 174.37.46.94:80   81      4281   1
2018-03-11 22:00:16.532 24.581 TCP    174.37.46.94:80  -> 10.3.58.6:10581  547    812916 1
2018-03-11 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 '2018/03/01-2018/04/01' -O
-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 just a few minutes.

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 AS 36351, a particular hosting provider.

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 -r /var/local/flows/nfcapd.201803120000 -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
2018-03-12 00:09:01.356  0.000 UDP      10.3.58.6:45524  -> 174.37.46.52:8765  1       78     1
2018-03-12 00:23:01.356  0.000 UDP      10.3.59.76:33597  -> 174.37.46.52:8765  1       76     1
2018-03-12 01:18:01.746  0.000 UDP      10.3.58.85:57002  -> 174.37.46.52:8765  1       78     1
2018-03-12 01:36:01.851  0.000 UDP      10.3.58.7:38306   -> 174.37.46.52:8765  1       53     1
2018-03-12 02:12:02.256  0.000 UDP      10.3.58.4:32833   -> 174.37.46.52:8765  1       53     1

$ nfdump -R /var/local/flows/ -t '2018/03/11.18:00:00-2018/03/12.09:00:00' -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
2018-03-11 22:04:48.411  0.000 UDP      10.3.58.6:11574   -> 174.37.46.52:8765  1       750    1

$ nfdump -R /var/local/flows/ -t '2018/03/11.18:00:00-2018/03/12.09:00:00' -O tstart 'host 10.3.58.6'
Date first seen      Duration Proto      Src IP Addr:Port  Dst IP Addr:Port  Packets  Bytes  Flows
2018-03-11 22:00:16.532  24.581 TCP      10.3.58.6:10581   -> 174.37.46.94:80    81      4281   1
2018-03-11 22:00:16.532  24.581 TCP      174.37.46.94:80   -> 10.3.58.6:10581   547     812916 1
2018-03-11 22:04:48.411  0.000 UDP      10.3.58.6:11574   -> 174.37.46.52:8765  1       750    1
...

$ 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 '2018/03/01-2018/04/01' -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
...

```



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

```
$ nfdump -r nfcapd.201805300000 -a -O tstart
```

Date first seen	Duration	Proto	Src IP Addr:Port	Dst IP Addr:Port	Packets	Bytes	Flows
2018-05-29 23:54:41.647	342.994	TCP	205.186.148.167:443	-> 66.249.66.156:45326	411	537757	1
2018-05-29 23:54:41.647	343.004	TCP	66.249.66.156:45326	-> 205.186.148.167:443	248	20174	1
2018-05-29 23:54:48.069	86545.814	ICMP	198.46.150.48:0	-> 205.186.148.46:8.0	1433	131836	249
2018-05-29 23:54:48.069	86545.814	ICMP	205.186.148.46:0	-> 198.46.150.48:0.0	1433	131836	249
2018-05-29 23:56:12.119	2109.580	TCP	205.186.148.46:443	-> 72.219.212.172:51296	189	96711	6
2018-05-29 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/2018/11/ -O bytes -A srcip,dstport -o 'fmt:%sa -> %dp %byt %fl'
```

192.168.75.3 ->	443	45.8 G	1154628
50.202.156.2 ->	1194	39.8 G	483
192.168.75.232 ->	443	31.7 G	197051
10.8.0.6 ->	443	31.4 G	227330
192.168.75.32 ->	443	30.1 G	243868

Because of the five-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` can aggregate records based on specified traffic characteristics.

The default aggregation uses the five key flow values: source and destination IP addresses, Layer 4 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 first example shown reflects an aggregation from a single day of traffic. The first two lines show a single connection (one flow in each direction) 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 a discrete end state for ICMP traffic. The underlying activity here is a ping every five minutes for availability monitoring purposes. However, because `nfcapd` uses the “destination port” field to represent the ICMP type and code, these flow records are aggregated together. The last two records show a flow event approximately 30 minutes in duration with six flows in each direction. This was webmail activity, in which the HTTPS socket is kept open by periodic background requests.

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 using the parameter “-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 connection attempt counts destined for static, well-known port numbers (not just 22).

The second command recursively uses flow files from the “/var/local/flows/2018/11/” directory. It will aggregate all flows sharing the same source IP and destination port to a single output record with the specified custom output format. This particular query would be suitable for identifying top consumers per apparent service.

```

$ nfdump -r nfcapd.201805300000 -a -O tstart
Date first seen Duration Proto Src IP Addr:Port Dst IP Addr:Port Packets Bytes Flows
2018-05-29 23:54:41.647 342.994 TCP 205.186.148.167:443 -> 66.249.66.156:45326 411 537757 1
2018-05-29 23:54:41.647 343.004 TCP 66.249.66.156:45326 -> 205.186.148.167:443 248 20174 1
2018-05-29 23:54:48.069 86545.814 ICMP 198.46.150.48:0 -> 205.186.148.46:8.0 1433 131836 249
2018-05-29 23:54:48.069 86545.814 ICMP 205.186.148.46:0 -> 198.46.150.48:0.0 1433 131836 249
2018-05-29 23:56:12.119 2109.580 TCP 205.186.148.46:443 -> 72.219.212.172:51296 189 96711 6
2018-05-29 23:56:12.119 2109.580 TCP 72.219.212.172:51296 -> 205.186.148.46:443 144 61374 6

$ nfdump -q -R /var/local/flows/2018/11/ -O bytes -A srcip,dstport -o 'fmt:%sa -> %dp %byt %fl'
192.168.75.3 -> 443 45.8 G 1154628
50.202.156.2 -> 1194 39.8 G 483
192.168.75.232 -> 443 31.7 G 197051
10.8.0.6 -> 443 31.4 G 227330
192.168.75.32 -> 443 30.1 G 243868

```

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/2018/ -s ip/bytes -s dstport/bytes -n 5
Top 5 IP Addr ordered by bytes:
Date first seen   Duration   Proto   IP Addr   Flows(%)   Packets(%)   Bytes(%)   pps   bps   bpp
2017-12-31 23:57:18.118 29178981.168 any 192.168.75.40 1.9 M( 1.7) 824.8 M(11.6) 2.0 T(21.0) 28 561107 2481
2017-12-31 23:57:13.195 29178994.403 any 192.168.75.3 40.5 M(37.0) 772.8 M(10.8) 973.7 G(10.0) 26 266956 1259
2018-07-23 15:59:41.260 11582104.629 any 192.168.75.232 3.3 M( 3.0) 621.2 M( 8.7) 965.8 G( 9.9) 53 667071 1554
2017-12-31 23:59:00.016 29178661.448 any 192.168.75.32 1.8 M( 1.7) 1.2 G(16.2) 912.7 G( 9.4) 39 250232 788
2017-12-31 23:55:00.972 23508032.459 any 192.168.75.128 7.2 M( 6.6) 633.4 M( 8.9) 735.5 G( 7.6) 26 250305 1161

Top 5 Dst Port ordered by bytes:
Date first seen   Duration   Proto   Dst Port   Flows(%)   Packets(%)   Bytes(%)   pps   bps   bpp
2017-12-31 23:55:00.972 29179184.917 any 443 19.5 M(17.8) 1.8 G(25.1) 1.3 T(13.7) 61 364569 744
2018-01-02 00:53:24.505 28904854.822 any 1194 37399( 0.0) 545.5 M( 7.7) 311.0 G( 3.2) 18 86089 570
2017-12-31 23:58:20.739 29178965.543 any 80 8.4 M( 7.6) 778.2 M(10.9) 46.8 G( 0.5) 26 12839 60
2018-01-01 04:39:21.298 29134169.949 any 61368 1399( 0.0) 36.9 M( 0.5) 41.8 G( 0.4) 1 11465 1131
2018-01-01 01:04:08.595 29136442.952 any 40016 1382( 0.0) 26.5 M( 0.4) 34.1 G( 0.4) 0 9360 1284
```

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:

- Sent the most traffic: Indicates they may have been used as staging systems or be the source of data loss
- Connected to the most hosts (highest number of flows): May show systems running various scanners

nfdump's TopN statistics can be ordered by a number of fields by using the "`-s statistic[/orderby]`" command-line option. The results will include the top ten results by default, or tuned with the "`-n`" parameter.

The following is a subset of available "statistic" values for these reports:

```
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
```

Reports can be sorted with these "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/2018/ -s ip/bytes -s dstport/bytes -n 5
```

```
Top 5 IP Addr ordered by bytes:
```

Date first seen	Duration	Proto	IP Addr	Flows (%)	Packets (%)	Bytes (%)	pps	bps	bpp
2017-12-31 23:57:18.118	29178981.168	any	192.168.75.40	1.9 M(1.7)	824.8 M(11.6)	2.0 T(21.0)	28	561107	2481
2017-12-31 23:57:13.195	29178994.403	any	192.168.75.3	40.5 M(37.0)	772.8 M(10.8)	973.7 G(10.0)	26	266956	1259
2018-07-23 15:59:41.260	11582104.629	any	192.168.75.232	3.3 M(3.0)	621.2 M(8.7)	965.8 G(9.9)	53	667071	1554
2017-12-31 23:59:00.016	29178661.448	any	192.168.75.32	1.8 M(1.7)	1.2 G(16.2)	912.7 G(9.4)	39	250232	788
2017-12-31 23:55:00.972	23508032.459	any	192.168.75.128	7.2 M(6.6)	633.4 M(8.9)	735.5 G(7.6)	26	250305	1161

```
Top 5 Dst Port ordered by bytes:
```

Date first seen	Duration	Proto	Dst Port	Flows (%)	Packets (%)	Bytes (%)	pps	bps	bpp
2017-12-31 23:55:00.972	29179184.917	any	443	19.5 M(17.8)	1.8 G(25.1)	1.3 T(13.7)	61	364569	744
2018-01-02 00:53:24.505	28904854.822	any	1194	37399(0.0)	545.5 M(7.7)	311.0 G(3.2)	18	86089	570
2017-12-31 23:58:20.739	29178965.543	any	80	8.4 M(7.6)	778.2 M(10.9)	46.8 G(0.5)	26	12839	60
2018-01-01 04:39:21.298	29134169.949	any	61368	1399(0.0)	36.9 M(0.5)	41.8 G(0.4)	1	11465	1131
2018-01-01 01:04:08.595	29136442.952	any	40016	1382(0.0)	26.5 M(0.4)	34.1 G(0.4)	0	9360	1284

```
Summary: total flows: 109517634, total bytes: 9731095701586, total packets: 7127439344, avg bps: 2667955, avg pps: 244, avg bpp: 1365
```

```
Time window: 2017-12-31 23:55:00 - 2018-12-04 17:14:45
```

```
Total flows processed: 109517634, Blocks skipped: 0, Bytes read: 6189215920
```

```
Sys: 26.406s flows/second: 4147313.6 Wall: 36.448s flows/second: 3004701.0
```



- SOF-ELK is also a NetFlow analysis platform
 - Receives live NetFlow v5 via UDP
 - Loads archived NetFlow parsed to text
- 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 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 into data that has been loaded.



- Loading archived NetFlow:

- Copy nfcapd data to SOF-ELK, load with script:
nfdump2sof-elk.sh
- Optionally set exporter IP with “-e”
- Create output text file in /logstash/nfarch/

```
$ nfdump2sof-elk.sh -r /path/to/netflow/nfcapd.201803190000 -w /logstash/nfarch/nf_export.txt
$ nfdump2sof-elk.sh -r /path/to/netflow/directory/ -w /logstash/nfarch/nf_export2.txt
$ nfdump2sof-elk.sh -e 10.3.58.1 -r /path/to/netflow/nfcapd.201810200000 -w /logstash/nfarch/nf_export3.txt
```

- Live collector:

- Open firewall port UDP/9995 on VM
- Point exporters or traffic splitter(s) to SOF-ELK

Loading archived NetFlow has been streamlined with a shell script that is part of the SOF-ELK distribution and GitHub repository^[1]. This script parses nfcapd-compatible source data and formats it in the specific structure that SOF-ELK requires. Note that the script can take an optional “-e” parameter, which allows the user to specify the exporter IP address from which the NetFlow was generated.

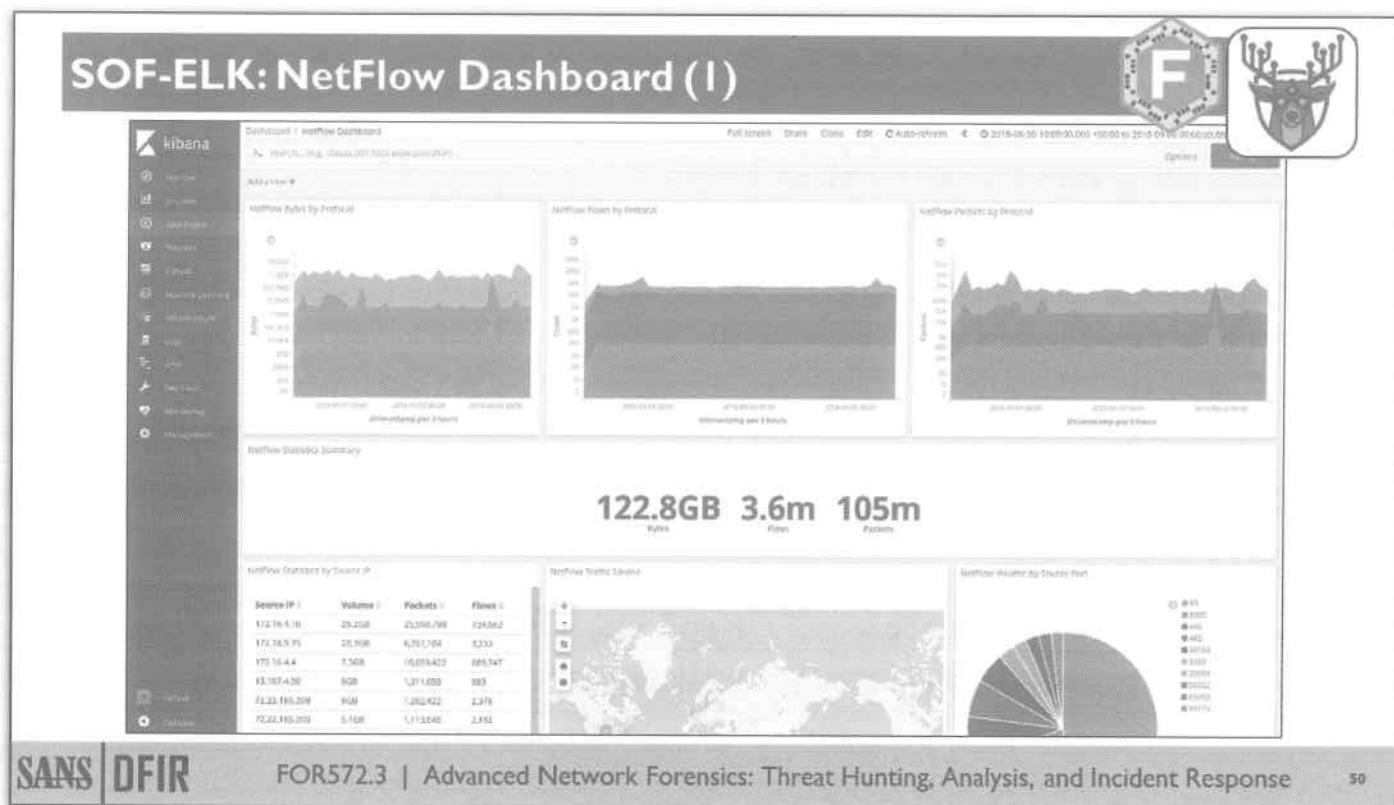
```
$ nfdump2sof-elk.sh -r /path/to/netflow/nfcapd.201803190000 ↵
-w /logstash/nfarch/nf_export.txt
$ nfdump2sof-elk.sh -r /path/to/netflow/directory/ ↵
-w /logstash/nfarch/nf_export2.txt
$ nfdump2sof-elk.sh -e 10.3.58.1 -r /path/to/netflow/nfcapd.201810200000 ↵
-w /logstash/nfarch/nf_export3.txt
```

Loading live NetFlow is 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-modify.sh -a open -p 9995 -r udp
```

References:

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



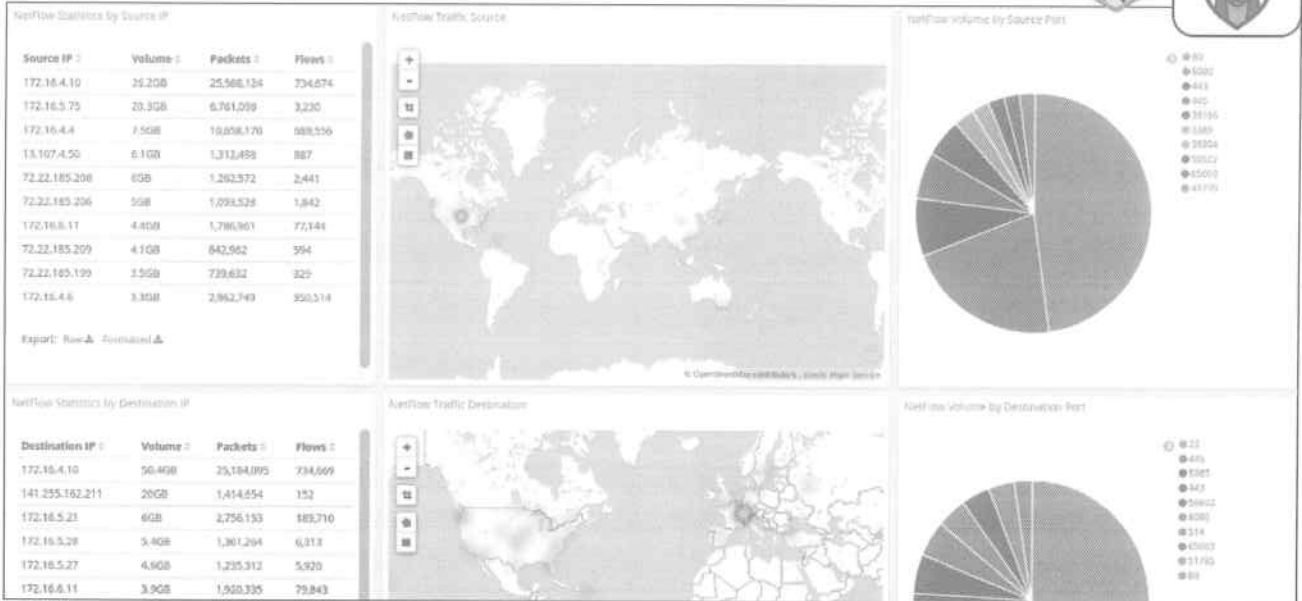
This is the top of SOF-ELK’s supplied NetFlow dashboard. Traffic is broken down by byte, packet, and flow count. This mirrors the basic metrics provided by `ncdump` and many other NetFlow platforms.

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, troughs, and plateaus of interest. Note that the graphs use logarithmic scale for their y-axis, preventing smaller-scale observations from being lost in large data sets. The counterpoint to this is that large spikes – such as the one at the very top and right side of the “NetFlow Bytes by Protocol” graph above – are less prominent. As with any tool, familiarity comes over time while using it.

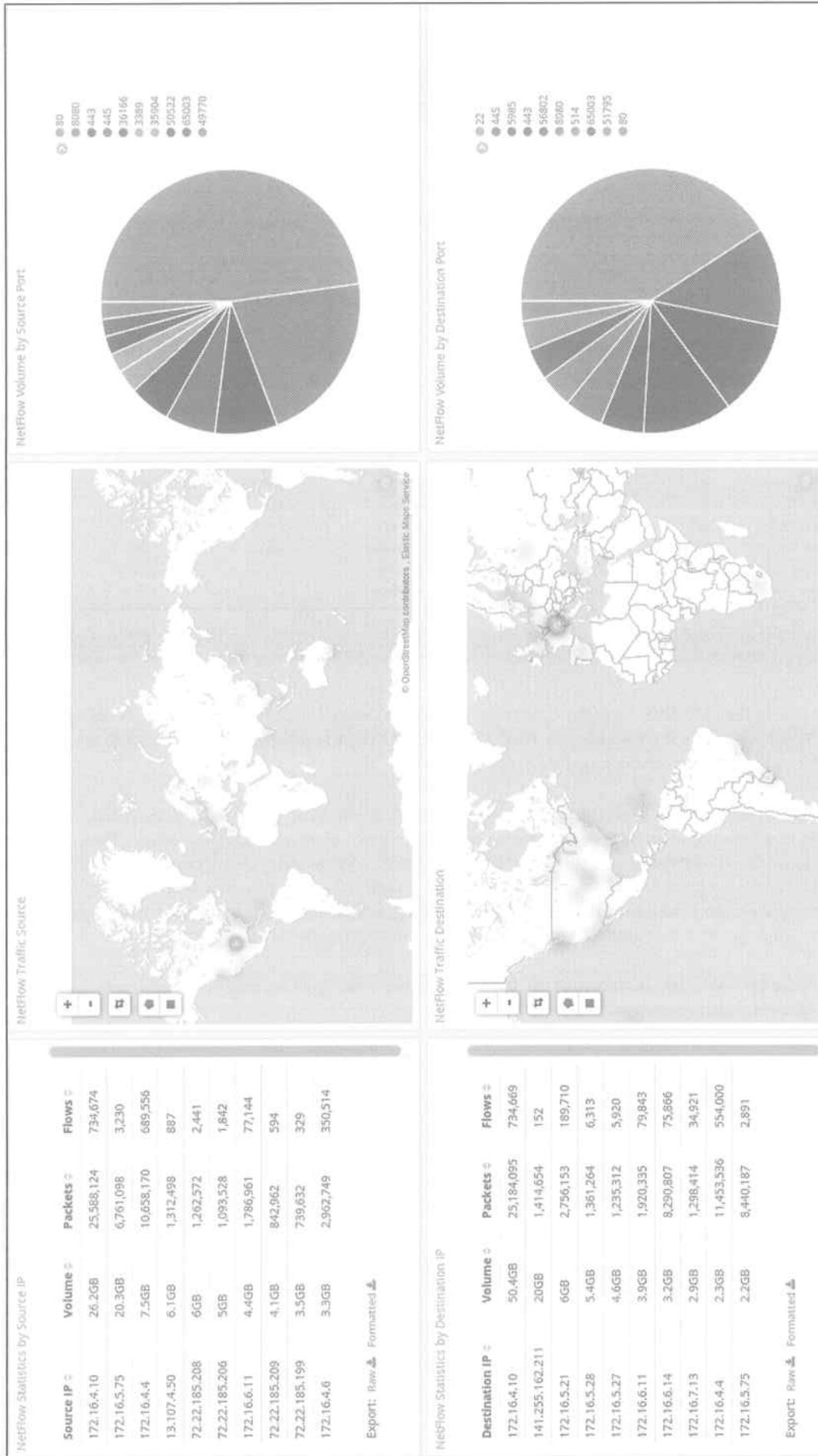
<https://t.me/learningnets>

SOF-ELK: NetFlow Dashboard (2)



Scrolling down the dashboard, the analyst can see summaries of the source and destination of NetFlow records currently in scope. The table at the left of each row reflects traffic by 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. (This enrichment is performed locally, with no network queries.) The pie graphs on the left show the traffic's 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.



SOF-ELK: NetFlow Dashboard (3)



NetFlow Statistics by Exporter

Exporter	Volume	Packets	Flows
172.16.21.1	122.80B	104,357,228	3,504,254

Export: raw & Formatted &

NetFlow Statistics by Source AS

Source AS	Volume	Packets	Flows
null (AS0)	74.0GB	86,102,388	3,129,876
Lightower Fiber Networks I, LLC (AS46887)	29.9GB	8,323,976	8,363
Microsoft Corporation (AS8068)	6.2GB	1,532,624	15,114
Microsoft Corporation (AS8073)	2.5GB	2,820,826	246,333
Level 3 Parent, LLC (AS3256)	2.2GB	492,520	843
Highwinds Network Group, Inc. (AS26448)	1.9GB	701,890	1,224
MCJ Communications Services, Inc. G/B/A Verizon Business (AS15133)	974.1MB	387,034	775

Export: raw & Formatted &

NetFlow Statistics by Destination AS

Destination AS	Volume	Packets	Flows
null (AS0)	96.2GB	85,233,502	2,005,885
Private Layer INC (AS11852)	20GB	1,414,654	132
GIT Communications Inc. (AS3257)	3.8GB	188,582	248
Microsoft Corporation (AS8073)	1.9GB	6,805,560	612,419
DigitalOcean, LLC (AS14061)	790.5MB	1,198,534	98,422
Lightower Fiber Network I, LLC (AS46887)	264.6MB	5,316,482	8,494
Microsoft Corporation (AS8068)	134.1MB	1,238,348	15,850

Export: raw & Formatted &

NetFlow Discovery

Time	exporter	protocol	source_ip	source_port	destination_ip	destination_port	total_bytes	total_packets
2018-09-05 23:54:05.879Z	172.16.21.1	TCP	172.16.5.21	50334	172.16.4.10	8080	4892	18
2018-09-05 23:54:05.879Z	172.16.21.1	HTTP	172.16.4.10	8080	172.16.5.21	50334	14,036	22
2018-09-05 23:54:08.317Z	172.16.21.1	TCP	172.16.4.4	88	172.16.5.25	80804	406	0
2018-09-05 23:54:38.917Z	172.16.21.1	UDP	172.16.5.25	80806	172.16.4.4	88	820	9
2018-09-05 23:54:38.918Z	172.16.21.1	HTTP	172.16.4.4	445	172.16.5.25	80805	1,632	9
2018-09-05 23:54:38.315Z	172.16.21.1	HTTP	172.16.5.25	80803	172.16.4.4	445	2,590	12
2018-09-05 23:54:38.367Z	172.16.21.1	UDP	172.16.5.25	80804	172.16.4.4	88	820	0

The next row in the SOF-ELK NetFlow dashboard includes additional 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 a larger 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 document listing as we saw previously with the SOF-ELK dashboard for logfile aggregation.

NetFlow Statistics by Exporter				NetFlow Statistics by Source AS				NetFlow Statistics by Destination AS			
Exporter	Volume	Packets	Flows	Source AS	Volume	Packets	Flows	Destination AS	Volume	Packets	Flows
172.16.21.1	122.8GB	104,957,228	3,584,254	null (AS0)	74.6GB	86,102,386	3,129,876	null (AS0)	96.2GB	85,233,502	2,605,865
				Lighttower Fiber Networks I, LLC (AS46887)	29.9GB	6,323,976	8,363	Private Layer INC (AS51852)	20GB	1,414,654	152
				Microsoft Corporation (AS8075)	6.2GB	1,553,624	15,114	GTT Communications Inc. (AS3257)	3.8GB	188,582	248
				Microsoft Corporation (AS8075)	2.5GB	2,830,836	246,393	Microsoft Corporation (AS8075)	1.3GB	6,895,540	612,419
				Level 3 Parent, LLC (AS3356)	2.2GB	492,928	843	DigitalOcean, LLC (AS14061)	790.5MB	1,188,534	98,422
				Highwinds Network Group, Inc. (AS20446)	1.9GB	701,690	1,224	Lighttower Fiber Networks I, LLC (AS46887)	284.6MB	5,316,482	8,484
				MCI Communications Services, Inc. d/b/a Verizon Business (AS15133)	974.1MB	387,834	775	Microsoft Corporation (AS8068)	134.1MB	1,238,348	15,850
Export: Raw Formatted				Export: Raw Formatted				Export: Raw Formatted			

NetFlow Discovery											
Time	exporter	protocol	source_ip	source_port	destination_ip	destination_port	total_bytes	total_packets			
2018-09-05 23:54:45.870Z	172.16.21.1	tcp	172.16.5.27	50334	172.16.4.10	8080	4,882	18			
2018-09-05 23:54:45.870Z	172.16.21.1	tcp	172.16.4.10	8080	172.16.5.27	50334	13,036	22			
2018-09-05 23:54:38.317Z	172.16.21.1	tcp	172.16.4.4	88	172.16.5.25	60386	606	8			
2018-09-05 23:54:38.317Z	172.16.21.1	tcp	172.16.5.25	60386	172.16.4.4	88	820	8			
2018-09-05 23:54:38.315Z	172.16.21.1	tcp	172.16.4.4	445	172.16.5.25	60385	1,832	8			
2018-09-05 23:54:38.315Z	172.16.21.1	tcp	172.16.5.25	60385	172.16.4.4	445	2,350	12			
2018-09-05 23:54:38.307Z	172.16.21.1	tcp	172.16.5.25	60384	172.16.4.4	88	820	8			
2018-09-05 23:54:38.307Z	172.16.21.1	tcp	172.16.4.4	88	172.16.5.25	60384	606	8			
2018-09-05 23:54:38.304Z	172.16.21.1	tcp	172.16.4.4	445	172.16.5.25	60383	1,832	8			
2018-09-05 23:54:38.304Z	172.16.21.1	tcp	172.16.5.25	60383	172.16.4.4	445	2,350	12			

SOF-ELK: NetFlow Dashboard (4)



NetFlow Discovery

Time	exporter	protocol	source_ip	source_port	destination_ip	destination_port	total_bytes	total_packets
2018-09-05 23:54:05.873Z	172.16.21.1	tcp	172.16.5.27	50334	172.16.4.10	8080	4,942	18
2018-09-05 23:54:05.873Z	172.16.21.1	udp	172.16.4.10	8080	172.16.5.27	50334	13,036	23
2018-09-05 23:54:38.817Z	172.16.21.1	tcp	172.16.4.4	80				
2018-09-05 23:54:38.817Z	172.16.21.1	tcp	172.16.5.25	50385				
2018-09-05 23:54:38.817Z	172.16.21.1	tcp	172.16.5.25	50385				
2018-09-05 23:54:38.267Z	172.16.21.1	tcp	172.16.5.25	60384				

- # destination_port 45236
- # engine_id 0
- # engine_type 0
- exporter 172.16.21.1
- first_switched 2018-09-05 23:12:30.503Z
- # flow_records 1
- # flow_records_vis 1
- t host.name sof-elk
- # in_bytes 4,995,286
- # in_bytes_vis 4.8MB
- # in_packets 1,382
- # in_packets_vis 1.4k

These records are displayed in a spreadsheet-like interface, but each row can be expanded to show all fields present in the record. To expand each record, simply click the triangle icon to the far left in the row of interest.

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. The asterisk icon will create a filter for any record where the field exists – essentially a wildcard match.

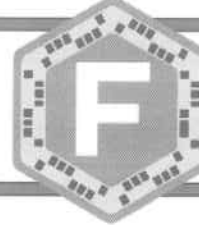
NetFlow Discovery

1-50 of 3,584,254 < >

Time	exporter	protocol	source_ip	source_port	destination_ip	destination_port	total_bytes	total_packets
2018-09-05 23:54:45.879Z	172.16.21.1	tcp	172.16.5.27	50334	172.16.4.10	8080	4,882	18
2018-09-05 23:54:45.879Z	172.16.21.1	tcp	172.16.4.10	8080	172.16.5.27	50334	13,036	22
2018-09-05 23:54:38.317Z	172.16.21.1	tcp	172.16.4.4	88	172.16.5.25	60386	606	8
2018-09-05 23:54:38.317Z	172.16.21.1	tcp	172.16.5.25	60306	172.16.4.4	88	820	8
2018-09-05 23:54:38.315Z	172.16.21.1	tcp	172.16.4.4	445	172.16.5.25	60385	1,832	8
2018-09-05 23:54:38.315Z	172.16.21.1	tcp	172.16.5.25	60385	172.16.4.4	445	2,550	12
2018-09-05 23:54:38.307Z	172.16.21.1	tcp	172.16.5.25	60384	172.16.4.4	88	820	8

# destination_port	🔍	🔍	🔍	🔍	* 45236
# engine_id	🔍	🔍	🔍	🔍	* 0
# engine_type	🔍	🔍	🔍	🔍	* 0
📄 exporter	🔍	🔍	🔍	🔍	* 172.16.21.1
🕒 first_switched	🔍	🔍	🔍	🔍	* 2018-09-05 23:12:30.503Z
# flow_records	🔍	🔍	🔍	🔍	* 1
# flow_records_vis	🔍	🔍	🔍	🔍	* 1
t host.name	🔍	🔍	🔍	🔍	* sof-elk
# in_bytes	🔍	🔍	🔍	🔍	* 4,995,286
# in_bytes_vis	🔍	🔍	🔍	🔍	* 4.8MB
# in_packets	🔍	🔍	🔍	🔍	* 1,382
# in_packets_vis	🔍	🔍	🔍	🔍	* 1.4k

Lab 3.1



Visual NetFlow Analysis with SOF-ELK®

This page intentionally left blank.

Lab 3.1 Objectives: Visual NetFlow Analysis with 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 3.1 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.

File Transfer Protocol (FTP)

This page intentionally left blank.

Welcome to the Olde School

- Designed for a simpler time
 - Firewalls were nonexistent
 - TCP ports were not scarce
 - Users were “trustworthy”
 - NAT simply wasn't needed
 - Network security hadn't been imagined yet

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 is today. It all started back in 1971. While at MIT, Abhay Bhushan wrote RFC 114, 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 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>

FTP's Origin Story and Modern Implementation

- Originally created for one simple purpose:
 - Move files from system A to system B
- 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

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

That's it.

But as with any simple, elegant solution, FTP has been Franken-protocol-ed 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.

References:

<http://for572.com/zgqx->

FTP Basics

- Multiple-stream protocol
 - Command channel (TCP/21)
 - Data channels (>TCP/1023 or source port TCP/20)
- “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”. 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.)

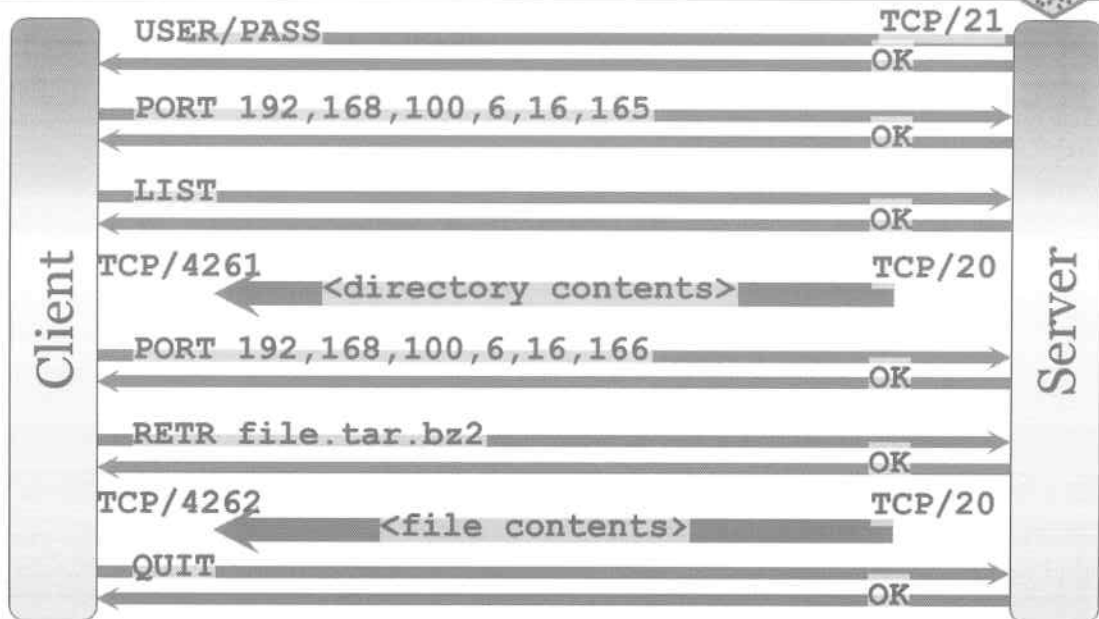
We will discuss this in more detail—and with pictures—in a few slides.

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 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?!)

Active FTP: Visualized



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 8 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 nonprivileged user may be using the FTP client software, but cannot typically bind to (aka “listen on”) a port between 1 and 1,023.

Shortcomings Today

- Firewalls complicate multistream protocols
- Does not handle NAT gracefully
 - “Passive” mode created to address these
 - Dynamic ports assigned in command channel
 - Firewalls must deeply inspect or proxy/intercept
- Plaintext

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 multistream nature of the protocol.

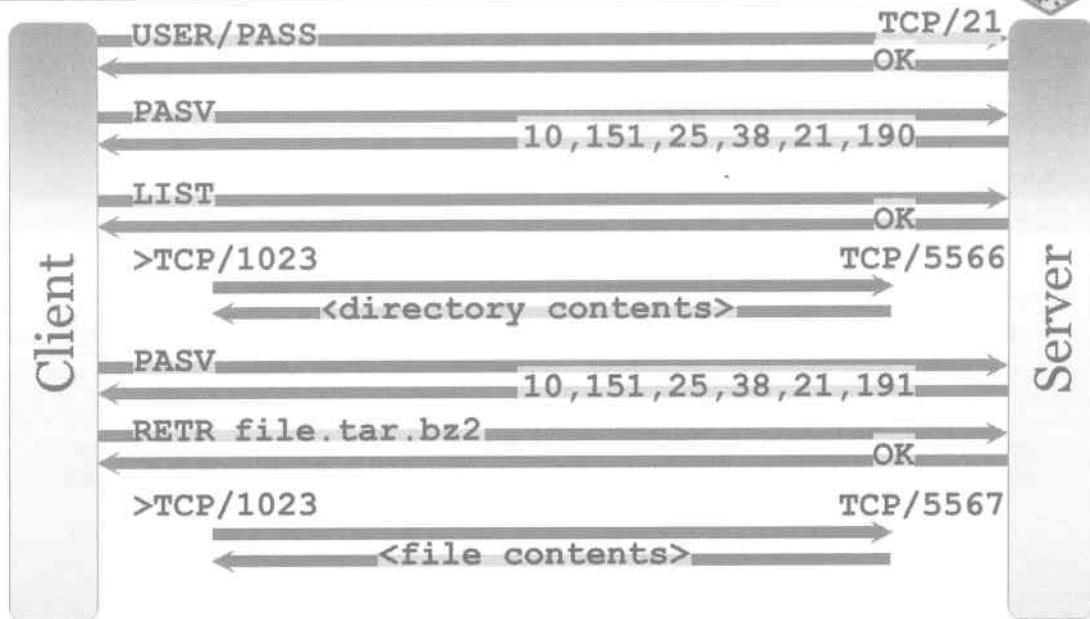
Firewalls are designed to limit network flows between two endpoints. NAT is a technology that enables a true endpoint's IP address to be altered without either end being aware of the change. FTP was designed with the expectation that traffic between endpoints can pass unimpeded, and the protocol's internals 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 FTP: Visualized

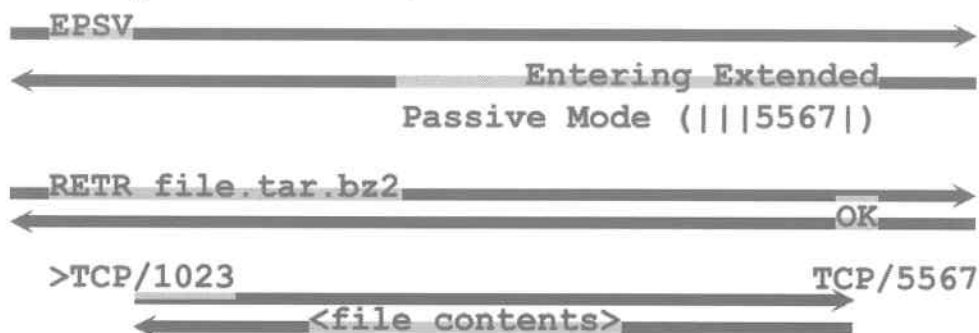


Passive mode FTP sessions generally use a similar command set and sequence of events. However, the client requests passive mode with the `PASV` command. The server then provides a reply similar to the `PORT` command we previously saw. The makeup and math behind these numerical values are the same.

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".

- 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>

- ~~\$ sudo tcpdump -i ens33 '(tcp and port 21)'~~
- \$ sudo tcpdump -i ens33 -w ftp_active_full.pcap `␣`
'(tcp and (port 21 or port 20))'
- \$ sudo tcpdump -i ens33 -w ftp_full.pcap `␣`
'(tcp and (port 21 or `␣`
((src portrange 1024-65535 or src port 20) and `␣`
(dst portrange 1024-65535 or dst port 20)))'



Just capturing TCP/21 leaves huge voids!

- 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

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. Since the response channels are generally established with a source port of TCP/20, this is a simple BPF.

```
$ sudo tcpdump -i ens33 -w ftp_active_full.pcap ␣  
'(tcp and (port 21 or port 20))'
```

However, this `tcpdump` command 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 ens33 -w ftp_full.pcap ␣  
'(tcp and (port 21 or ␣  
((src portrange 1024-65535 or src port 20) and ␣  
(dst portrange 1024-65535 or dst port 20)))'
```

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!



- Because it's an old protocol, there is broad FTP parsing support in commercial IDS, IPS, and perimeter tools
- Lots of reliable free/open-source tools
- Login details, command histories, automatically extract transferred files, etc.
- Server logs can be parsed for useful data

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 is the fact 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>

FTP File Extraction with Wireshark (I)



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 results in `tsniff` 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 available in the `/cases/for572/sample_pcaps/` directory within your SIFT workstation or the `/sample_pcaps/` directory on 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`”.

ftp-example.pcap

Expression...

No.	Time	Source	Destination	Protocol	Length	Info
767	21.13-11.22	192.168.75.29	149.20.20.135	FTP	108	Request: RETR yum-3.2.29-40.el6.centos.noarch.rpm
768	21.13-11.22	192.168.75.29	149.20.20.135	FTP	101	Request: RETR xchat-2.6.8-1.el6.x86_64.rpm
769	21.13-11.22	192.168.75.29	149.20.20.135	FTP	103	Request: RETR zenity-2.28.0-1.el6.x86_64.rpm
770	21.13-11.22	192.168.75.29	149.20.20.135	FTP	115	Request: RETR scenery-backgrounds-6.0.0-1.el6.noarch.rpm

Frame 767: 108 bytes on wire (864 bits), 108 bytes captured (864 bits) on interface eth0, Src: PcsCompu_97:ac:02:08:00:27:97:ac:02, Dst: Apple_72:08:01:2c:83:5e:dc:4a:5e:c8:47:80:18, Src Port: 173, Dst Port: 21, Seq: 173, Len: 108

```

Ethernet II, Src: PcsCompu_97:ac:02:08:00:27:97:ac:02, Dst: Apple_72:08:01:2c:83:5e:dc:4a:5e:c8:47:80:18
Internet Protocol Version 4, Src: 192.168.75.29, Dst: 149.20.20.135
Transmission Control Protocol, Src Port: 37628, Dst Port: 21, Seq: 173, Len: 108
File Transfer Protocol (FTP)
  Request command: RETR
    Request arg: yum-3.2.29-40.el6.centos.noarch.rpm\r\n
    Current working directory: centos/6.4/os/x86_64/Packages]
    Command response frames: 795]
    Command response bytes: 1019540]
    Command response first frame: 770]
    Command response last frame: 1662]
    Response duration: 1676ms]
    Response bitrate: 4866kbps]
    Setup frame: 762]
  
```

Packets: 35006 · Displayed: 4 (0.0%)

Profile: Default

FTP File Extraction with Wireshark (2)



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.

Profile: Default

Packets: 35006 - Displayed: 35006 (100.0%)

No.	Time	Source	Destination	Protocol	Length	Info
750	2013-11-22 16:38:51.796359	149.20.20.135	192.168.75.29	FTP	15	15... FTP Data: 1448 bytes (PASV) (LIST)
751	2013-11-22 16:38:51.796363	149.20.20.135	192.168.75.29	FTP	15	15... FTP Data: 1448 bytes (PASV) (LIST)
752	2013-11-22 16:38:51.796366	149.20.20.135	192.168.75.29	FTP	478	478... FTP Data: 412 bytes (PASV) (LIST)
753	2013-11-22 16:38:51.796714	192.168.75.29	149.20.20.135	TCP	66	66 49943 → 30760 [ACK] Seq=1 Ack=611470 Win=142720 Len=0 TSval=2505472004 ...
754	2013-11-22 16:38:51.801913	192.168.75.29	149.20.20.135	TCP	66	66 49943 → 30769 [FIN, ACK] Seq=1 Ack=611470 Win=153536 Len=0 TSval=2505472004 ...
755	2013-11-22 16:38:51.891286	149.20.20.135	192.168.75.29	TCP	66	66 30760 → 49943 [ACK] Seq=611470 Ack=2 Win=14592 Len=0 TSval=1799813084 ...
756	2013-11-22 16:38:51.891597	149.20.20.135	192.168.75.29	FTP	90	90 Response: 226 Directory send OK.
757	2013-11-22 16:38:51.891644	192.168.75.29	149.20.20.135	TCP	66	66 37028 → 21 [ACK] Seq=159 Ack=2927 Win=21504 Len=0 TSval=2505472099 TSec...
758	2013-11-22 16:39:00.267470	192.168.75.29	149.20.20.135	FTP	74	74 Request: TYPE I
759	2013-11-22 16:39:00.363497	149.20.20.135	192.168.75.29	FTP	97	97 Response: 200 Switching to Binary mode.
760	2013-11-22 16:39:00.363572	149.20.20.135	192.168.75.29	FTP	66	66 27028 → 21 [ACK] Seq=1 Ack=2927 Win=21504 Len=0 TSval=2505472099 TSec...
761	2013-11-22 16:39:00.363679	192.168.75.29	149.20.20.135	FTP	72	72 Request: PASV
762	2013-11-22 16:39:00.453372	149.20.20.135	192.168.75.29	FTP	118	118 Response: 227 Entering Passive Mode (149,20,20,135,120,173). TSval=25...
764	2013-11-22 16:39:00.497012	192.168.75.29	149.20.20.135	TCP	66	66 37028 → 21 [ACK] Seq=173 Ack=3010 Win=21504 Len=0 TSval=2505480704 TSec...
765	2013-11-22 16:39:00.541963	149.20.20.135	192.168.75.29	TCP	74	74 30893 → 30410 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM...
766	2013-11-22 16:39:00.541916	192.168.75.29	149.20.20.135	TCP	66	66 38410 → 30893 [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=25055480748 TSecr=...
767	2013-11-22 16:39:00.542002	192.168.75.29	149.20.20.135	FTP	108	108 Request: RETR yum-3.2.29-40.el6.centos.noarch.rpm

```

- 227 Entering Passive Mode (149,20,20,135,120,173).\r\n
Response code: Entering Passive Mode (227)
Response arg: Entering Passive Mode (149,20,20,135,120,173).
Passive IP address: 149.20.20.135
Passive port: 30893
[Current working directory: centos/6.4/os/x86_64/Packages]
[Command: RETR yum-3.2.29-40.el6.centos.noarch.rpm]
[Command frame: 767]

```

Apply a display filter: *Ctrl*

FTP File Extraction with Wireshark (3)

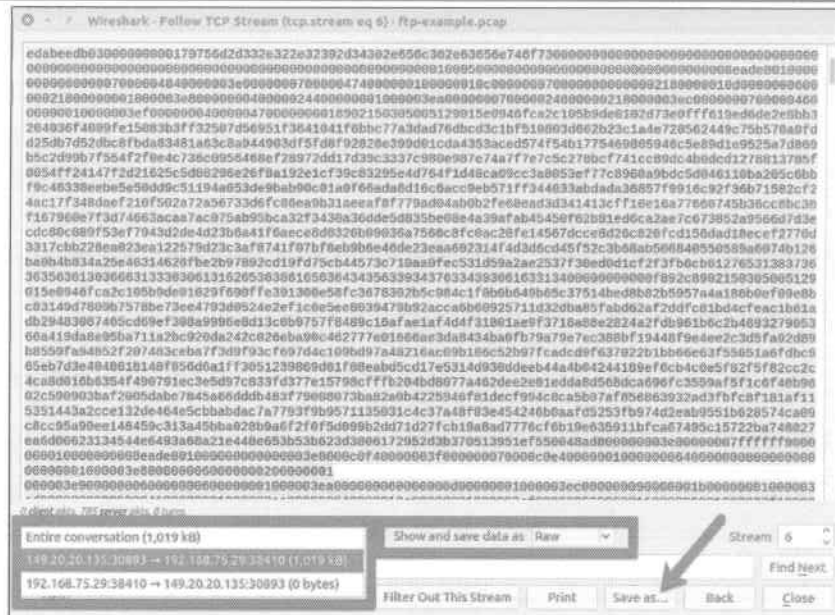


Using this IP and TCP port number, we have created a display filter to isolate network data matching these parameters. The display filter for this is:

```
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 occurring 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. The “rpm -K” command performs a cryptographic integrity check using PGP, which provides complete validation that the file was extracted correctly.

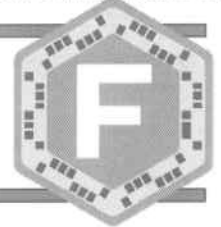
```
$ ls -l yum-3.2.29-40.el6.centos.noarch.rpm
-rw-rw-rw- 1 sansforensics sansforensics 1019540 2013-11-22 17:11 yum-3.2.29-40.el6.centos.noarch.rpm

$ 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
```

```
$ rpm -qpi yum-3.2.29-40.el6.centos.noarch.rpm
Name       : yum
Version    : 3.2.29
Release    : 40.el6.centos
Architecture: noarch
Install Date: (not installed)
Group      : System Environment/Base
Size       : 4735341
License    : GPLv2+
Signature  : RSA/SHA1, Sat 23 Feb 2013 05:50:22 PM UTC, Key ID
0946fca2c105b9de
Source RPM : yum-3.2.29-40.el6.centos.src.rpm
Build Date : Fri 22 Feb 2013 11:26:41 AM UTC
Build Host : c6b8.bsys.dev.centos.orgRelocations : (not relocatable)
Packager   : CentOS BuildSystem <http://bugs.centos.org>
Vendor     : CentOS
URL        : http://yum.baseurl.org/
Summary    : RPM package installer/updater/manager
Description: Yum is a utility that can check for and automatically download
and
install updated RPM packages. Dependencies are obtained and downloaded
automatically, prompting the user for permission as necessary.
*
$ rpm -K yum-3.2.29-40.el6.centos.noarch.rpm
yum-3.2.29-40.el6.centos.noarch.rpm: rsa sha1 (md5) pgp md5 OK
```

Lab 3.2



Tracking Lateral Movement with NetFlow

This page intentionally left blank.

Lab 3.2 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.



- 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.

Microsoft Protocols

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 needed to compromise a system. Once the attackers gain 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 ones the attacker conducted. At a technical level, the analyst will need to understand the data being accessed by SMB and email 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, and 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/lpz9u>

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 email client and Exchange the most common enterprise email/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 not be readdressed in this section, but are commonly seen in MS networking environments.

Point of Visibility Drives Expected Protocols Seen

Internal Vantage Point

- File access
 - Domain authentication
 - SMB file access
 - Group Policy synchronization
- Email
 - Outlook-to-Exchange
 - POP, IMAP, Webmail
- SharePoint/Web
 - HTTP or HTTPS

Perimeter Vantage Point

- VPN
 - Roaming users
 - Site-to-site
- Email
 - SMTP (TLS)
 - Webmail or Office 365
 - Outlook (RPC over HTTP)
 - Mobile Exchange clients (ActiveSync)
- SharePoint/Web
 - HTTP or HTTPS

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

Email may be transmitted in clear (both inbound and outbound), but the increasing use of 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 TLS
 - SMTP – Needed to send email with IMAP/POP3 accounts
 - Increasingly uses TLS, especially for server-to-server relaying

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 email 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 email can be read with the human eye.

As a multipurpose email client, Outlook can be configured to simultaneously access multiple email 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 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 email 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>

Key SMB Release Dates

- Core file access protocol developed by Microsoft
 - SMB – 1980s
 - CIFS – 1996
 - SMB 1.0 (Windows 2000 Extensions) – 2000
 - SMB 2.0/2.0.2 (aka SMB2) – 2008
 - SMB 2.1 – 2010
 - SMB 3.0 (aka SMB3) – 2012
 - SMB 3.0.2 (aka SMB3.02) – 2013
 - SMB 3.1.1 – 2015

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 provide a command set that allows a local user to create, edit, modify and delete 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. These ports are also used for IPC\$ and Null Sessions under both SMB1 and SMB2.

After CIFS and SMB1, Microsoft released updates to the protocol in line with the other operating system improvements. With the release of SMB 2.0, the protocol used TCP port 445, as it no longer required the NetBIOS layer and needed a means of differentiating the encapsulation (or lack thereof) used.

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 10, Svr 2016 (v1709, v1803)
Win XP/2k, Svr 2003	SMB 1.0	SMB 1.0	SMB 1.0	SMB 1.0	SMB 1.0	SMB 1.0	XXXX
Win Vista, Svr 2008	SMB 1.0	SMB 2.02	SMB 2.02	SMB 2.02	SMB 2.02	SMB 2.02	SMB 2.02
Win 7, Svr 2008 R2	SMB 1.0	SMB 2.02	SMB 2.1	SMB 2.1	SMB 2.1	SMB 2.1	SMB 2.1
Win 8, Svr 2012	SMB 1.0	SMB 2.02	SMB 2.1	SMB 3.0	SMB 3.0	SMB 3.0	SMB 3.0
Win 8.1, Svr 2012 R2	SMB 1.0	SMB 2.02	SMB 2.1	SMB 3.0	SMB 3.02	SMB 3.02	SMB 3.02
Win 10, Svr 2016	SMB 1.0	SMB 2.02	SMB 2.1	SMB 3.0	SMB 3.02	SMB 3.1.1	SMB 3.1.1
Win 10, Svr 2016 (v1709, v1803)	XXXX	SMB 2.02	SMB 2.1	SMB 3.0	SMB 3.02	SMB 3.1.1	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.

However, with the release of Windows 10 Creators Edition and the corresponding Windows Server 2016 update in late 2017, note that SMB1 has finally been removed by default – an arguably long-overdue move prompted by the WannaCry outbreak, which exploited vulnerabilities in the legacy version of the SMB protocol.

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>

<http://for572.com/ftvqg>

<http://for572.com/t6d47>

Who Else Uses What Version?

Vendor	Product	SMB Version
Apple	macOS 10.14 Mojave	SMB 3.02
Apple	macOS 10.13 High Sierra	SMB 3.0 (Required for APFS shares and Time Machine)
Apple	OS X 10.11 El Capitan, 10.10 Yosemite, 10.12 Sierra	SMB 3.0
Apple	OS X 10.9 Mavericks	SMB 2.0 (Replacing AFP!)
EMC	VNX 8.1.9	SMB 3.02
EMC	VNX, Isilon 7.1.1	SMB 3.0
EMC	Isilon 7.0	SMB 2.1
EMC	Isilon 6.5	SMB 2
NetApp	Data ONTAP 9	SMB 3.1
NetApp	Data ONTAP 8.2	SMB 3.0
NetApp	Data ONTAP 7.3.1	SMB 2.0
Samba	Samba 4.3	SMB 3.1.1
Samba	Samba 4.1, 4.2	SMB 3.0
Samba	Samba 3.6	SMB 2.0 (with some 2.1)

As can be seen from the table above, if you have a multiplatform 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.

Note that older versions of EMC, NetApp, and Samba all solely supported SMB1, with many more recent versions shown negotiating back to the prior versions for backward compatibility reasons.

Source data correct as of November 2015 (source: SNIA SMB3 Remote File Protocol Presentation by Jose Barreto of Microsoft (author-approved reproduction)), augmented for FOR572 with direct research. (Thanks, Mike P!)

References:

<http://for572.com/8yz0d>

<http://for572.com/xopaq>

<http://for572.com/63bmn>

<http://for572.com/autsn>

<http://for572.com/yja4b>

Key Changes with SMB3

- SMB3 used with Windows 8+ and Server 2012+
- Encryption using AES-CCM
- Signing using AES-CMAC
- Volume Shadow Copies over SMB
- Transparent server failover
- Secure Dialect Negotiation
 - Detect attempted “Meddler-in-the-Middle” attacks
- Some features lock out SMB2 clients

It's becoming more common to see Windows environments migrate toward newer client and server builds, which also brings a corresponding increase in SMB3. By default, a client that uses Windows 8 or newer will use SMB3 when connecting to a server running Windows Server 2012 or newer (or another Windows 8 or newer client that is acting as a server).

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 prior versions. However, an outdated analysis platform will not provide the same visibility for this newer protocol variant.

References:

[1] <http://for572.com/wqdmv>

Forensic Goals for SMB Analysis

- Understand user/attacker actions
 - Where have they been?
 - What have they looked at?
- Detect patterns of activity or targeting
- Support other forensic or DFIR roles
 - Corroborate findings from other sources

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 subdisciplines.

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 email, 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 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 cleanup 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 and SMB3 Commands

- **Protocol negotiation**
 - NEGOTIATE, SESSION_SETUP, LOGOFF, TREE_CONNECT, TREE_DISCONNECT
- **Metadata query and modification**
 - QUERY_DIRECTORY, CHANGE_NOTIFY, QUERY_INFO
- **File, directory, and volume access**
 - CREATE, CLOSE, FLUSH, READ, WRITE, LOCK, IOCTL, SET_INFO
- **Other**
 - CANCEL, ECHO, OPLOCK_BREAK

The SMB2 and SMB3 command sets consist 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.

In this section, we will examine several of these that provide the most useful forensic insight.

References:

[1] <http://for572.com/xzsk9>

[2] <http://for572.com/bpyde>

“Typical” SMB2/SMB3 File Access Session

- NEGOTIATE
- SESSION_SETUP
- TREE_CONNECT
- CREATE
 - Get directory handle
- QUERY_DIRECTORY
- CREATE
 - Get/create file handle
- READ/WRITE
- CLOSE
- TREE_DISCONNECT
- LOGOFF

Many different paths through SMB2/SMB3 operations!

The extensive nature of the SMB2 and SMB3 protocol variants 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. Other commands listed on the previous slide may be seen in operational environments, or may be absent. The various ways of configuring the services provide limitless sequences and combinations.

In operational usage, each command above typically 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 uncleanly, for example. Additionally, network conditions might cause some messages to occur out of sequence. The walkthrough in this section serves as a general guideline rather than an absolute process map.

SMB2/3: Protocol Negotiation



```
SMB (Server Message Block Protocol)
  SMB Header
    Server Component: SMB
    SMB Command: Negotiate Protocol (0x72)
    NT Status: STATUS_SUCCESS (0x00000000)
    Flags: 0x18, Canonicalized Pathnames, Ca
    Flags2: 0xc853, Unicode Strings, Error C
    Process ID High: 0
    Signature: 0000000000000000
    Reserved: 0000
    Tree ID: 65535
    Process ID: 65279
    User ID: 0
    Multiplex ID: 0
  Negotiate Protocol Request (0x72)
    Word Count (WCT): 0
    Byte Count (BCC): 120
    Requested Dialects
      Dialect: PC NETWORK PROGRAM 1.0
      Dialect: LANMAN1.0
      Dialect: Windows for Workgroups 3.1a
      Dialect: LM1.2X002
      Dialect: LANMAN2.1
      Dialect: NT LM 0.12
      Dialect: SMB 2.002
      Dialect: SMB 2.777
```

Client Request

• NEGOTIATE_SESSION

• Display filter:

```
"smb.cmd==0x72 ||
smb2.cmd == 0"
```

```
Negotiate Protocol Response (0x00)
  StructureSize: 0x0041
  Security Flags: 0x00000001, Signing enabled
  Dialect: 0x0302
  NegotiateContextOffset: 0
  Server Guid: a136751d-2830-4bfc-aa19-4023b4d38e27
  Capabilities: 0x0000000f, DFS, LEASING, LARGE MTU, MULTI CHANNEL, D
  Max Transaction Size: 8388608
  Max Read Size: 8388608
  Max Write Size: 8388608
  Current Time: Sep 5, 2018 14:00:53.059509800 UTC
  Boot Time: Aug 8, 2018 18:08:06.226199200 UTC
  Blob Offset: 0x00000000
  Blob Length: 128
  Security Blob: 607606062b0681058582a86c396aa83c303a060a2b060104...
  NegotiateContextOffset: 0x0000
```

Server Response

The first command, "NEGOTIATE_SESSION", is used to establish 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 pane. In this section, the client is running Microsoft Windows 10 v1709 and the server is running Microsoft Windows Server 2012 R2. The presence of such ancient negotiation protocols on a modern system may be a shock, but at this early stage in the negotiation, the client has to be "ready for anything". If a server were to reply with a preference for the older dialects, the newer client would refuse to continue.

The server's response packet includes several important fields, notably the selected dialect. In this case, the newer pair of systems agree to use SMB version 3.02. The server also sends over its current date and time as well as the date and time when it originally booted. The "Security Blob" near the end of the packet data sets up for the subsequent authentication process, which typically requires several steps of challenge-response to avoid sending sensitive data over a plaintext channel.

```

[-] SMB (Server Message Block Protocol)
  [-] SMB Header
    Server Component: SMB
    SMB Command: Negotiate Protocol (0x72)
    NT Status: STATUS_SUCCESS (0x00000000)
    [+ Flags: 0x18, Canonicalized Pathnames, Cas
    [+ Flags2: 0xc853, Unicode Strings, Error C
    Process ID High: 0
    Signature: 0000000000000000
    Reserved: 0000
    Tree ID: 65535
    Process ID: 65279
    User ID: 0
    Multiplex ID: 0
  [-] Negotiate Protocol Request (0x72)
    Word Count (WCT): 0
    Byte Count (BCC): 120
    [-] Requested Dialects
      [+ Dialect: PC NETWORK PROGRAM 1.0
      [+ Dialect: LANMAN1.0
      [+ Dialect: Windows for Workgroups 3.1a
      [+ Dialect: LM1.2X002
      [+ Dialect: LANMAN2.1
      [+ Dialect: NT LM 0.12
      [+ Dialect: SMB 2.002
      [+ Dialect: SMB 2.???

```

Client Request

```

[-] Negotiate Protocol Response (0x00)
  [+ StructureSize: 0x0041
  [+ Security, mode: 001, Signing enabled
  [+ Dialect: 0x0302
  [+ NegotiateContextCount: 0
  Server Guid: a136751d-2830-4bfc-aa19-4023b4d38e27
  [+ Capabilities: 0x0000006f, DFS, LEASING, LARGE MTU, MULTI CHANNEL, D
  Max Transaction Size: 8388608
  Max Read Size: 8388608
  Max Write Size: 8388608
  Current Time: Sep 5, 2018 14:06:53.059509800 UTC
  Boot Time: Aug 8, 2018 18:08:06.226199200 UTC
  Blob Offset: 0x00000080
  Blob Length: 120
  [+ Security Blob: 607606062b0601050502a06c306aa03c303a060a2b060104...
  NegotiateContextOffset: 0x0000

```

Server Response



- SESSION_SETUP
 - Display filter: "smb2.cmd == 1"
- User authentication
 - Challenge-response
- Server assigns Session ID
 - e.g. 0x0000900014000049, "smb2.sesid"

```

- Security Blob: a182022f3082922ba0030a019...
- GSS-API Generic Security Service Applic...
  - Simple Protected Negotiation
    - negTokenTarg
      - negResult: accept-incomplete (1)
      - responseToken: 4e544c4d5353500000...
    - NTLM Secure Service Provider
      - NTLMSSP identifier: NTLMSSP
      - NTLM Message Type: NTLMSSP_AUT...
    - Lan Manager Response: 00000000
    - LMv2 Client Challenge: 00000000
    - NTLM Response: 57c7eb1f4e70424...
  - Domain name: shieldbase
  - User name: spsql
  - Host name: BASE-RD-01
  - Session Key: 000000000000000000000000...
  - Negotiate Flags: 0x00000000
    
```

Client Request

```

- Security Blob: a11b3019a0030a0100a312041001000000a65d0314ce75ff...
- GSS-API Generic Security Service Application Program Interface
  - Simple Protected Negotiation
    - negTokenTarg
      - negResult: accept-completed (0)
      - responseToken: 000000000000000000000000...
    
```

Server Response

After negotiating the communication parameters, the client sends login credentials according to the server’s directed negotiation dialect. This step uses the “SESSION_SETUP” command. Note that in these examples, we are showing just one request/response pair per volley. Since SMB allows for “stacking” multiple commands into a single message, it’s likely that you will encounter more complex samples in the wild.

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. While authentication also occurs at this stage, the password is not sent in cleartext. This is sent per the previously-agreed-upon negotiation dialect.

The exchange shown above also includes Generic Security Service – Application Program Interface (GSS-API) data, which handles client authentication through the use of the Simple Protected Negotiation (SPNEGO) specification. Several SPNEGO fields contain Unicode content, making them reasonably easy to read even without a parser such as Wireshark. The SPNEGO fields also include various hash values and tokens that permit domain authentication to occur.

Assuming the client successfully authenticates, the server’s response also includes a Session ID value. This 8-byte value will be used by the client and the server for the duration of the connection to uniquely identify the authenticated user.

```

- Security Blob: a182022f3082022ba0030a0101
- GSS-API Generic Security Service Applic
  - Simple Protected Negotiation
    - negTokenTarg
      negResult: accept-incomplete (1)
      responseToken: 4e544c4d5353500000
    - NTLM Secure Service Provider
      NTLMSSP identifier: NTLMSSP
      NTLM Message Type: NTLMSSP_AUT
      + Lan Manager Response: 00000000
      LMv2 Client Challenge: 00000000
      + NTLM Response: 57c7eb1f4e78424
        + Domain name: shieldbase
        + User name: spsql
        + Host name: BASE-RD-01
      + Session Key: 953ecae177a417496
      + Negotiate Flags: 0xe2888215
  
```

Client Request

```

- Security Blob: a11b3019a0030a0100a312041001000000a65d0314ce75ff...
- GSS-API Generic Security Service Application Program Interface
  - Simple Protected Negotiation
    - negTokenTarg
      negResult: accept-completed (0)
      responseToken: 01000000a65d0314ce75ff6500000000
  
```

Server Response

SMB2/3: Access Services



- TREE_CONNECT
 - Display filter: “smb2.cmd == 3”
- Server assigns Tree ID, “smb2.tid”
 - Used for all subsequent actions under this resource
 - Only valid/meaningful for this session

```
Tree Connect Request (0x03)
  StructureSize: 0x0009
  Reserved: 0000
  Tree: \\172.16.4.5\c$
  Blob Offset: 0x00000048
  Blob Length: 30
```

Client Request

```
Tree Id: 0x00000005 \\172.16.4.5\c$
  Tree: \\172.16.4.5\c$
  [Share Type: Physical disk (0x01)]
  [Connected in Frame: 66]
```

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. Note that in the reply, Wireshark includes the Tree name as well as some additional fields. It's important to recognize that these fields are not actually present in the resulting packet. Instead, Wireshark is displaying those as a convenience to the user. These pseudo-fields are “imported” from elsewhere in the session. Such pseudo-fields are present in many of Wireshark's protocol parsers and are designated in the Packet Detail pane with square brackets. Pseudo-fields are available for display filters and field extraction as well.

In this response packet, the server allocates a Tree ID, another ephemeral value that will only last for the duration of the SMB session.

```
Tree Connect Request (0x03)
  StructureSize: 0x0009
  Reserved: 0000
  Tree: \\172.16.4.5\c$
  Blob Offset: 0x00000048
  Blob Length: 30
```

Client Request

```
Tree Id: 0x00000005 \\172.16.4.5\c$
  Tree: \\172.16.4.5\c$
  [Share Type: Physical disk (0x01)]
  [Connected in Frame: 66]
```

Server Response

SMB2/3: Directory Navigation



- CREATE
 - Display filter: “smb2.cmd == 5”

```
- Create Request (0x05)
  - StructureSize: 0x0039
    Oplock: Lease (0xff)
    Impersonation level: Impersonation (2)
    Create Flags: 0x0000000000000000
    Reserved: 0000000000000000
  - Access Mask: 0x00100001
  - File Attributes: 0x00000000
  - Share Access: 0x00000007, Read, Write, Delete
    Disposition: Open (if file exists open it, else fail) (1)
  - Filename: Windows
  - Blob Offset: 0x00000000
  - Blob Length: 180
  - ExtraInfo SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 SMB2_CREATE
```

Client Request

```
- Create Response (0x05)
  - StructureSize: 0x0059
    Oplock: Lease (0xff)
    Response Flags: 0x00
  - Create Action: The file existed and was opened (1)
    Create: Aug 22, 2018 13:38:16.026448200 UTC
    Last Access: Aug 15, 2018 17:17:29.590615300 UTC
    Last Write: Aug 15, 2018 17:17:29.590615300 UTC
    Last Change: Aug 15, 2018 17:17:29.590615300 UTC
  - Allocation Size: 24576
  - End of File: 24576
  - File Attributes: 0x00000010
  - Reserved: 00000000
  - GUID handle File: Windows
  - File Id: 00001f1b-0024-0000-2100-000024000000
  - [Frame handle opened: 100]
  - [Frame handle closed: 2704]
```

Server Response

In this phase of the session, the client requests information about the “\windows\” directory. This path is relative to the Tree path from the previous set of messages – so “\\172.16.4.5\c\$\windows\” in this case. 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 server also issues a GUID File ID value. This 16-byte field indicates the client has navigated to this directory, retrieving this ephemeral directory handle value. Note that Wireshark has also included links to the frames in which that directory handle was opened and closed. Double-clicking these will take the user to the respective frame in the packet capture.

```

- Create Request (0x05)
  + StructureSize: 0x0039
    Oplock: Lease (0xff)
    Impersonation level: Impersonation (2)
    Create Flags: 0x0000000000000000
    Reserved: 0000000000000000
  + Access Mask: 0x00100081
  + File Attributes: 0x00000000
  + Share Access: 0x00000007, Read, Write, Delete
    Disposition: Open (if file exists open it, else fail) (1)
  + Filename: Windows
    Blob Offset: 0x00000088
    Blob Length: 180
  + ExtraInfo SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 SMB2_CREAT

```

Client Request

```

- Create Response (0x05)
  + StructureSize: 0x0059
    Oplock: Lease (0xff)
  + Response Flags: 0x00
  + Create Action: The file existed and was opened (1)
    Create: Aug 22, 2013 13:36:16.025448200 UTC
    Last Access: Aug 15, 2018 17:17:29.590615300 UTC
    Last Write: Aug 15, 2018 17:17:29.590615300 UTC
    Last Change: Aug 15, 2018 17:17:29.590615300 UTC
    Allocation Size: 24576
    End Of File: 24576
  + File Attributes: 0x00000010
    Reserved: 00000000
  + GUID handle File: Windows
    File Id: 00001f15-0024-0000-2100-000024000000
    [Frame handle opened: 106]
    [Frame handle closed: 2704]

```

Server Response

SMB2/3: Obtain Directory Listing

- QUERY_DIRECTORY

- Display filter:
“smb2.cmd == 14”

```
- Find Request (0x0c)
  - StructureSize: 0x0021
  - Info Level: SMB2_FIND_ID_BOTH_DIRECTORY_INFO (37)
  - Find Flags: 0x00
  - File Index: 0x00000000
  - GUID handle File: Windows
  - File Id: 00001f15-0024-0000-2100-000024000000
  - [Frame handle opened: 1061]
  - [Frame handle closed: 2704]
  - Search Pattern: *
  - Blob Offset: 0x00000000
  - Blob Length: 2
```

Client Request

```
- Find Response (0x0e)
  [Info Level: SMB2_FIND_ID_BOTH_DIRECTORY_INFO (37)]
  - StructureSize: 0x0009
  - Blob Offset: 0x00000048
  - Blob Length: 11682
  - Info: 7000000000000000122a68933c9fce0169b978d8bb34d481...
  - FileIdBothDirectoryInfo: ..
  - FileIdBothDirectoryInfo: ..
  - FileIdBothDirectoryInfo: ADFS
  - FileIdBothDirectoryInfo: AppCompat
  - FileIdBothDirectoryInfo: apppatch
  - FileIdBothDirectoryInfo: AppReadiness
  - FileIdBothDirectoryInfo: assembly
  - FileIdBothDirectoryInfo: bfsvc.exe
  - Next Offset: 128
  - File Index: 0x00000000
  - Create: Aug 22, 2013 11:21:53.417682000 UTC
  - Last Access: Aug 22, 2013 11:21:53.417682000 UTC
  - Last Write: Aug 22, 2013 11:21:47.213991000 UTC
  - Last Change: Aug 11, 2016 22:59:26.549389000 UTC
  - End Of File: 56832
  - Allocation Size: 57344
  - File Attributes: 0x00000020
  - Filename Length: 18
  - File Size: 0
```

Server Response

The client can obtain directory listings with the “QUERY_DIRECTORY” message. This includes a root directory from which to search, referenced by GUID File ID. It also includes a search pattern, which is a wildcard in this case.

The server’s response includes the list of files as well as typical filesystem metadata for each. Note the MACB times once again, as well as the “End Of File” value, which is more commonly recognized as the file size.

```

- Find Request (0x0e)
  + StructureSize: 0x0021
    Info Level: SMB2_FIND_ID_BOTH_DIRECTORY_INFO (37)
  + Find Flags: 0x00
    File Index: 0x00000000
  - GUID handle File: Windows
    File Id: 00001f15-0024-0000-2100-000024000000
    [Frame handle opened: 106]
    [Frame handle closed: 2704]
    Output Buffer Length: 65536
  - Search Pattern: *
    Blob Offset: 0x00000060
    Blob Length: 2

```

Client Request

```

- Find Response (0x0e)
  [Info Level: SMB2_FIND_ID_BOTH_DIRECTORY_INFO (37)]
  + StructureSize: 0x0009
    Blob Offset: 0x00000048
    Blob Length: 11082
  - Info: 7000000000000000122a68933c9fce0169b978d8bb34d401...
    + FileIdBothDirectoryInfo: .
    + FileIdBothDirectoryInfo: ..
    + FileIdBothDirectoryInfo: ADFS
    + FileIdBothDirectoryInfo: AppCompat
    + FileIdBothDirectoryInfo: apppatch
    + FileIdBothDirectoryInfo: AppReadiness
    + FileIdBothDirectoryInfo: assembly
  - FileIdBothDirectoryInfo: bfsvc.exe
    Next Offset: 128
    File Index: 0x00000000
    Create: Aug 22, 2013 11:21:53.417682800 UTC
    Last Access: Aug 22, 2013 11:21:53.417682800 UTC
    Last Write: Aug 22, 2013 11:21:47.213991000 UTC
    Last Change: Aug 11, 2016 22:59:26.549389000 UTC
    End Of File: 56832
    Allocation Size: 57344
  + File Attributes: 0x00000020
    Filename Length: 18
    EA Size: 0

```

Server Response

SMB2/3: Open a File



- CREATE

- Display filter: “smb2.cmd == 5”
- Client sends filename (smb2.filename)
- Server assigns GUID File ID (smb2.fid)

```
- Create Request (0x95)
  > StructureSize: 0x0039
  > Oplock: Lease (0x0f)
  > Impersonation Level: Impersonation (2)
  > Create Flags: 0x0000000000000000
  > Reserved: 0000000000000000
  > Access Mask: 0x00120000
  > File Attributes: 0x00000000
  > Share Access: 0x00000007, Read, Write, Delete
  > Disposition: Open (if file exists open it, else fail) (1)
  > Filename: Windows\ToastData\desktop.ini
  > Blob Length: 50
  > Blob Offset: 0x00000000
  > Blob Length: 100
```

Client Request

```
Response Flags: 0x00
Create Action: The file existed and was opened (1)
Create: Aug 22, 2016 15:39:47.291848000 UTC
Last Access: Aug 16, 2016 16:20:32.077013800 UTC
Last Write: Aug 16, 2016 16:28:32.078901900 UTC
Last Change: Aug 11, 2016 23:01:05.330348400 UTC
Allocation Size: 480
End Of File: 477
File Attributes: 0x00000000
.....0 = Read Only: NOT read only
.....1 = Hidden: HIDDEN
.....1 = System: SYSTEM file/dir
.....0 = Volume ID: NOT a volume
.....0 = Directory: NOT a directory
.....1 = Archive: Modified since
.....0 = Device: NOT a device
.....0 = Normal: Has some attributes
```

Server Response

In this exchange, the client requests a file handle to a specific resource. The server will validate the request against its permissions, including the user, share, and ACL checks. If the resource exists and the client is permitted to access it, the server will return another GUID File ID value, unique to this SMB session, with that particular client. The FID value does not have a persistent relationship to the resource path outside of a single session. A client's subsequent accesses to the same file will result in a different GUID File ID value. Therefore, if you are tracking access to a specific file within a pcap file, use the filename and not any of its GUID File ID values.

The server's response message includes the file-based permissions the client system has to the resource, based on the authentication it has established. The response typically includes a bit mask called the “SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST”, which contains a list of all permissions the authenticated user has for the resource.

Note that this SMB command is also used to create a new resource. There are six different “Disposition” values defined in the SMB2/3 specification that allow the client to dictate what will happen if a requested resource is not available, including creating a new file.

```

[-] Create Request (0x05)
  [+] StructureSize: 0x0039
    Oplock: Lease (0xff)
    Impersonation level: Impersonation (2)
    Create Flags: 0x0000000000000000
    Reserved: 0000000000000000
  [+] Access Mask: 0x00120089
  [+] File Attributes: 0x00000000
  [+] Share Access: 0x00000007, Read, Write, Delete
  [+] Disposition: Open (if file exists open it, else fail) (1)
  [-] Filename: Windows\ToastData\desktop.ini
    Blob Offset: 0x00000070
    Blob Length: 58
    Blob Offset: 0x000000b8
    Blob Length: 180

```

Client Request

```

[-] Response Flags: 0x00
  Create Action: The file existed and was opened (1)
  Create: Aug 22, 2013 15:39:47.291848400 UTC
  Last Access: Aug 10, 2016 16:28:32.077013800 UTC
  Last Write: Aug 10, 2016 16:28:32.078001900 UTC
  Last Change: Aug 11, 2016 23:01:05.336349400 UTC
  Allocation Size: 480
  End Of File: 477
  [-] File Attributes: 0x00000026
    .....0 = Read Only: NOT read only
    .....1. = Hidden: HIDDEN
    .....1.. = System: SYSTEM file/dir
    .....0... = Volume ID: NOT a volume
    .....0.... = Directory: NOT a directory
    .....1. .... = Archive: Modified since last access
    .....0.. .... = Device: NOT a device
    .....0... .... = Normal: Has some attributes

```

Server Response

SMB2/3: Read from a File



- READ/WRITE
 - Display filter:
“smb2.cmd == 8”
 - Partial file reads with byte offsets and length
 - Cannot “Follow TCP Stream” to get original file

```
- Read Request (0x00)
  - StructureSize: 0x0031
    Reserved: 5000
    Read Length: 477
    File Offset: 0
  - GUID handle File: Windows\ToastData\desktop.ini
    File Id: 00001f1a-0024-0000-3500-000024000000
    [Frame handle opened: 143]
    [Frame handle closed: 195]
  Min Count: 0
  Channel: None (0x00000000)
  Remaining Bytes: 0
  Blob Offset: 0x00000000
  Blob Length: 0
```

Client Request

```
- SMB2 Header
- Read Response (0x00)
  - StructureSize: 0x0011
    Data Offset: 0x0050
    Read Length: 477
    Read Remaining: 0
    Reserved: 00000000
  Data (477 bytes)
    Data: 0d0a5b4c6f63616c697a656446696c654e616d65735d0d0a...
    [length: 477]
```

Server Response

To read file contents, the client sends the “READ” message, requesting content by referencing the GUID File ID, requested number of bytes, and offset at which to start reading. The server's corresponding response contains the file data, among other fields. Writing a file to an SMB destination uses the same process but the message type is “WRITE”.

Because the allowable packet size is smaller than many files, a complete access might require hundreds or thousands of READ or WRITE 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 Request (0x08)
  [+ StructureSize: 0x0031
    Reserved: 5000
    Read Length: 477
    File Offset: 0
  [-] GUID handle File: Windows\ToastData\desktop.ini
    File Id: 00001f1a-0024-0000-3500-000024000000
    [Frame handle opened: 143]
    [Frame handle closed: 195]
    Min Count: 0
    Channel: None (0x00000000)
    Remaining Bytes: 0
    Blob Offset: 0x00000000
    Blob Length: 0
```

Client Request

```
[+] SMB2 Header
[-] Read Response (0x08)
  [+ StructureSize: 0x0011
    Data Offset: 0x0050
    Read Length: 477
    Read Remaining: 0
    Reserved: 00000000
  [Data (477 bytes)
    Data: 0d0a5b4c6f63616c697a656446696c6554e616d65735d0d0a...
    [Length: 477]
```

Server Response



- CLOSE
 - Display filter: “smb2.cmd == 6”
 - De-assigns GUID FILE ID
- TREE_DISCONNECT
 - Display filter: “smb2.cmd == 4”
 - De-assigns Tree ID
- LOGOFF
 - Display filter: “smb2.cmd == 2”
 - De-assigns Session ID
 - Does not happen often

After the file is unlocked, the client sends the “CLOSE” message to request that the server close the file handle. The ephemeral GUID File ID value is de-assigned from the server’s pool at this stage.

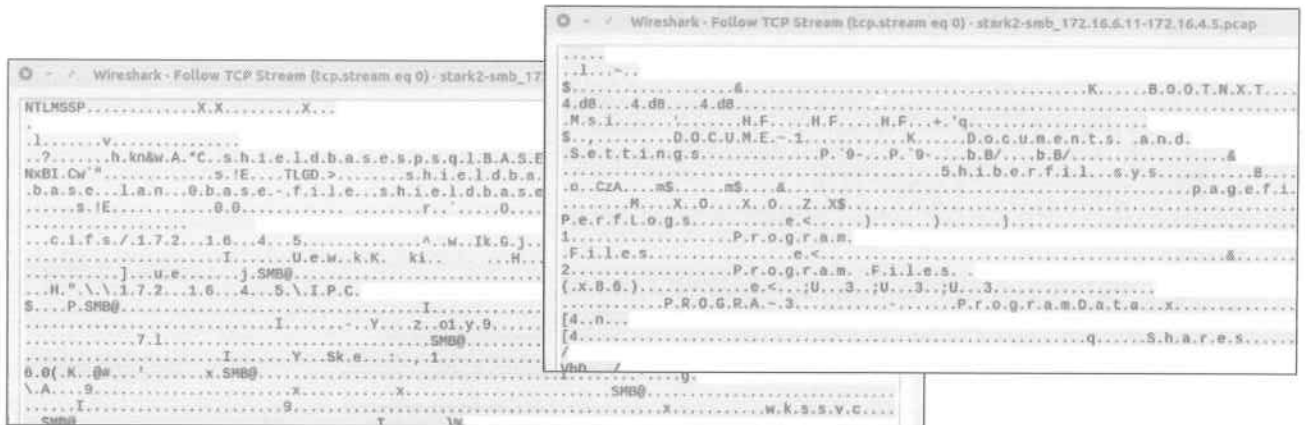
Each tree attachment requires an explicit disconnection request. These messages essentially un-map a drive letter or stock tracking a UNC connection. The ephemeral Tree ID value is de-assigned and returned to the pool of available values for the server to potentially use on new TREE_CONNECT requests.

Lastly, the logoff can occur after all trees are de-assigned. However, this is not a common event – especially not in relation to the other SMB messages, which are quite numerous. This event only happens on an explicit disconnect event, or after all trees have been disconnected. In most domain configurations, the client may automatically mount a shared drive at system login. This incurs a very long-running SMB session to that server – often days or more, depending on forced logoff policies.

Spotting Oddities Through UTF-16



- All SMB2+ traffic uses UTF-16 encoding
- “Follow TCP Stream” can help generate leads



While it is not possible to reliably use Wireshark’s “Follow TCP Stream” feature for file reconstruction due to the SMB overhead in each of the READ segments, it is often possible to find artifacts that warrant further investigation. Because SMB uses UTF-16 encoding for its protocol fields, printable characters are interspersed with NULL bytes (0x00), represented as the “.” character in the stream view.

In the first screenshot, you may quickly spot a Windows UNC string including an IP address. Depending on local practices, the apparent presence of “\\172.16.4.5\IPC\$”, a system management pseudo-share, may be concerning. If observed in conjunction with other suspicious traffic, it may be cause for outright alarm.

In the second screenshot, there appears to be a listing of files commonly found in the root directory of a Windows system’s C:\ drive. This may be the result of a QUERY_DIRECTORY message, or contents of a segment of a file being transferred. Further inspection would be needed to confirm this of course, but a quick visual lead such as this can be a great asset when time is of the essence during an investigation.

```

Wireshark · Follow TCP Stream (tcp.stream eq 0) · stark2-smb_172.16.6.11-172.16.4.5.pcap
NTLMSSP.....X.X.....X...
.l.....v.....
..7.....h.kn&w.A.*C.s.h.i.e.l.d.b.a.s.e.s.p.s.q.l.B.A.S.E.-R.D.-0.1.....W...
NXBI.Cw".....s!E.TLGD>.....s.h.i.e.l.d.b.a.s.e...B.A.S.E.-F.I.L.E.....s.h.i.e.l.d
.b.a.s.e...l.a.n...0.b.a.s.e.-f.i.l.e.s.h.i.e.l.d.b.a.s.e...l.a.n...s.h.i.e.l.d.b.a.s.e...l.a.n.
.s.IE.....0.0.....f.....0...../.....;n
.....c.i.f.s./1.7.2.1.6..4..5.....^..w.Ik.G.j.....I.....e.SMB@.....
.....]...u.e.....j.SMB@.....I.....H.....0.....I.....
$.H"\.1.7.2.1.6..4..5.I.P.C,
$.P.SMB@.....I.....Y..z..01.y.9.....0.....X..".....X.....SMB@.....
.....7.l.....I.....Y..z..01.y.9.....p.....p.....0.....u
6.0(K.#'.....Y..Sk.e.....I.....SMB@.....g.....
\A...9.....X.....x.....SMB@.....x.....W.k.s.s.v.c.....
SMB@.....I.....9.....I

```

```

Wireshark · Follow TCP Stream (tcp.stream eq 0) · stark2-smb_172.16.6.11-172.16.4.5.pcap
.....l.....~..
$.4.d8..4.d8..4.d8.....&.....K.....B.O.O.T.N.X.T.....
.M.s.i.....H.F...H.F...H.F.+q.....
$.S.e.t.t.i.n.g.s.....D.O.C.U.M.E.N.T.S.....K.....D.O.C.U.M.E.N.T.S..a.n.d.
.P.9...P.9...b.b/b/.....&
.O.CzA...m$.m$.&.....5.h.i.b.e.r.f.i.l..s.y.s.....p.a.g.e.f.i
.P.e.r.f.l.o.g.s.....X.0..X.0..Z.X$......).....
I.....P.r.o.g.r.a.m.....
.F.i.l.e.s.....e.<.....&.....
2.....P.r.o.g.r.a.m..F.i.l.e.s.
(.x.8.6.).....e.<;U..3.;U..3.;U..3.....
[4..n...P.R.O.G.R.A.-3.....P.r.o.g.r.a.m.D.a.t.a..x.....
[4.....q.....S.h.a.r.e.s.....
/vb0 /

```

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 in to 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.

Lab 3.3



SMB Session Analysis & Reconstruction

This page intentionally left blank.

Lab 3.3 Objectives: SMB Session Analysis & 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 data set in the GUI
 - Scale to larger data set with shell-based tools

This page intentionally left blank.



- SMB is a complex and comprehensive protocol
 - Hybrid Unicode/binary nature
 - Wireshark SMB parsers provide critical insight to these complex communications
- Backward compatibility may keep old/weak protocols in use far beyond their life span

This page intentionally left blank.



NetFlow and File Access Protocols

©2019 Lewes Technology Consulting, LLC | All Rights Reserved | Version # FOR572_E01_02

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

COURSE RESOURCES AND CONTACT INFORMATION



AUTHOR CONTACT

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



SANS INSTITUTE

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



DFIR RESOURCES

digital-forensics.sans.org
Twitter: @sansforensics



SANS EMAIL

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

This page intentionally left blank.

