SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

Network Forensics and Analysis Poster

Continuous Incident Response and Threat Hunting: Proactive Threat Identification

CORE CONCEPT:
Apply new intelligence to existing data to discover unknown incidents

NETWORK FORENSICS USE CASE:
Threat intelligence often contains network-based indicators such as IP addresses, domain names, signatures, URLs, and more. When these are known, existing data stores can be reviewed to determine if there were indications of the intel-informed activity that warrant further investigation.

Post-Incident Forensic Analysis: Reactive Detection and Response

CORE CONCEPT:
Examine existing data to more fully understand a known incident

NETWORK FORENSICS USE CASE:
Nearly every phase of an attack can include network activity. Understanding an attacker's actions during Reconnaissance, Delivery, Exploitation, Installation, Command and Control, and Post-Exploitation phases can provide deep and valuable insight into their actions, intent, and capability.

DFIR-Network_v1_4-17

Network Forensics is a critical component for most modern digital forensic, incident response, and threat hunting work. Whether pursued alone or as a supplement or driver to traditional endpoint investigations, network data can provide decisive insight into the human or automated communications within a compromised environment.

Network Forensic Analysis techniques can be used in a traditional forensic capacity as well as for continuous incident response/threat hunting operations.

Additional Resources

SANS FOR572: Advanced Network Forensics and Analysis:
`http://for572.com/course`

FOR572 Course Notebook:
`http://for572.com/notebook`

Network Forensics and Analysis Poster:
`http://for572.com/poster`

GIAC Certified Network Forensic Analyst certification available:
`http://for572.com/gnfa`

Network Source Data Types

Full-Packet Capture (pcap)

pcap files contain original packet data as seen at the collection point. They can contain partial or complete packet data.

Benefits

- Often considered the "holy grail" of network data collection, this data source facilitates deep analysis long after the communication has ended.
- Countless tools can read from and write to pcap files, giving the analyst many approaches to examine them and extract relevant information.

Drawbacks

- These files can grow extremely large – tens of terabytes of pcap data can be collected each day from a 1Gbps link. This scale often makes analysis challenging.
- Legal constraints often limit availability of this source data. Such constraints are also complicated when an organization crosses legal jurisdictions.
- Encrypted communications are increasingly used, rendering full-packet capture less useful for low-level analysis.

NetFlow and Related Flow-Based Collections

Flow records contain a summarization of network communications seen at the collection point. NetFlow contains no content – just a summary record including metadata about each network connection. Whether used alone to determine if communications occurred or in conjunction with other data sources, NetFlow can be extremely helpful for timely analysis.

Benefits

- NetFlow and similar records require much less storage space due to the lack of content. This facilitates much longer-term records retention.
- Analysis processes are much faster with NetFlow than full-packet capture. It can be 100-1000x faster to run a query against NetFlow than the corresponding pcap file.
- There are generally fewer privacy concerns with collecting and storing Netflow. Local legal authority should be consulted prior to use.
- Analysis processes apply equally to all protocols – encrypted or plaintext, custom or standards-based.

Drawbacks

- Without content, low-level analysis and findings may not be possible.
- Many collection platforms are unique and require training or licenses to access.

Log Files

Log files are perhaps the most widely-used source data for network and endpoint investigations. They contain application or platform-centric items of use to characterize activities handled or observed by the log creator.

Benefits

- Since they are collected and retained for business operations purposes, logs are widely available and processes often in place to analyze them.
- Raw log data can be aggregated for centralized analysis. Many organizations have this capability in some form of SIEM or related platform.

Drawbacks

- Log data contains varying levels of detail in numerous formats, often requiring parsing and enrichment to add context or additional data to corroborate findings.
- If log data is not already aggregated, finding it can involve significant time and effort before analysis can begin.

Network Source Data Collection Platforms

Switch

A port mirror is a "software tap" that duplicates packets sent to or from a designated switch port to another switch port. This is sometimes called a "SPAN port." The mirrored traffic can then be sent to a platform that performs collection or analysis, such as full-packet capture or a NetFlow probe.

Benefits

- Activating a port mirror generally requires just a configuration change, usually avoiding downtime.
- Switch presence at all levels of a typical network topology maximizes flexibility of capture/observation platform placement.

Drawbacks

- Data loss is possible with high-traffic networks, as bandwidth is limited to half-duplex speed.

Layer 2-7 Devices

Any platform with control of or purview over a network link can provide valuable logging data regarding the communications that pass through or by it. These may be network infrastructure devices like switches, routers, firewalls, and a variety of layer 7 devices such as web proxies, load balancers, DHCP and DNS servers, and more. Endpoints may also be configured to generate full-packet capture data or to export NetFlow.

Benefits

- Many perspectives on the same incident can yield multiple useful data points about an incident.

Drawbacks

- Log data may include numerous formats and varying levels of detail in their contents. This may require labor-intensive parsing and analysis to identify the useful details.
- Platforms that create the logs are often scattered across the enterprise – logically and physically. This requires a sound log aggregation plan and platform – or a lot of manual work.

Router

Routers generally provide NetFlow export functionality, enabling flow-based visibility with an appropriate collector.

Benefits

- Infrastructure is already in place, again just requiring a configuration modification and little to no downtime.
- Many organizations already collect NetFlow from their routing infrastructures, so adding an additional exporter is usually a straightforward process.

Drawbacks

- Routers don't generally provide the ability to perform full-packet capture.

Tap

A network tap is a hardware device that provides duplicated packet data streams that can be sent to a capture or observation platform to which it is connected. An "aggregating" tap merges both directions of network traffic to a single stream of data on a single port, while others provide two ports for the duplicated data streams – one in each direction. A "regenerating" tap provides the duplicated data stream(s) to multiple physical ports, allowing multiple capture or monitoring platforms to be connected.

Benefits

- Purpose-built to duplicate traffic – truly the best case for network traffic capture.
- Engineered for performance and reliability. Most taps will continue to pass mirrored traffic even without power, for example.

Drawbacks

- Can be very expensive, especially at higher network speeds and higher-end feature sets.
- Unless a tap is already in place at the point of interest, downtime is typically required to install one.

Distilling Full-Packet Capture Source Data

While full-packet capture is often collected strategically as a component of a continuous monitoring program or tactically during incident response actions, it is often too large to process natively. Instead, distill pcap files to other formats for more practical analysis. This offers the best of both worlds – fast analysis against the distilled source data, while retaining the original pcap file for in-depth analysis and extraction.

P Distill pcap file to F

NetFlow

"nfpcapd" utility from nfdump suite

- Permits quick Layer 3 – Layer 4 searching for network traffic in pcap file without parsing entire file
- `http://for572.com/nfdump`

```
$ nfpcapd -r infile.pcap -S 1 -z -l output_directory/
```

`-r infile.pcap`	pcap file to read
`-S 1`	Directory hashing structure for output data. "1" = "year/month/day/"
`-z`	Compress output files
`-l output_directory/`	Directory in which to place output files

P Distill pcap file to L

Bro NSM Logs

Bro network security monitoring platform

- Logs include numerous views of network traffic in a form that allows flexible queries and parsing in numerous platforms
- `http://for572.com/bro-nsm`

```
$ bro -r infile.pcap
```

`-r infile.pcap`	pcap file to read

P Distill pcap file to L

Passive DNS Logs

PassiveDNS lightweight DNS traffic logger

- Generates simplified log records detailing DNS queries and responses
- `http://for572.com/passivedns`

```
$ passivedns -r infile.pcap -l dnslog.txt -L nxdomain.txt
```

`-r infile.pcap`	pcap file to read
`-l dnslog.txt`	Output file containing log entries of DNS queries and responses
`-L nxdomain.txt`	Output file containing log entries of queries that generated NXDOMAIN responses

SOF-ELK

SOF-ELK is a VM appliance with a preconfigured, customized installation of the Elastic Stack. It was designed specifically to address the ever-growing volume of data involved in a typical investigation, as well as to support both threat hunting and security operations components of information security programs. The SOF-ELK customizations include numerous log parsers, enrichments, and related configurations that aim to make the platform a ready-to-use analysis appliance. The SOF-ELK platform is a free and open-source appliance, available for anyone to download. The configuration files are publicly available in a Github repository and the appliance is designed for upgrades in the field. The latest downloadable appliance details are at `http://for572.com/sof-elk-readme`.

What is "ELK" and the "Elastic Stack"?

The Elastic Stack consists of the Elasticsearch search and analytics engine, the Logstash data collection and enrichment platform, and the Kibana visualization layer. It is commonly known as "ELK", named for these three components.

The broader Elastic Stack includes other components such as the Elastic Beats family of log shippers, and various security and performance monitoring components.

All of the ELK components and the Beats log shippers are free and open-source software. Some other components of the Elastic Stack are commercially-licensed.

Booting and Logging into SOF-ELK

The SOF-ELK VM is distributed in ready-to-boot mode. You may want to add additional CPU cores and RAM allocation if available.

The VM's IP address is displayed after it boots, on the pre-authentication screen. This IP address is needed for both remote shell access (SSH) and web access to the Kibana interface.

The user name is "`elk_user`" and the default password is "`forensics`" for both this and the "`root`" users. Passwords for both the "`elk_user`" and "`root`" accounts should be changed immediately upon first boot.

The SSH server is running on the default port, 22. Access this with your preferred SSH/SCP/SFTP client software.

The Kibana interface is running on port 5601. Access this with your preferred web browser.

Lucene Query Syntax

The Elastic Stack uses the Apache Lucene query syntax for searching its data. Below are some of the basic syntaxes that will help you to search data that has been loaded to SOF-ELK.

For further information, an online tutorial is available at the following page: `http://for572.com/lucene`

Basic Searching

The most basic search syntax is "**`fieldname:value`**", which will match all documents with a "`fieldname`" field set to a value of "`value`". Searches can be negated by prefixing them with a "`-`" character. Some examples:

- `source_ip:192.168.25.0`
- `hostname:webserver`
- `-querytype:AAAA`

Partial String Searches

The "`*`" is used as a wildcard character.

- `username:*admin*`
- `query:*.cz.cc`

Numerical and IP Address Ranges

The "`[`" and "`]`" characters denote inclusive range boundaries (i.e. greater or equal to, less than or equal to) and the "`{`" and "`}`" character denote exclusive range boundaries (i.e. greater than, less than). Note that the "`TO`" must be capitalized.

- `ip:[10.58.3.0 TO 10.58.3.255]`
- `rrcount:{5 TO 20]`

Logical Construction

Multiple searches can be combined using "`AND`" and "`OR`", which must also be capitalized.

- `destination_geo.asn:Amazon.com AND in_bytes:[1000000 TO 100000000]`
- `aprotocol:tcp OR aprotocol:udp`

Loading Data to SOF-ELK

SOF-ELK can ingest several data formats, including:

- Syslog (many different log formats supported)
- HTTP server access logs
- NetFlow
- Bro logs

More sources are being tested and added to the platform and can be activated through the Github repository. See the "Updating With Git" section for more details on how to do this.

All source data can be loaded from existing files (DFIR Model) as well as from live sources (Security Operations Model).

DFIR Model

Place source data onto the SOF-ELK VM's filesystem in the appropriate location:

Syslog data: `/logstash/syslog/`
Since syslog entries often do not include the year, subdirectories for each year can be created in this location – for example, `/logstash/syslog/2016/`

HTTP server logs: `/logstash/httpd/`
Supports common, combined, and related formats

PassiveDNS logs: `/logstash/passivedns/`
Raw logs from the `passivedns` utility

NetFlow from nfcapd-collected data stores:
`/logstash/nfarch/`
Use the `nfdump2sof-elk.sh` script to create compatible ASCII format data (Script included on the SOF-ELK VM and available from the Github repository)

Bro NSM logs: `/logstash/bro/`
Supports multiple different log types, based on default Bro NSM filenames

Security Operations Model

Open the necessary firewall port(s) to allow your preferred network-based ingest to occur.

Syslog: TCP and UDP syslog protocol

```
$ sudo fw_modify.sh -a open -p 5514 -r tcp
$ sudo fw_modify.sh -a open -p 5514 -r udp
```

Syslog: Reliable Event Logging Protocol (RELP)

```
$ sudo fw_modify.sh -a open -p 5516 -r tcp
```

Syslog: Elastic Filebeat shipper

```
$ sudo fw_modify.sh -a open -p 5044 -r tcp
```

NetFlow: NetFlow v5 protocol

```
$ sudo fw_modify.sh -a open -p 9995 -r udp
```

HTTP Server logs: TCP and UDP syslog protocol

```
$ sudo fw_modify.sh -a open -p 5515 -r tcp
$ sudo fw_modify.sh -a open -p 5515 -r udp
```

HTTP Server logs: RELP

```
$ sudo fw_modify.sh -a open -p 5517 -r tcp
```

Configure the log shipper or source to send data to the port indicated above.

Clearing and Re-Parsing Data

Removing data from SOF-ELK's Elasticsearch indices as well as forcing the platform to re-parse source data on the filesystem itself have both been automated with a shell script. Removal is done by index, and optionally allows a single source file to be removed. The index name is required.

Get a list of currently-loaded indices:

```
$ sof-elk_clear.py -i list
```

Remove all data from the `netflow` index:

```
$ sof-elk_clear.py -i netflow
```

Remove all data from the `syslog` index and reload all source data:

```
$ sof-elk_clear.py -i syslog -r
```

Remove all data from the `httpdlog` index, but only documents originally loaded from the `/logstash/httpdlog/access_log` file:

```
$ sof-elk_clear.py -i httpdlog ⏎
  -f /logstash/httpdlog/access_log
```

Updating With Git

The SOF-ELK VM uses a clone of the Github-based repository containing all configuration files. This allows the user to update an operational install's configuration files without needing to download a new copy of the VM itself. ALWAYS check the current Github repository for any notes or special instructions before updating an operational SOF-ELK platform.

To update the VM, ensure it has Internet connectivity and run the following command:

```
$ sudo sof-elk_update.sh
```

SOF-ELK Dashboards

Several Kibana dashboards are provided, each designed to address basic analysis requirements. Open the Kibana interface in a web browser using the SOF-ELK VM's IP address on port 5601.

The following dashboards are included:

- SOF-ELK VM Introduction Dashboard
- Syslog Dashboard
- HTTPD Log Dashboard
- NetFlow Dashboard

Additional dashboards will be distributed through the Github repository. (See the "Updating With Git" section.)

The Kibana dashboards allow the analyst to interact with and explore the data contained in the underlying Elasticsearch engine. Several features provide a level of interactivity that allows dynamic analysis across vast volumes of data.

Querying Available Data

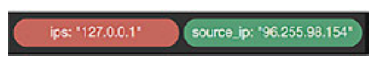
The top of each dashboard allows the user to input Lucene queries, detailed in the "Lucene Query Syntax" section. Elasticsearch determines how well its documents match, including a "`_score`" field that indicates how well each document matches the query.

NetFlow Dashboard | destination_geo.asn:Inc

Filtering

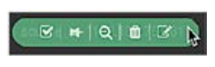
Filters can also be applied in the Kibana interface. These are similar to queries, but are a binary match/non-match search without a "`_score`" field. Elasticsearch caches frequently-used filters to optimize their performance.

Kibana shows filters as bubbles below the query field. Green bubbles indicate positive match filters, red bubbles indicate negative match filters.



ips: "127.0.0.1"
source_ip: "96.255.98.154"

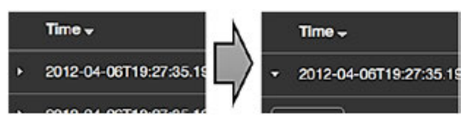
Filters can be modified with the menu that appears after hovering over the filter bubble.



From left to right, these options are: toggle filter on or off, pin filter to all dashboards, negate filter, delete filter, and manually edit filter.

Document Expansion

When a dashboard includes a document listing panel, each document can be expanded by clicking the triangle icon on the left.



This will show all fields for the document.

t _type bro_c
t aprotocol tcp
t beat.hostname sof-el

Interactive Filter Generation

Each field displayed in the record details can be interactively built into a filter with the magnifying glass icons. The plus sign creates a positive filter, the minus sign creates a negative filter. The table icon adds the field to the document listing panel.

Network-Based Processing Workflows

Although there is no single workflow to exhaustively perform network forensic analysis, the most common and beneficial tasks can generally be placed into the categories below. Note that these categories are not generally iterative. They are components of a dynamic process that can adapt to adversaries' actions.

Ingest and Distill

GOAL: Prepare for analysis and derive data that will more easily facilitate the rest of the analytic workflow

- Log source data according to local procedure
- If pcap files are available, distill to other data source types (NetFlow, Bro logs, Passive DNS logs, etc.)
- Consider splitting source data into time-based chunks if the original source covers an extended period of time
- Load source data to large-scale analytic platforms such as SOF-ELK, Moloch, etc.

Reduce and Filter

GOAL: Reduce large input data volume to a smaller volume, allowing analysis with a wider range of tools

- Reduce source data to a more manageable volume using known indicators and data points
- Initial indicators and data points may include IP addresses, ports/protocols, time frames, volume calculations, domain names and hostnames, etc.
- For large-scale analytic platforms, build filters to reduce visible data to traffic involving known indicators

Analyze and Explore

GOAL: Identify traffic and artifacts that support investigative goals and hypotheses

- Within the reduced data set, seek knowledge about the suspicious traffic
- This may include evaluating traffic contents, context, anomalies, consistencies – anything that helps to clarify its relevance to the investigation
- Seek any protocol anomalies that could indicate traffic being misused for suspicious purposes
- Use any available environmental baselines to identify deviations from normal traffic behaviors

Extract Indicators and Objects

GOAL: Find artifacts that help identify malicious activity, including field values, byte sequences, files, or other objects

- As additional artifacts are identified, maintain an ongoing collection of these data points for further use during and after the investigation
- These may include direct observations from within the network traffic or ancillary observations about the nature of the communications – related DNS activity, before/after events, etc.
- Extracting files and other objects such as certificates or payloads can help feed other parts of the IR process such as malware reverse engineering and host-based activity searches
- Protect this data according to local policies and share in accordance with appropriate operational security constraints

Scope and Scale

GOAL: Search more broadly within source data for behavior that matches known indicators

- After identifying useful artifacts that define activity of interest, scale up the search using large-scale analytic platforms and tools
- Identify additional endpoints that exhibit the suspicious behavior, aiming to fully scope the incident within the environment
- Pass appropriate indicators to security operations for live identification of suspicious activity

Establish Baselines

GOAL: Identify parameters for "normal" patterns of behavior to help find anomalies that need to be investigated

- Determine typical cycles of traffic, top-talking hosts, ports/protocols, GET vs POST ratio for HTTP activity, etc.
- Build all baselines for multiple periods – most metrics have different cycles for daily, weekly, monthly, and annual time frames
- Consider the levels within the organization at which the baselines should be built – enterprise-level rollups will generally differ from those at lower levels

Bro NSM Log Files

The Bro Network Security Monitoring platform produces numerous log files containing useful artifacts extracted from the source pcap data. These logs are in ASCII format, but generally require the "`bro-cut`" utility for more streamlined analysis. Note that not all log files will be created – Bro only generates log files that pertain to source traffic it has parsed. This is not an exhaustive list of all logs created – see `http://for572.com/bro-logs` for more log types.

Network Protocols

`conn.log`
- TCP/UDP/ICMP connections
- A NetFlow-like view of traffic

`dns.log`
- DNS artifacts, including queries and responses
- A form of passive DNS logs in the Bro format

`http.log`
- HTTP artifacts, including URLs, User-Agents, Referrers, MIME types, and many others

`rdp.log`
- Remote Desktop Protocol artifacts

`smtp.log`
- SMTP (email sending and relaying) artifacts

File Metadata

`files.log`
- File metadata such as hash, MIME type, and more for all files observed, via any protocol

`x509.log`
- Certificate metadata for SSL and TLS connections

Special Cases

`signatures.log`
- Events that match content signatures Bro has been directed to search for
- Not a replacement for an IDS, but often useful for targeted searching

`weird.log`
- Protocol anomalies that Bro did not expect
- Includes events such as unrequested DNS responses, TCP truncations, etc.

Inventory

`known_hosts.log`
- A list of IP client addresses that have been observed completing at least one TCP handshake

`known_services.log`
- List of server IP addresses and ports that have been observed providing at least one TCP handshake, including the protocol (if available)

`software.log`
- List of software identified operating within the source data
- Generally extracted from server banners or client fields such as the HTTP User-Agent

PassiveDNS Log Format

The following entries are part of the results for a DNS query/response for the "`www.reddit.com`" hostname:

```
1456702040.919984||192.168.75.6||192.168.75.1||IN||www.reddit.com.||A||198.41.208.136||297||11
1456702040.919984||192.168.75.6||192.168.75.1||IN||www.reddit.com.||A||198.41.208.140||297||11
1456702040.919984||192.168.75.6||192.168.75.1||IN||www.reddit.com.||A||198.41.209.142||297||11
1456702040.919984||192.168.75.6||192.168.75.1||IN||www.reddit.com.||A||198.41.209.140||297||11
1456702040.919984||192.168.75.6||192.168.75.1||IN||www.reddit.com.||A||198.41.209.137||297||11
```

The lightweight "`passivedns`" utility creates ASCII records that detail DNS queries and responses. This format is ideal for searching for activity across multiple protocols, as most software (good or evil) makes DNS requests before initiating a network connection. These logs can also be easily parsed by a SIEM or log aggregator such as SOF-ELK.

Each entry consists of the following fields:

`1456702040.919984`	UNIX timestamp + microseconds	`A`	Record type
`192.168.75.6`	Client IP address	`198.41.209.137`	Answer received (>1 gives multiple rows)
`192.168.75.1`	Server IP address		
`IN`	Class (IN = "INTERNET" class)	`297`	TTL value (seconds to cache)
`www.reddit.com`	Name requested	`11`	Total record count

This poster was created by SANS Instructor Phil Hagen with support from SANS DFIR Faculty

tcpdump: Log or parse network traffic

Classically used to dump live network traffic to pcap files, tcpdump is more commonly used in network forensics to perform data reduction by reading from an existing pcap file, applying a filter, then writing the reduced data to a new pcap file. tcpdump uses the BPF (Berkeley Packet Filter) language for packet selection.

Usage:

```
$ tcpdump <options> <bpf filter>
```

Common command-line parameters:

-n Prevent DNS lookups on IP addresses. Use twice to also prevent port-to-service lookups
-r Read from specified pcap file instead of the network
-w Write packet data to a file
-i Specify the network interface on which to capture
-s Number of bytes per packet to capture
-C Number of megabytes to save in a capture file before starting a new file
-G Number of seconds to save in each capture file (requires time format in output filename)
-W Used with the -C or -G options, limit the number of rotated files

Note: The BPF filter is an optional parameter

Common BPF primitives:

host	IP address or FQDN	tcp	Layer 4 protocol is TCP
net	Netblock in CIDR notation	udp	Layer 4 protocol is UDP
port	TCP or UDP port number	icmp	Layer 4 protocol is ICMP
ip	Layer 3 protocol is IP		

Parameters such as host, net, and port can be applied in just one direction with the src or dst modifiers. Primitives can be combined with and, or, or not, and order can be enforced with parentheses.

BPF Examples:

- tcp and port 80
- udp and dst host 8.8.8.8
- src host 1.2.3.4 and (dst net 10.0.0.0/8 or dst net 172.16.0.0/12)

Capturing live traffic generally requires elevated operating system permissions (e.g. sudo), but reading from existing pcap files only requires filesystem-level read permissions to the source file itself.

Examples:

```
$ tcpdump -n -r infile.pcap ⏎
  -w tcp80.pcap 'tcp port 80'
$ sudo tcpdump -n -i enp0s3 -w outfile.pcap
$ sudo tcpdump -n -i enp0s3 -C 1024 ⏎
  -G 100 -w 10GB_rolling_buffer.pcap
$ sudo tcpdump -n -i enp0s8 -G 86400 ⏎
  -w dns-%F.%T.pcap
```

Wireshark: Deep, protocol-aware packet exploration and analysis

Wireshark is perhaps the most widely known packet data exploration tool. It provides extensive protocol coverage and low-level data exploration features. Its included protocol parsers number over 1,500 and extract over 140,000 different data fields. Wireshark parsers often normalize the content in these fields for readability. (DNS hostnames, for example, are presented in FQDN form rather than literal strings as they appear in the packet.)

Wireshark display filters:

Wireshark provides rich and extensive display filtering functionality based on the fields identified by protocol decoders. Any of the 140,000+ fields can be evaluated in a display filter statement.

Basic filters use the following syntax:

- fieldname == value
- fieldname < value
- fieldname >= value

Note: Avoid using the != operator, as it can produce unintended results with fields that occur more than once in a single packet.

Complex display filters can be built with the && and || logical conjunctions, and parenthesis to enforce order of operations.

Display filter resources:

See the wireshark-filter man page for more command-line details on how to construct display filters.

mergecap: Merge two or more pcap files

When faced with a large number of pcap files, it may be advantageous to merge a subset of them to a single file for more streamlined processing. This utility will ensure the packets written to the output file are chronological.

Usage:

```
$ mergecap <options> -w <output file> ⏎
  <input file 1> <input file 2> ⏎
  <input file n>
```

Common command-line parameters:

-w New pcap file to create, containing merged data
-s Number of bytes per packet to retain

Example:

```
$ mergecap -w new.pcap infile1.pcap ⏎
  infile2.pcap
```

editcap: Modify contents of a capture file

Since the BPF is limited to evaluating packet content data, a different utility is required to filter on pcap metadata. This command will read capture files, limit the time frame, file size, and other parameters, then write the resulting data to a new capture file, optionally de-duplicating packet data.

Usage:

```
$ editcap <options> <input file> ⏎
  <output file>
```

Common command-line parameters:

-A Select packets at or after the specified time (Use format: YYYY-MM-DD HH:MM:SS)
-B Select packets before the specified time
-d De-duplicate packets (Can also use -D or -w for more fine-grained control)
-c Maximum number of packets per output file
-i Maximum number of seconds per output file (Note that the -c and -i flags cause multiple files to be created, each named with an incrementing integer and initial timestamp for each file's content, e.g. output_00000_20170417174516.pcap)

Examples:

```
$ editcap -A '2017-01-16 00:00:00' ⏎
  -B '2017-02-16 00:00:00' ⏎
  infile.pcap 2017-jan-16.pcap
$ editcap -d infile.pcap dedupe.pcap
$ editcap -i 3600 infile.pcap hourly.pcap
```

tshark: Command-line access to nearly all Wireshark features

For all of Wireshark's features, the ability to access them from the command line provides scalable power to the analyst. Whether building repeatable commands into a script, looping over dozens of input files, or performing analysis directly within the shell, tshark packs nearly all of Wireshark's features in a command-line utility.

Usage:

```
$ tshark -n -r <input file> <options> ⏎
  -Y '<display filter>'
```

Common command-line parameters:

-n Prevent DNS lookups on IP addresses
-r Read from specified pcap file
-w Write packet data to a file
-Y Specify Wireshark-compatible display filter
-T Specify output mode (fields, text (default), pdml, etc.)
-e When used with -T fields, specifies a field to include in output tab-separated values (can be used multiple times)
-G Specify glossary to display (protocols, fields, etc.) – shows available capabilities via command line, suitable for grep'ing, etc.

Display filter resources:

See the wireshark-filter man page for more command-line details on how to construct display filters.

Examples:

```
$ tshark -n -r infile.pcap ⏎
   -Y 'http.host contains "google"' ⏎
   -T fields -e ip.src -e http.host ⏎
   -e http.user_agent
$ tshark -n -r infile.pcap ⏎
   -Y 'ssl.handshake.certificates' ⏎
   -w just_certificates.pcap
```

tcpxtract: Carve reassembled TCP streams for known header and footer bytes to attempt file reassembly

This is the TCP equivalent to the venerable foremost and scalpel disk/memory carving utilities. tcpxtract will reassemble each TCP stream, then search for known start/end bytes in the stream, writing out matching sub-streams to disk. It is not protocol-aware, so it cannot determine metadata such as filenames and cannot handle protocol content consisting of non-contiguous byte sequences. Notably, tcpxtract cannot parse SMB traffic, encrypted payload content, or chunked-encoded HTTP traffic. Parsing compressed data requires signatures for the compressed bytes rather than the corresponding plaintext.

Usage:

```
$ tcpxtract -r <input file> <options>
```

Common command-line parameters:

-f Read from specified pcap file
-c Configuration (signature) file to use
-o Place output files into specified directory

Signature format:

- file_ext(max_size, start_bytes, end_bytes);

Signature examples:

- gif(3000000, \x47\x49\x46\x38\x37\x61, \x00\x3b);
- rpm(400000000, \xed\xab\xee\xdb);

Example:

- tcpxtract -f infile.pcap ⏎ -c rpm-tcpxtract.conf -o ./

Network Forensic Toolbox

Tools are a critical part of any forensic process, but they alone cannot solve problems or generate findings. The analyst must understand the available tools and their strengths and weaknesses, then asses the best approach between raw source data and the investigative goals at hand. The tools detailed here are far from a comprehensive list, but represent a core set of utilities often used in network forensic analysis. More extensive documentation is available in the tools' man pages and online documentation.

grep: Display lines from input text that match a specified regular expression pattern

Searches input text from a file or via STDIN pipes using extremely flexible and age-old regular expressions. Matching lines are displayed, but output can be fine-grained to address specific analytic requirements.

Usage:

```
$ grep <options> <pattern> <input file>
```

Common command-line parameters:

-i Case-insensitive search
-r Recursively process all files within a directory tree
-a Fully search all files as ASCII, even if they appear to contain binary data
-l Only display file names that contain matches instead of the lines on which the match is found
-F Disable the regular expression engine, providing a significant speed benefit
-c Display count of matching lines
-A Display a number of lines before each line that matches the search pattern
-B Display a number of lines after each line that matches the search pattern
-C Display a number of context lines before and after each line that matches the search pattern
-H Display filenames in addition to matching line contents – this is the default with -r
-h Omit filenames from output as displayed with -r
-v Invert match – only show results that do not match the search pattern – with -l, show files' names in which there is at least one line not matching the search pattern – with -c, show count of non-matching lines

Regular expressions are a dark art of shell commands.

Examples:

```
$ grep pastebin access.log
$ grep -rail google /var/spool/squid/
$ grep -Fv 192.168.75. syslog-messages
$ grep -C 5 utmscr error.log
```

NetworkMiner: Protocol-aware object extraction tool that writes files to disk

Object extraction is often a tedious task, but NetworkMiner reliably performs this function for a number of common protocols. File objects are written to disk as they are encountered, while fields (credentials, hosts, etc.) can be exported to CSV format.

Writing files to disk often triggers host-based defenses, so running this utility in an isolated and controlled environment is the most common use model.

NetworkMiner is a commercial utility that also provides a free version. The free version is licensed for operational use, not just testing.

bro-cut: Extract specific fields from Bro logs

The Bro NSM creates log files as needed to document observed network traffic. These are in tab-separated-value format, but require postprocessing to extract just the fields of interest.

Usage:

```
$ cat <log file> | bro-cut <options> <fields>
```

Common command-line parameters:

-u Convert timestamp to human-readable, UTC format
-c Display header blocks at start of output

Identifying fields of interest:

Each different log file type contains various fields, detailed in the header of the file. Inspect the first few lines and identify the one that begins with the string #fields. The remainder of this line contains the Bro-specific names for each column of data, which can be extracted with the bro-cut utility. Consult the Bro NSM documentation for details on each column's meaning.

Examples:

```
$ cat files.log | bro-cut -u ts ⏎
  fuid tx_hosts sha256
$ zcat http*.gz | bro-cut -u ts id.orig_h ⏎
  host uri user_agent info_code
```

capinfos: Calculate and display high-level summary statistics for an input pcap file

This utility displays summary metadata from one or more source pcap files. Reported metadata includes but is not limited to start/end times, hash values, packet count, and byte count.

Usage:

```
$ capinfos <options> <input file 1> ⏎
  <input file 2> <...>
```

Common command-line parameters:

-A Generate all available statistics
-T Use "table" output format instead of list format

Examples:

```
$ capinfos -A infile.pcap
$ capinfos -A -T infile2.pcap
$ capinfos -A *.pcap
```

ngrep: Display metadata and context from packets that match a specified regular expression pattern

While grep is a very capable tool for ASCII input, it does not understand the pcap file format. ngrep performs the same function but against the Layer 4 – Layer 7 payload in each individual packet. It does not perform any TCP session reassembly, so matches are made against individual packets only.

Usage:

```
$ ngrep -I <input file> <options> ⏎
  <pattern> <bpf filter>
```

Common command-line parameters:

-I Read from specified pcap file
-O Write matching packets to specified pcap file
-i Case-insensitive search
-v Invert match – only show packets that do not match the search pattern
-t Show timestamp from each matching packet

Note: The BPF filter is an optional parameter

Examples:

```
$ ngrep -I infile.pcap 'RETR' 'tcp and port 21'
$ ngrep -I infile.pcap -i 'l33tAUTH'
```

tcpflow: Reassemble input packet data to TCP data segments

This utility will perform TCP reassembly, then output each side of the TCP data flows to separate files. This is essentially a scalable, command-line equivalent to Wireshark's "Follow | TCP Stream" feature. Additionally, tcpflow can perform a variety of decoding and post-processing functions on the resulting flows.

Usage:

```
$ tcpflow <options> -r <input file> ⏎
  -o <output path>
```

Common command-line parameters:

-r Read from specified pcap file (can be used multiple times for multiple files)
-l Read from multiple pcap files (with wildcards)
-o Place output files into specified directory

Examples:

```
$ tcpflow -r infile.pcap  -o /tmp/output/
$ tcpflow -l *.pcap -o /tmp/output/
```

flowgrep.py: Extract TCP streams that match a specified regular expression pattern

While ngrep only searches within a single packet for its search patterns, flowgrep.py reconstructs TCP sessions first, then searches the resulting streams for matches. The reassembled data streams that match the search pattern are written to disk.

Usage:

```
$ sudo flowgrep.py <options> ⏎
  <bpf filter>
```

Common command-line parameters:

-r Read from specified pcap file
-c Specify pattern to search for in client-to-server side
-s Specify pattern to search for in server-to-client side
-a Specify pattern to search for in either side
-i Case-insensitive search
-v Invert match - only write streams that do not match the search pattern
-l Directory in which to place the reconstructed payloads of matched streams

Notes: flowgrep.py requires root access because it changes its effective userid to nobody at runtime

The BPF filter is an optional parameter

Examples:

```
$ sudo flowgrep.py -r infile.pcap ⏎
   -c 'STARTTLS'
$ sudo flowgrep.py -r infile.pcap -i ⏎
   -a '706173736D656F77' -l ./output/
```

nfdump: Process NetFlow data from nfcapd-compatible files on disk

Files created by nfcapd (live collector) or nfpcapd (pcap-to-NetFlow distillation) are read, parsed, and displayed by nfdump. Filters include numerous observed and calculated fields, and outputs can be customized to unique analysis requirements.

Usage:

```
$ nfdump (-R <input directory path> |
          -r <nfcapd file> ) ⏎
  <options> <filter>
```

Common command-line parameters:

-r Read from the specified single file
-R Recursively read from the specified directory tree
-t Specify time window in which to search (Use format: YYYY/MM/DD.hh:mm:ss-YYYY/MM/DD.hh:mm:ss)
-o Output format to use (line, long, extended, or custom with fmt:<format string>)
-O Output sort ordering (tstart, bytes, packets, more)
-a Aggregate output on source IP+port, destination IP+port, layer 4 protocol
-A Comma-separated custom aggregation fields

Filter syntax:

host IP address or FQDN
net Netblock in CIDR notation
proto Layer 4 protocol (tcp, udp, icmp, etc)
as Autonomous System number

Parameters such as host, net, and port can be applied in just one direction with the src or dst modifiers. Primitives can be combined with and, or, or not, and order can be enforced with parenthesis.

Filter examples:

- proto tcp and port 80
- proto udp and dst host 8.8.8.8
- src host 1.2.3.4 and (dst net 10.0.0.0/8 or dst net 172.16.0.0/12)
- src as 32625 (Note: Not all collections include ASNs)

Custom output formatting:

Format strings for the custom output format option (-o 'fmt:<format string>')consist of format tags, including but not limited to those below.

%ts	Start time	%sa	Source IP address
%te	End time	%da	Destination IP address
%td	Duration (In seconds)	%sp	Source port (TCP or UDP)
%pr	Layer 4 protocol		
%dp	Destination port (TCP or UDP; formatted as type.code for ICMP)		
%sap	Source IP address and port		
%dap	Destination IP address and port		
%pkt	Packet count		
%byt	Byte count		
%flg	TCP flags (sum total for flow)		
%bps	Bits per second (average)		
%pps	Packets per second (average)		
%bpp	Bytes per packet (average)		

Custom aggregation:

Records displayed can be aggregated (tallied) on user-specified fields including but not limited to those below:

proto	Layer 4 protocol
srcip	Source IP address
dstip	Destination IP address
srcport	TCP or UDP source port
dstport	TCP or UDP destination port
srcnet	Source netblock in CIDR notation
dstnet	Destination netblock in CIDR notation

Examples:

```
$ nfdump -r nfcapd.201703271745 ⏎
  -o long 'proto tcp and port 53'
$ nfdump -R /var/log/netflow/2017/03/ ⏎
  -o 'fmt:%sa %da %pr' -A srcip,dstip,proto
  'dst net 66.35.59.0/24'
$ nfdump -R /var/log/netflow/2016/ ⏎
  -O tstart 'proto tcp and port 4444'
```

calamaris: Generate summary reports from web proxy server log files

The calamaris utility performs high-level summary analysis of many different formats of web proxy log files. These reports are broken down by HTTP request methods, second-level domains, client IP addresses, HTTP response codes, and more.

Usage:

```
cat <input file> | calamaris <options>
```

Common command-line parameter:

-a Generate all available reports

Examples:

```
$ cat access.log | calamaris -a
$ zgrep 1.2.3.4 access.log.gz | calamaris -a
$ grep badhost.cc.cz | calamaris -a
```

Network Traffic Anomalies

Knowing what is "normal" in any environment is critical in order to quickly determine outlier events that may suggest suspicious or malicious activity. In the world of network protocols, this can be a significant challenge. There are countless ways network traffic can be manipulated to the attacker's advantage while still appearing to be normal. In many cases, these deviations still follow all the rules of the carrier protocol. The conditions presented here can be useful in identifying anomalies, but this is not an exhaustive list. They may be useful in establishing or boosting a baselining program or for providing a healthy dose of skepticism during an investigation.

HTTP GET vs POST Ratio

How: HTTP proxy logs, NSM logs, HTTP server logs

What: The proportion of observed HTTP requests that use the GET, POST or other methods.

Why: This ratio establishes a typical activity profile for HTTP traffic. When it skews too far from the normal baseline, it may suggest brute force logins, SQL injection attempts, RAT usage, server feature probing, or other suspicious/malicious activity.

Top-Talking IP Addresses

How: NetFlow

What: The list of hosts responsible for the highest volume of network communications in volume and/or connection count. Calculate this on a rolling daily/weekly/monthly/annual basis to account for periodic shifts in traffic patterns.

Why: Unusually large spikes in traffic may suggest exfiltration activity, while spikes in connection attempts may suggest C2 activity.

HTTP User-Agent

How: HTTP proxy logs, NSM logs, HTTP server logs

What: The HTTP User-Agent generally identifies the software responsible for issuing an HTTP request. This can be useful to profile software operating within the environment.

Why: This is an invaluable identifier to profile activity within the environment. It can profile which web browser titles, versions, and extensions are in use. More recently, desktop and mobile applications use unique User-Agent strings as well. Knowing the "normal" strings present causes outliers to stand out, which may highlight suspicious activity. However, this is an arbitrary and optional header, so be skeptical of behavior that suggests forgery – such as rapid change for a given IP address, significant increase in the number of observed User-Agent strings, etc.

Top DNS Domains Queried

How: Passive DNS logs, DNS server-side query logs, NSM logs

What: The most frequently queried second-level domains (e.g. "example.com" or "example.co.uk") based on internal clients' request activity. The top 1000 domains on a rolling daily basis may be a good starting point, but this number should be adjusted to local requirements.

Why: In general, the behaviors of a given environment don't drastically change on a day-to-day basis. Therefore, the top 500-700 domains queried on any given day should not differ too much from the top 1000 from the previous day. (The difference in count allows for natural ebb and flow of daily behavior.) Any domain that rockets to the top of the list may suggest an event that requires attention, such as a new phishing campaign, C2 domain, or other anomaly.

HTTP Return Code Ratio

How: HTTP Proxy logs, NSM logs, HTTP server logs

What: The return code is a three-digit integer that helps to indicate "what happened" on the server answering a request. These are grouped into "families" by hundreds: 100s = informational, 200s = success, 300s = redirection, 400s = client-side error, 500s = server-side error.

Why: Knowing what happened at the server end of the transaction can be extremely useful in characterizing HTTP activity. A spike in 400-series codes could indicate reconnaissance or scanning activity, while an unusually high number of 500-series codes could indicate failed login or SQL injection attempts. As with other observations, knowing the typically-observed ratios of the these values can help to identify anomalous trends that require further investigation.

Newly-Observed/Newly-Registered Domains

How: Passive DNS logs, DNS server-side query logs, NSM logs

What: Any domain that has never previously been queried from within the environment, according to the historical domain query logs, or the age of a domain, according to its WHOIS "Date Registered."

Why: The first time a domain is queried in a given environment may indicate a new or highly-focused targeting operation. Brand new domains are often associated with malicious activity, given that attackers generally require a dynamic infrastructure for their operations.

External Infrastructure Usage Attempts

How: NetFlow, Firewall logs, NSM logs

What: Although best practice is to restrict outbound communications by default and approve necessary services and connections by exception, this is often not the case – perimeters are still notoriously porous in the outbound direction. Even in a properly-constrained environment, these attempts should create artifacts of the failed connection attempts.

Why: By identifying internal clients that attempt to or succeed in using external services, it is possible to quickly collect a list of endpoints that exhibit anomalous behavior. These may include connections to external DNS servers rather than internal resolvers, HTTP connection attempts that seek to bypass proxy servers, connections to VPN providers, raw socket connections to unusual ports, and more.

Typical Port and Protocol Usage

How: NetFlow

What: The list of ports and corresponding protocols that account for the most communication in terms of volume and/or connection count. Calculate this on a daily/weekly/monthly/annual basis to account for periodic shifts in traffic patterns.

Why: Similar to the purpose for tracking top-talking IP addresses, knowing the typical port and protocol usage enables quick identification of anomalies that should be further explored for potential suspicious activity.

DNS TTL Values and RR Counts

How: Passive DNS logs, NSM logs

What: TTL refers to the number of seconds that a caching DNS server should retain a given record. The number of Resource Records in a given DNS packet is noted in the RR count field.

Why: Very short TTLs may suggest fast-flux DNS or potential tunneling behavior. A high RR count could indicate large-scale load balancing associated with fast-flux or similar elastic architectures. While these behaviors can suggest suspicious behavior, they are also commonly seen with benign network activity such as content delivery networks, round robin DNS-based load balancing, and similar architectures.

Autonomous System Communications

How: NetFlow, NSM logs

What: Autonomous System Numbers (ASNs) are numerical 'handles' assigned to netblock owners such as ISPs, datacenters, and other service providers. These can suggest Internet "neighborhoods" to characterize network traffic based on more than IP address or CIDR blocks.

Why: Certain ASNs are often more prominently associated with malicious activity than others. Reputation databases can be useful in determining these. Even without an intelligence overlay, identifying the ASNs with which systems in the environment communicate is a useful baseline metric that can easily identify communications with unusual ASNs that require further attention.

Periodic Traffic Volume Metrics

How: NetFlow

What: Maintaining traffic metrics on time-of-day, day-of-week, day-of-month and similar bases.

Why: These will identify normative traffic patterns, making deviations easier to spot and investigate. A sudden spike of traffic or connections during an overnight or weekend period when there is typically little or no traffic would be a clear anomaly of concern.

Moloch

Moloch is a full-packet ingestion and indexing platform. It reads a live network data stream of existing pcap files, then extracts data from known protocol fields to store in an Elasticsearch backend. Moloch calls these fields Session Protocol Information, or SPI data. Moloch uses a session-centric view, associating both the client- and server-sourced directions of a communication stream for easy analysis. Moloch separates full-packet data and SPI data, allowing different storage allocation and retention policies. The user can also export a subset of traffic in pcap format, making it a valuable addition to the network forensic workflow since any other capable tool can be used on the derived data.

Loading Data to Moloch

Moloch can load network traffic from existing pcap files (DFIR Model) or a live network interface (Security Operations Model).

DFIR Model

Place pcap files into Moloch's "raw" directory, often /data/moloch/raw/. Ensure the Moloch user (typically "nobody") has read permissions to the file(s).

Load the data with the following command:

```
$ moloch-capture -r /data/moloch/raw/infile.pcap
```

Security Operations Model

(Note: Consult the Moloch documentation for more comprehensive instructions on this model. The steps here are a brief overview, not a full tutorial.)

Add a network interface to the Moloch platform and connect it to a network data source such as a tap or port mirror.

In Moloch's "config.ini" file, set the "interface" setting to the interface detailed above.

Clearing Data

To remove SPI data from Moloch's Elasticsearch index, first stop any running capture and viewer processes. Then, run the following command:

```
$ /data/moloch/db/db.pl ⏎
  <elasticsearch url> wipe
```

(Your path may vary – /data/moloch/db/ is the typical default path for this script.)

On the FOR572-distributed Moloch VM, the "moloch_clear.sh" script automates the entire process, including stopping and restarting the Moloch services.

To re-parse any input data, re-load the pcap files as described in the "Loading Data to Moloch" section.

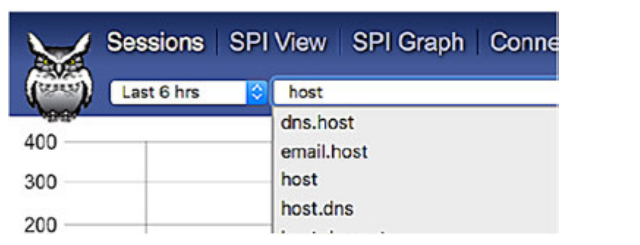
Examples:

```
$ /data/moloch/db/db.pl ⏎
  http://127.0.0.1:9200 wipe
$ sudo moloch_clear.sh
```

Query Syntax

Moloch uses a unique query syntax, but offers UI features that keep it easy to learn and use.

The search interface uses a "drop-down suggestion" feature to show the analyst all matching field names.



For more comprehensive online documentation, including a list of all fields, search syntax, and the Moloch UI itself, click the owl icon in the top left.

Basic searching uses the following syntax:

- fieldname == value
- fieldname != value
- fieldname > value
- fieldname <= value

Strings can use "*" as a wildcard. IP address fields can use full IPs or netblocks in CIDR notation. Logical conjunction is performed with "&&", "||", and parenthesis.

Searching for sessions in which any specific field exists at all requires the following syntax:

```
fieldname == EXISTS!
```

Examples:

- host.dns == *google*
- http.method == POST && host.http == *homedepot.com
- tls.cipher == EXISTS !! && tls.cipher != *DHE*

Moloch UI

The Moloch web-based interface includes several tabs, each presenting a different view of the underlying source data.

Sessions: This is the most frequently-used tab, where session data is displayed and queried. Each session can be unrolled to expose all SPI data extracted from the original content.

SPI View: Explore all of SPI fields within a data set.

SPI Graph: Any SPI field can be charted and compared to other fields over time.

Connections: A graph view comparing any two SPI fields. Extremely useful for identifying of relationships between data points at scale.

Files: Information regarding the pcap files that Moloch has loaded and parsed.

Users: List, create, delete, and manage Moloch user accounts.

Stats: Metrics for each Moloch capture node and Elasticsearch cluster member.

Settings: Manage settings for the current user.