The University of Texas at San Antonio™

The Cyber Center for Security and Analytics

UNIVERSITY OF SOUTH CAROLINA

ZEEK INSTRUSION DETECTION SERIES

Lab 8: Advanced Zeek Scripting for Anomaly and Malicious Event Detection

Document Version: 03-13-2020

Award 1829698
"CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers"

Contents

Overview

This lab covers Zeek's scripting language and introduces more advanced scripting capabilities. This lab simulates a new zero-day scanning technique and explains a Zeek script that captures this new event. The lab is designed to further highlight the customization properties of Zeek scripting.

Objectives

By the end of this lab, students should be able to:

1. Use precompiled Zeek scripts for identifying network traffic anomalies.
2. Develop a Zeek script for identifying and organizing specific malicious traffic events.
3. Generate customized malicious traffic to be used for testing purposes.

Lab topology

Figure 1 shows the lab topology. The topology uses 10.0.0.0/8 which is the default network assigned by Mininet. The *zeek1* and *zeek2* virtual machines will be used to generate and collect network traffic.

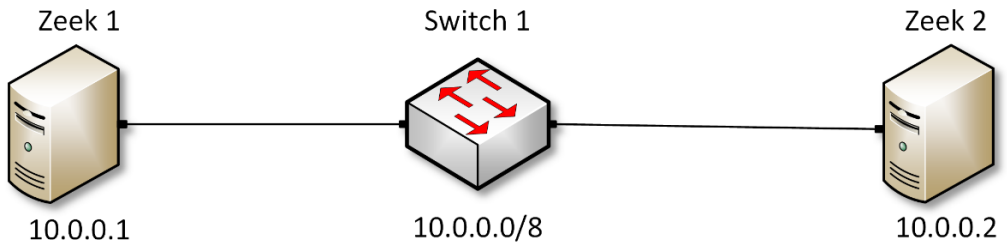


Figure 1. Lab topology.

Lab settings

The information (case-sensitive) in the table below provides the credentials necessary to access the machines used in this lab.

Table 1. Credentials to access the Client machine

Device	Account	Password
Client	admin	password

Table 2. Shell variables and their corresponding absolute paths.

Variable Name	Absolute Path
$ZEEK_INSTALL	/usr/local/zeek
$ZEEK_TESTING_TRACES	/home/zeek/zeek/testing/btest/Traces
$ZEEK_PROTOCOLS_SCRIPT	/home/zeek/zeek/scripts/policy/protocols

Lab roadmap

This lab is organized as follows:

1. Section 1: Zeek's default anomaly detection scripts.
2. Section 2: Generating customized malicious network traffic.
3. Section 3: Applying Zeek scripts to filter network traffic.

1 Zeek's default anomaly detection scripts

Zeek's scripting language can be used to identify and report network anomalies by using event-driven functions. This section introduces two default Zeek script filters that are installed by default after Zeek installation.

While these default Zeek scripts might not correctly identify every unique anomaly, they provide a comprehensive starter code that can be customized further for anomaly-based detection.

1.1 Zeek scan-event

The first default Zeek script is the *scan.zeek* script. More information on this script can be found in Zeek's documentation pages. To access the following link, users must have access to an external computer connected to the Internet, because the Zeek Lab topology does not have an active Internet connection.

```
https://docs.zeek.org/en/latest/scripts/policy/misc/scan.zeek.html
```

The file has been copied into the Zeek lab workspace directory and renamed to *ZeekDetectScans.zeek* for ease of access and name-reference clarity.

This Zeek script is used to identify scan-related traffic. Internet scanning can be split into three main categories:

1. `Vertical Scanning`: an attacker scans many ports on a single destination host address.

2. `Horizontal Scanning`: an attacker scans a single port on many destination host addresses.
3. `Block Scanning`: an attacker interweaves vertical and horizontal scanning techniques to increase complexity and become harder to track.

The script shown in the figure below list the first few lines of the *ZeekScanDetection.zeek* file.

```
1  ##! TCP Scan detection
2  # ..Authors: Sheharbano Khattak
3  #            Seth Hall
4  #            All the authors of the old scan.bro
5  @load base/frameworks/notice
6  @load base/frameworks/sumstats
7  @load base/utils/time
```

As shown in the figure above, loading other scripts is done through the `@load` statement with the following format:

```
@load <zeekscriptfile>
```

Lines 5, 6 and 7 include the functionalities found within the export blocks of the respectively included Zeek scripts.

The script leverages thresholds to determine if scan-like activities are present when processing network capture. If all the thresholds are exceeded, traffic is inferred to be scan-related.

For real time deployment, these thresholds will need to be modified dependent on the network size. For instance, a smaller network containing less IP addresses will need a lower threshold of scan packets to identify a scan-event. However, modifying these thresholds may result in an increase of false positives and true negatives, so it highly recommended to simulate and test network traffic before modification.

```
25      ## Failed connection attempts are tracked over this time interval for
26      ## the address scan detection.  A higher interval will detect slower
27      ## scanners, but may also yield more false positives.
28      const addr_scan_interval = 5min &redef;
29      ## Failed connection attempts are tracked over this time interval for
30      ## the port scan detection.  A higher interval will detect slower
31      ## scanners, but may also yield more false positives.
32      const port_scan_interval = 5min &redef;
33      ## The threshold of the unique number of hosts a scanning host has to
34      ## have failed connections with on a single port.
35      const addr_scan_threshold = 25.0 &redef;
36      ## The threshold of the number of unique ports a scanning host has to
37      ## have failed connections with on a single victim host.
38      const port_scan_threshold = 15.0 &redef;
39      global Scan::addr_scan_policy: hook(scanner: addr, victim: addr, scanned_port: port);
40      global Scan::port_scan_policy: hook(scanner: addr, victim: addr, scanned_port: port);
41  }
```

The figure above shows the thresholds in the *ZeekScanDetection.zeek* file. The thresholds are explained as follows. Each number represents the respective line number:

28. `const addr_scan_interval`: threshold to check a source IP address for varying destination IP address scan-related traffic. The default interval is 5 minutes.
32. `const port_scan_interval`: threshold to check a source IP address for varying destination port scan-related traffic. The default interval is 5 minutes.
35. `const addr_scan_threshold`: threshold of unique destination IP addresses that a single host attempts to contact. The default threshold is 25 unique destination IP addresses.
38. `const port_scan_threshold`: threshold of unique destination ports that a single host attempts to contact. The default threshold is 15 unique destination ports.

1.2 Zeek bruteforce-event

The second default Zeek script is the *detect-bruteforcing.zeek* script. More information on this script can be found in Zeek's documentation pages. To access the following link, users must have access to an external computer connected to the Internet, because the Zeek Lab topology does not have an active Internet connection.

```
https://docs.zeek.org/en/stable/scripts/policy/protocols/ssh/detect-
bruteforcing.zeek.html
```

The file has been copied into the Zeek lab workspace directory and renamed to *ZeekDetectBruteForce.zeek* for ease of access and name-reference clarity.

This Zeek script is used to identify brute-force password attacks. Brute-force attacks can be identified by several failed login attempts. This denotes that an attacker is attempting to systematically submit credentials until the correct credentials are found. The motivation behind this attack is to gain authorized access to an account, machine or server.

The script leverages the following thresholds to determine if scan-like activities are present when processing network capture. During real time deployment, these thresholds should be modified depending on the network size. The number of failed login attempts (or duration) should be modified to increase the script's accuracy.

```
1  ##! FTP brute-forcing detector, triggering when too many rejected usernames or
2  ##! failed passwords have occurred from a single address.
3  @load base/protocols/ftp
4  @load base/frameworks/sumstats
5  @load base/utils/time
6  module FTP;
7  export {
8      redef enum Notice::Type += {
9          ## Indicates a host bruteforcing FTP logins by watching for too
10         ## many rejected usernames or failed passwords.
11         Bruteforcing
12     };
13     ## How many rejected usernames or passwords are required before being
14     ## considered to be bruteforcing.
15     const bruteforce_threshold: double = 20 &redef;
16     ## The time period in which the threshold needs to be crossed before
17     ## being reset.
18     const bruteforce_measurement_interval = 15mins &redef;
19 }
```

The thresholds are explained as follows. Each number represents the respective line number:

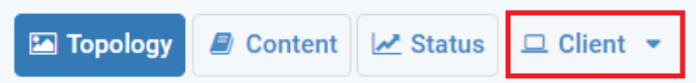
15. `const bruteforce_threshold`: threshold for the number of failed authentications attempts a source IP address can make. The default value is 20 failed attempts within the related time interval threshold.
18. `const bruteforce_measurement_interval`: threshold for the time to check a source IP address for failed authentication attempts. The default interval is 15 minutes.

2 Generating customized malicious network traffic

This section introduces creating and using a new Zeek script, tailored to react to more specific events.

2.1 Starting a new instance of Zeek

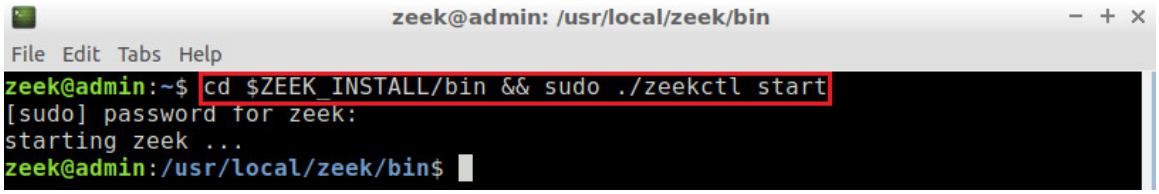
Step 1. From the top of the screen, click on the *Client* button as shown below to enter the *Client* machine.



Step 2. The *Client* machine will now open, and the desktop will be displayed. On the left side of the screen, click on the LXTerminal icon as shown below.

Step 3. Start Zeek by entering the following command on the terminal. This command enters Zeek's default installation directory and invokes `Zeekctl` tool to start a new instance. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key. When prompted for a password, type `password` and hit `Enter`.

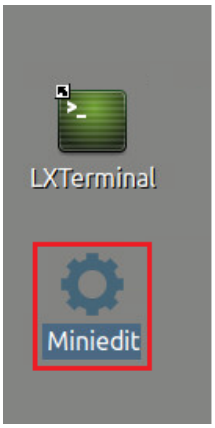
```
cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
```



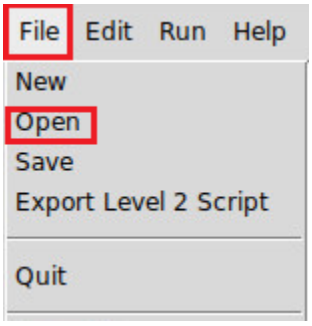
A new instance of Zeek is now active, and we are ready to proceed to the next section of the lab.

2.2 Launching Mininet

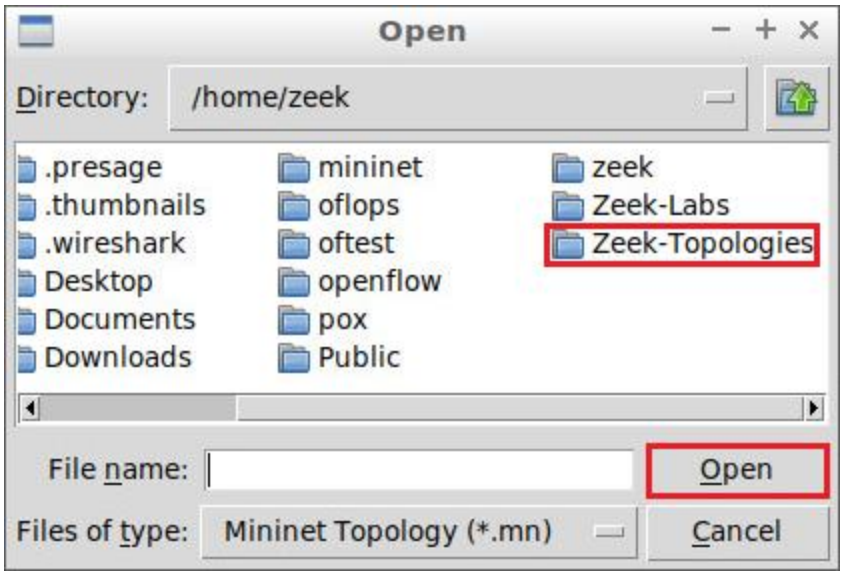
Step 1. From the *Client* machine's desktop, on the left side of the screen, click on the MiniEdit icon as shown below. When prompted for a password, type `password` and hit *Enter*. The MiniEdit editor will now launch.



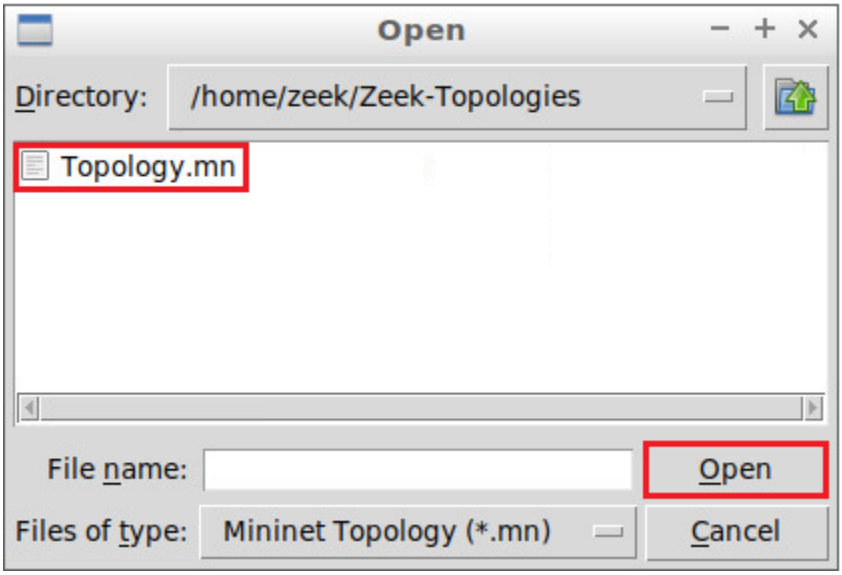
Step 2. The MiniEdit editor will now launch and allow for the creation of new, virtualized lab topologies. Load the correct topology by clicking the *Open* button within the *File* tab on the top left of the MiniEdit editor.



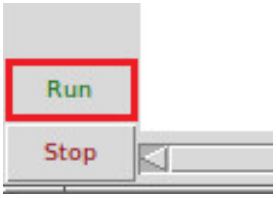
Step 3. Navigate to the Zeek-Topologies directory by scrolling to the right of the active directories and double clicking the Zeek-Topolgies icon, or by clicking the *Open* button.



Step 4. Select the *Topology.mn* file by double clicking the *Topolgies.mn* icon, or by clicking the *Open* button.

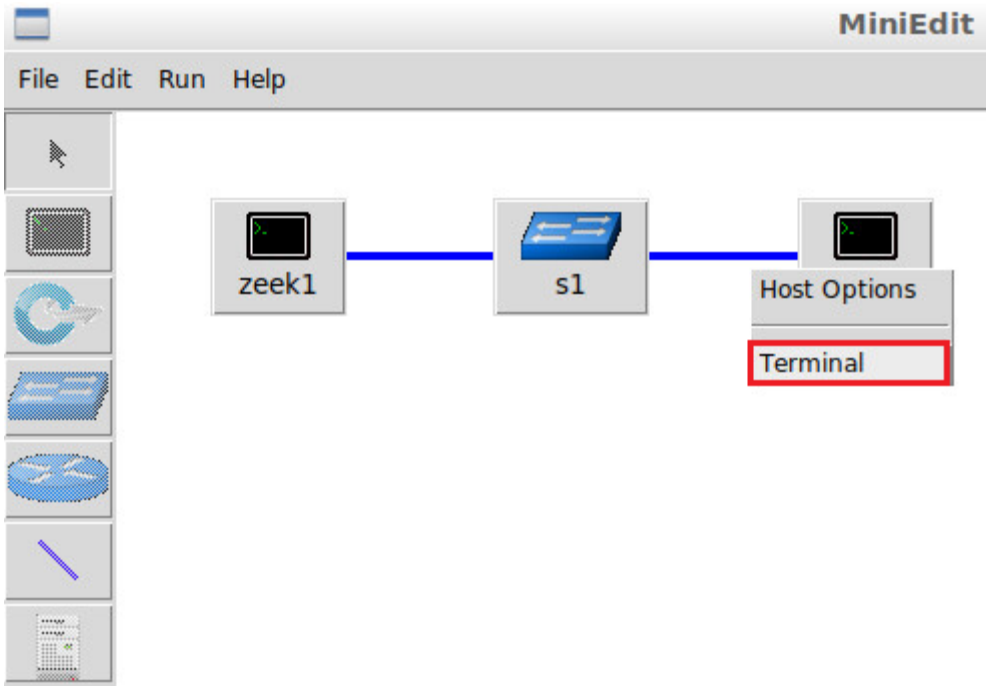


Step 5. To begin running the virtual machines, navigate to the *Run* button, found on the bottom left of the Miniedit editor, and select the *Run* button, as seen in the image below.



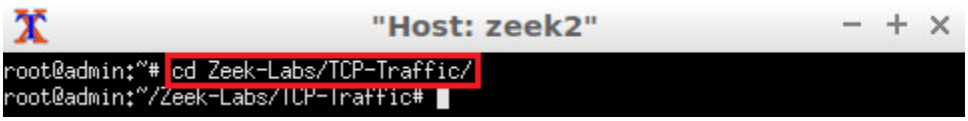
2.3 Setting up the zeek2 virtual machine for live network capture

Step 1. Launch the *zeek2* terminal by holding the right mouse button on the desired machine and clicking the `Terminal` button.



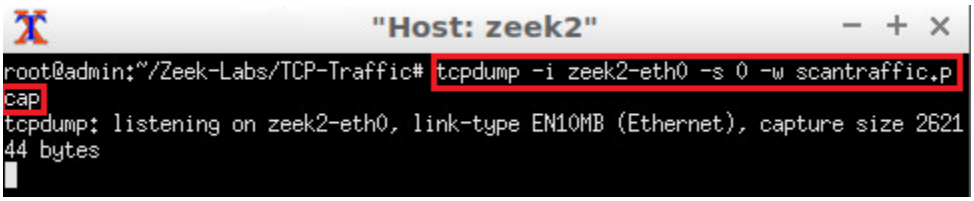
Step 2. Using the *zeek2* terminal, navigate to the TCP-Traffic directory.

```
cd Zeek-Labs/TCP-Traffic/
```



Step 3. Start live packet capture on interface *zeek2-eth0* and save the output to a file named *scantraffic.pcap*.

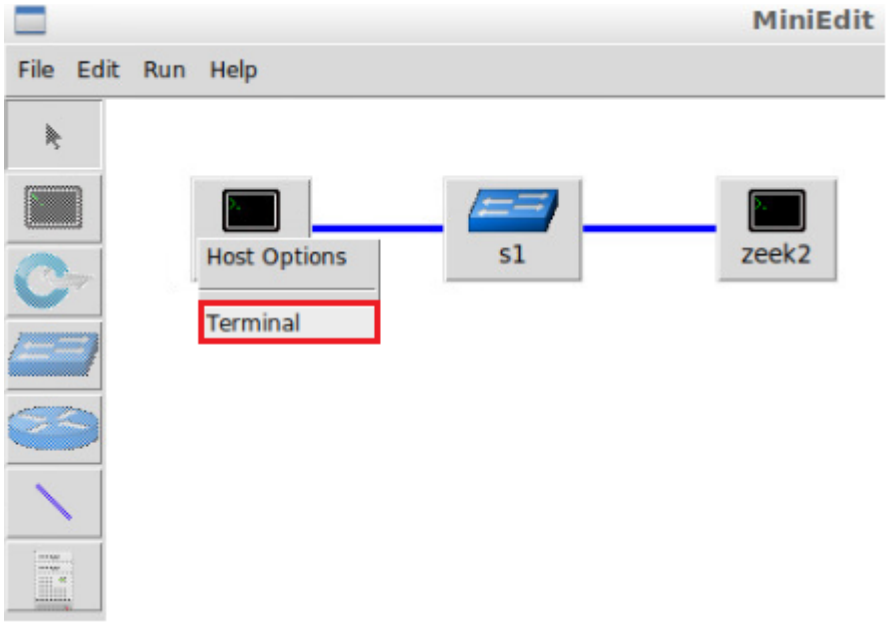
```
tcpdump -i zeek2-eth0 -s 0 -w scantraffic.pcap
```



The *zeek2* virtual machine is now ready to begin collecting live network traffic. Next, we will use the *zeek1* machine to generate scan-based network traffic.

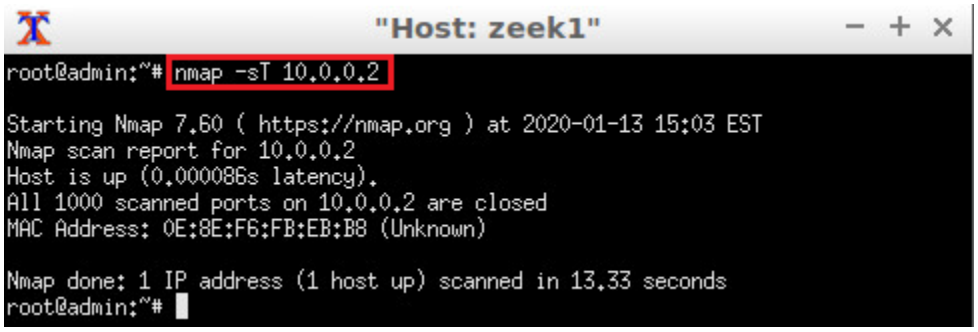
2.4 Using the zeek1 virtual machine for network scanning activities

Step 1. Minimize the *zeek2* `Terminal` and open the *zeek1* `Terminal` by following the previous steps. If necessary, right click within the Miniedit editor to activate your cursor.



Step 2. Launch a TCP connect scan against the *zeek2* machine.

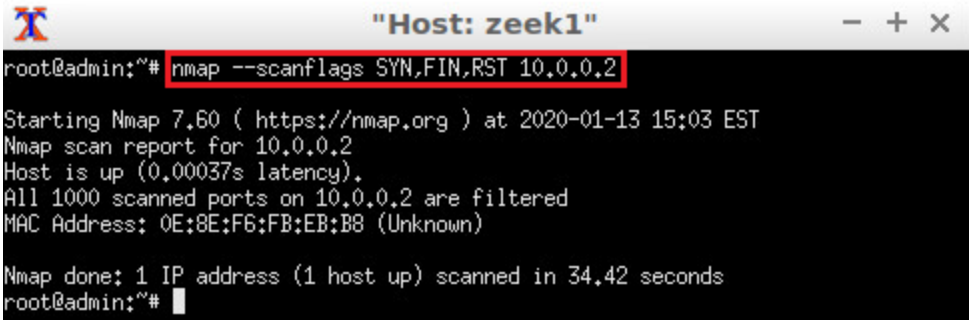
```
nmap -sT 10.0.0.2
```



Step 3. Launch a scan against the *zeek2* machine with the SYN, FIN and RST flags set. We will label this scan as *Case1*.

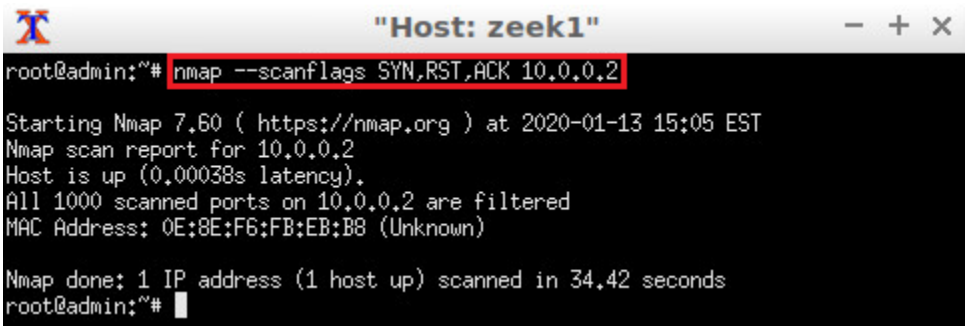
```
nmap --scanflags SYN,FIN,RST 10.0.0.2
```

By specifying the `--scanflags` option, we can control which TCP flags are included in the packet header.



Step 4. Launch a scan against the *zeek2* machine with the SYN, RST and ACK flags set. We will label this scan as *Case2*.

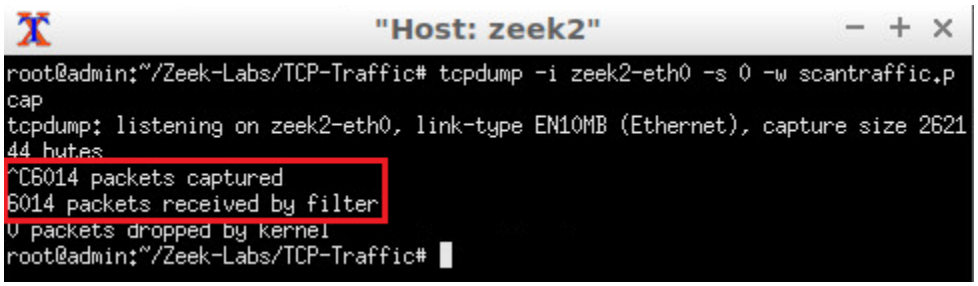
```
nmap --scanflags SYN,RST,ACK 10.0.0.2
```



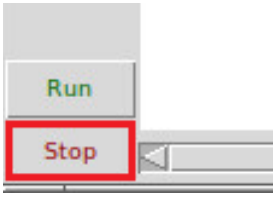
2.4.1 Terminating live network capture

Step 1. Minimize the *zeek1* `Terminal` and open the *zeek2* `Terminal` using the navigation bar at the bottom of the screen. If necessary, right click within the Miniedit editor to activate your cursor.

Step 2. Use the `Ctrl+c` key combination to stop live traffic capture. Statistics of the capture session will we be displayed. 6014 packets were recorded by the interface, which were then captured and stored in the new *scantraffic.pcap* file.



Step 3. Stop the current Mininet session by clicking the *Stop* button on the bottom left of the MiniEdit editor and close the MiniEdit editor by clicking the ☒ on the top right of the editor.



3 Applying Zeek scripts to filter network traffic

Now that we have collected traffic containing the *zero-day* exploits, we will process the packet capture file using Zeek.

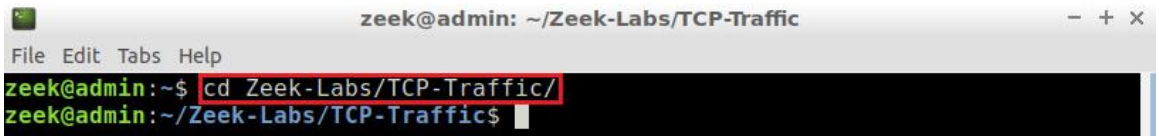
3.1 Applying the ZeekDetectScans filter

After successfully conducting a number of TCP-based scans, the *scanpackets.pcap* packet capture file now contains the required traffic. In this section we analyze the collected network traffic using Zeek.

Step 1. On the left side of the *Client* desktop, click on the LXTerminal icon as shown below.

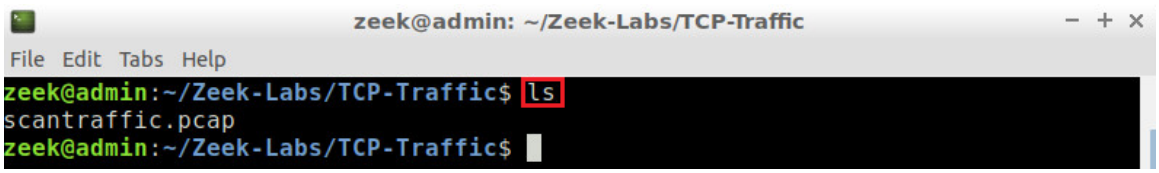
Step 2. Navigate to the *TCP-Traffic* directory to find the *scantraffic.pcap* file.

```
cd Zeek-Labs/TCP-Traffic/
```



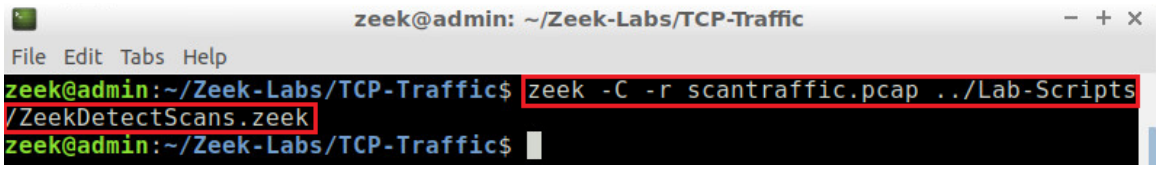
Step 3. View the file contents of the *TCP-Traffic* directory to ensure that the *scantraffic.pcap* file was successfully saved.

```
ls
```



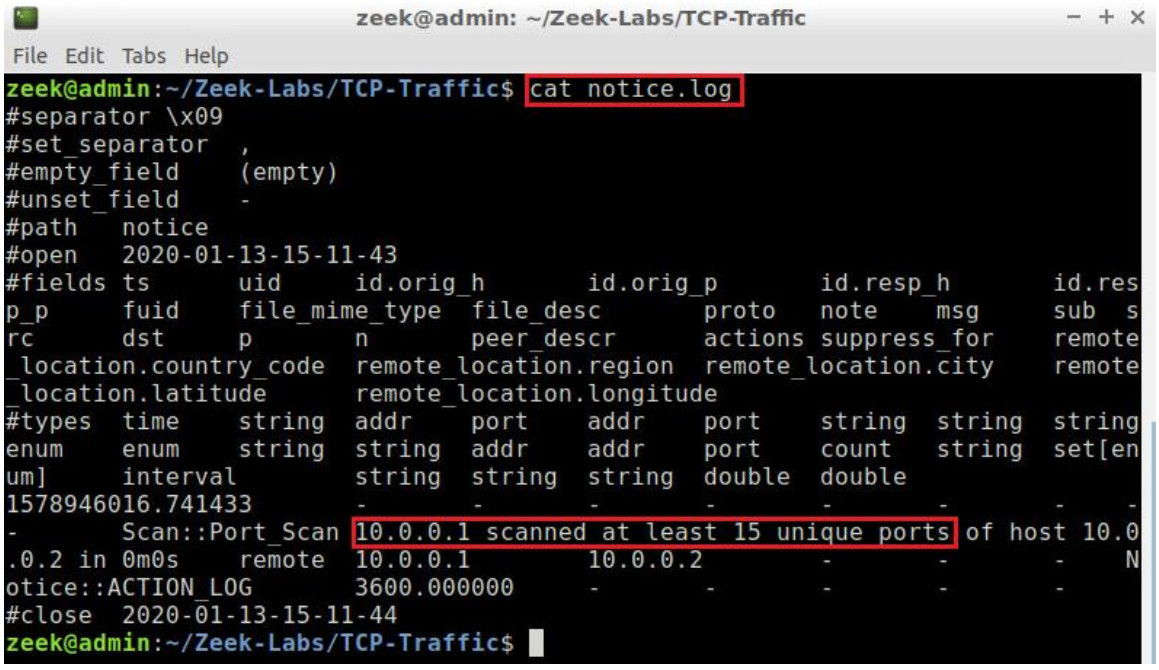
Step 4. Process the *scantraffic.pcap* packet capture file using *ZeekScanDetection.zeek*. It is possible to use the `tab` key to autocomplete the longer paths.

```
zeek -C -r scantraffic.pcap ../Lab-Scripts/ZeekDetectScans.zeek
```



Step 5. Display the contents of the *notice.log* file using the `cat` command.

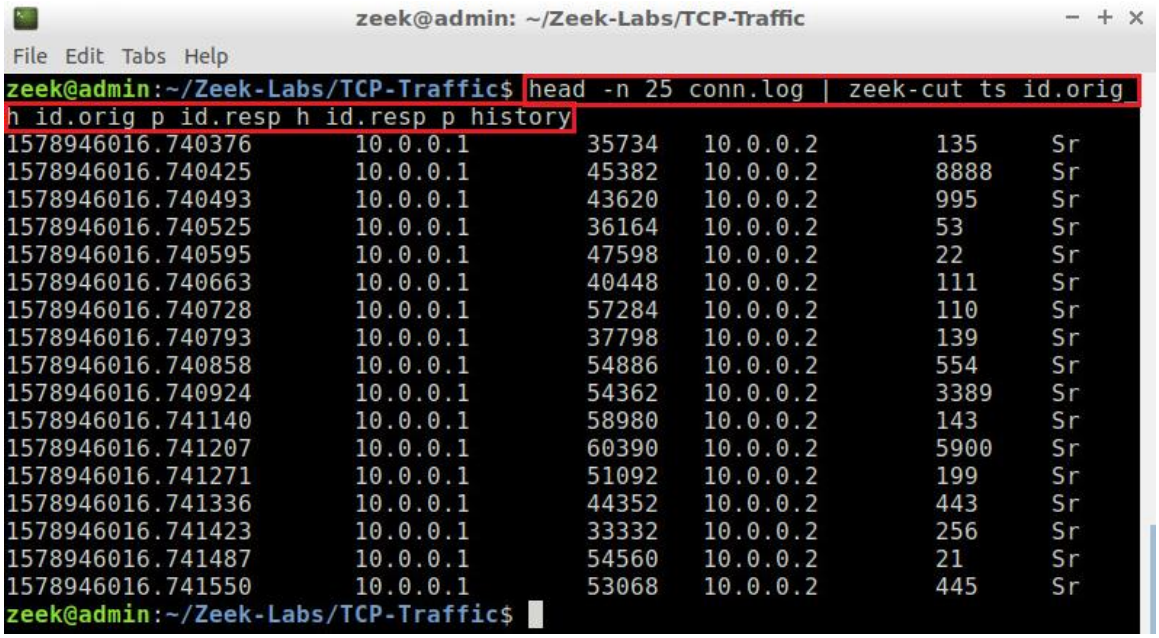
```
cat notice.log
```



Within the *notice.log* file, we can see the *zeek1* machine has been identified for creating scan-based network traffic and exceeding the 15-ports threshold configured earlier.

Step 6. Display the contents of the *conn.log* file using the following command.

```
head -n 25 conn.log | zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p
history
```



The Terminal command is explained as follows:

- `head -n 25 conn.log`: returns the top 25 rows of the *conn.log* file, specified by the `-n` option.
- `| zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p history`: uses the `zeek-cut` utility to return the specified columns and remove padding.

The `history` column (last column in the figure above) contains information regarding which TCP flags were found within a packet header:

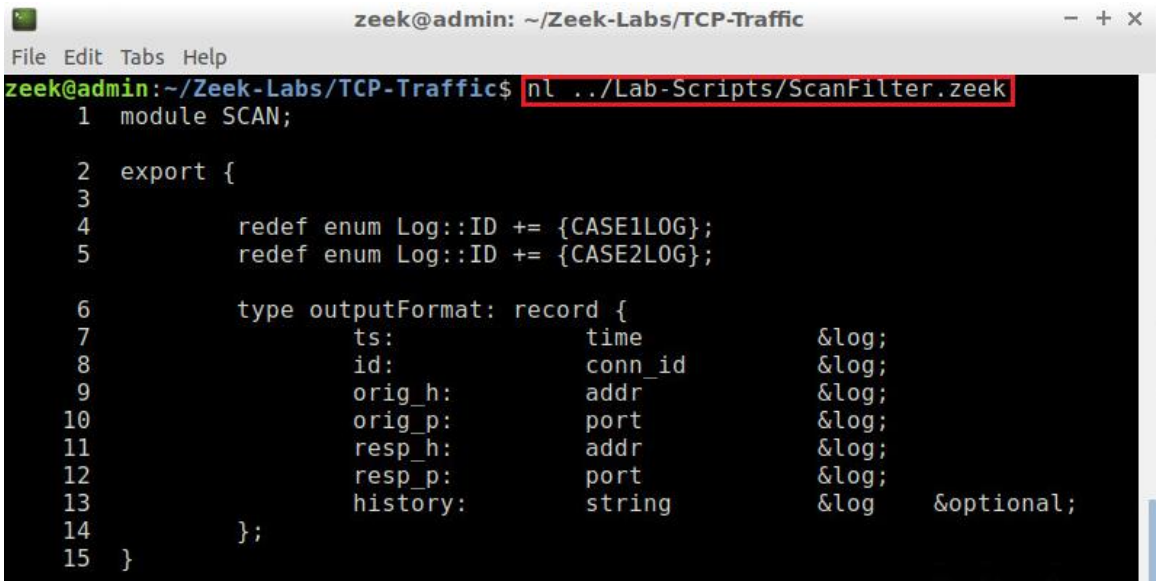
- `s`: SYN flag.
- `h`: SYN+ACK flags.
- `a`: ACK flag.
- `f`: FIN flag.
- `r`: RST flag.
- `u`: URG flag.
- `q`: Multiple flags set.

The event is attributed to the host when the flag letter is uppercase; otherwise, it is attributed to the receiver. In this example, the capital S and lowercase r denotes the SYN flag sent from the host, while the receiver responded with a RST flag.

3.2 Applying the ScanFilter filter

Step 1: Display the contents of the *ScanFilter.zeek* file using `nl`.

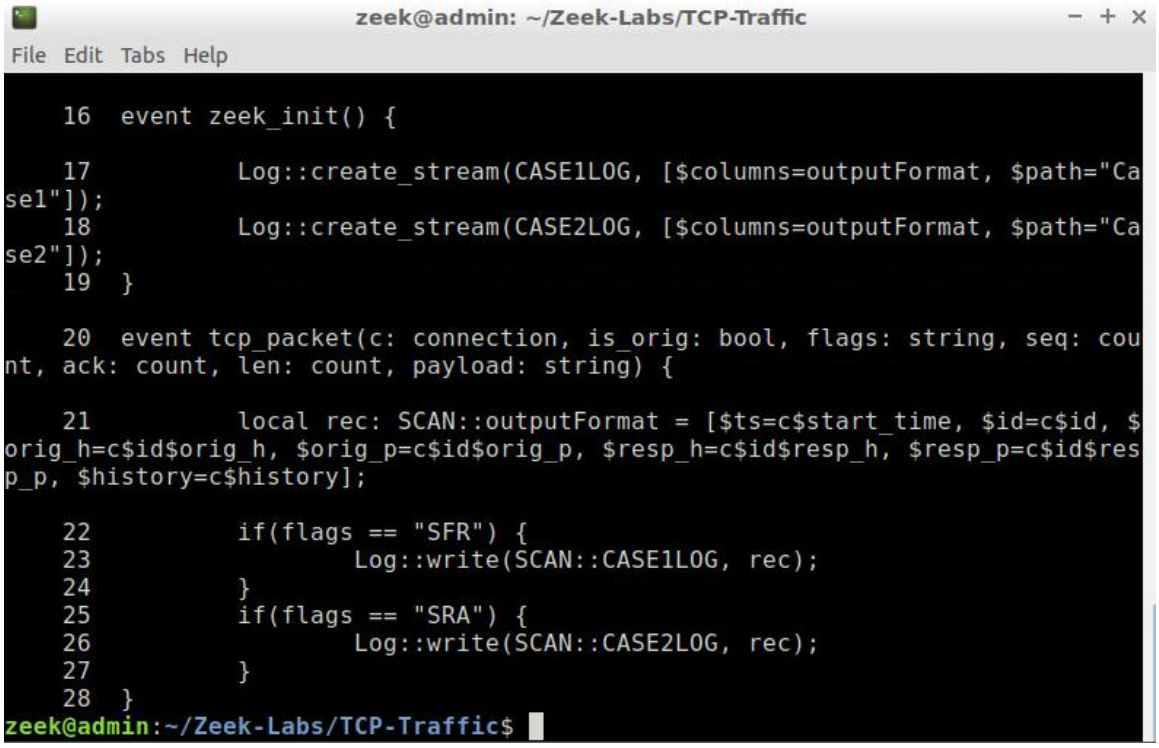
```
nl ../Lab-Scripts/ScanFilter.zeek
```



The script is explained as follows. Each number represents the respective line number:

1. Declares a new module workspace.
2. Export block allows code to be accessed outside the current module workspace.
3. Creates and appends the `CASE1LOG` to the list of Log files.
4. Creates and appends the `CASE2LOG` to the list of Log files.
6. Block that includes all the columns and features to be included in these new log files. Each will contain a variable type and output location:

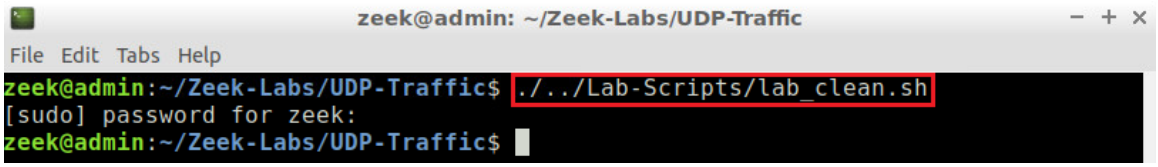
    - `ts`: time that the packet was received.
    - `id`: packet identification number.
    - `orig_h`: source IP address.
    - `orig_p`: source port.
    - `resp_h`: destination IP address.
    - `resp_p`: destination port.
    - `history`: string of flag characters.



16. Initialization event.
17. Creates a new log stream using the previously introduced `CASE1LOG` LOG ID, `outputFormat` column formatting and a file name path.
18. Creates a new log stream using the previously introduced `CASE2LOG` LOG ID, `outputFormat` column formatting and a file name path.
20. Event triggered when a TCP packet is processed.
21. Creates a local variable `rec` to store the column-related information, using the current packet data, accessed with the `c$id$<column>` format.
22. Checks if the SFR flag combination is present in the packet. This relates to the history column, containing SYN-FIN-RST flags.
23. If the SFR flag combination is present, the packet will be written to the `CASE1LOG` log stream with the packet information passed through the local variable `rec`.
24. Checks if the SRA flag combination is present in the packet. This relates to the history column, containing SYN-RST-ACK flags.
25. If the SRA flag combination is present, the packet will be written to the `CASE2LOG` log stream with the packet information passed through the local variable `rec`.

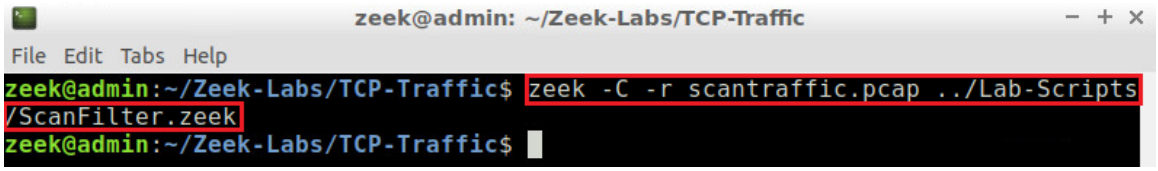
Step 2. Execute the *lab_clean.sh* shell script to clear the directory. If required, type `password` as the password.

```
./../Lab-Scripts/lab_clean.sh
```



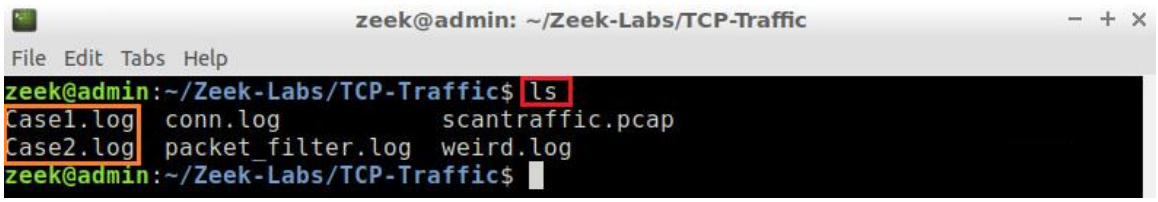
Step 3: Process the *scantraffic.pcap* packet capture file using *ScanFilter.zeek*. It is possible to use the tab key to autocomplete the longer paths.

```
zeek -C -r scantraffic.pcap ../Lab-Scripts/ScanFilter.zeek
```



Step 4: List the generated log files in the current directory.

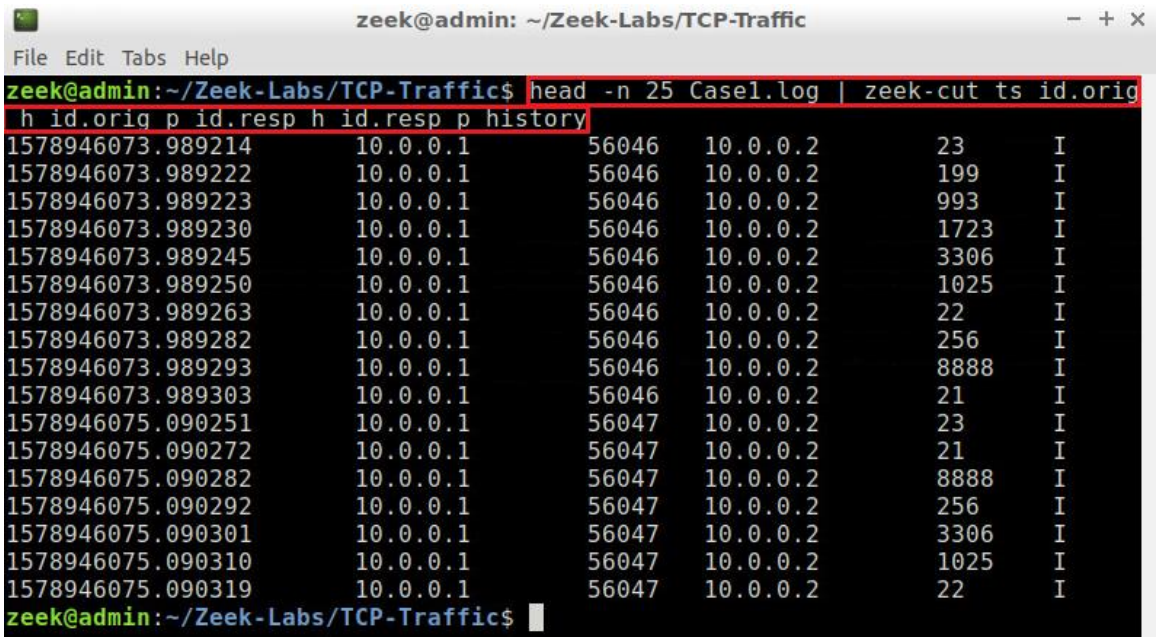
```
ls
```



Note the *Case1.log* and *Case2.log* files, highlighted by the orange box, generated by including the *ScanFilter.zeek* filter during processing.

Step 5: View the contents of the *Case1.log* file.

```
head -n 25 Case1.log | zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p
history
```



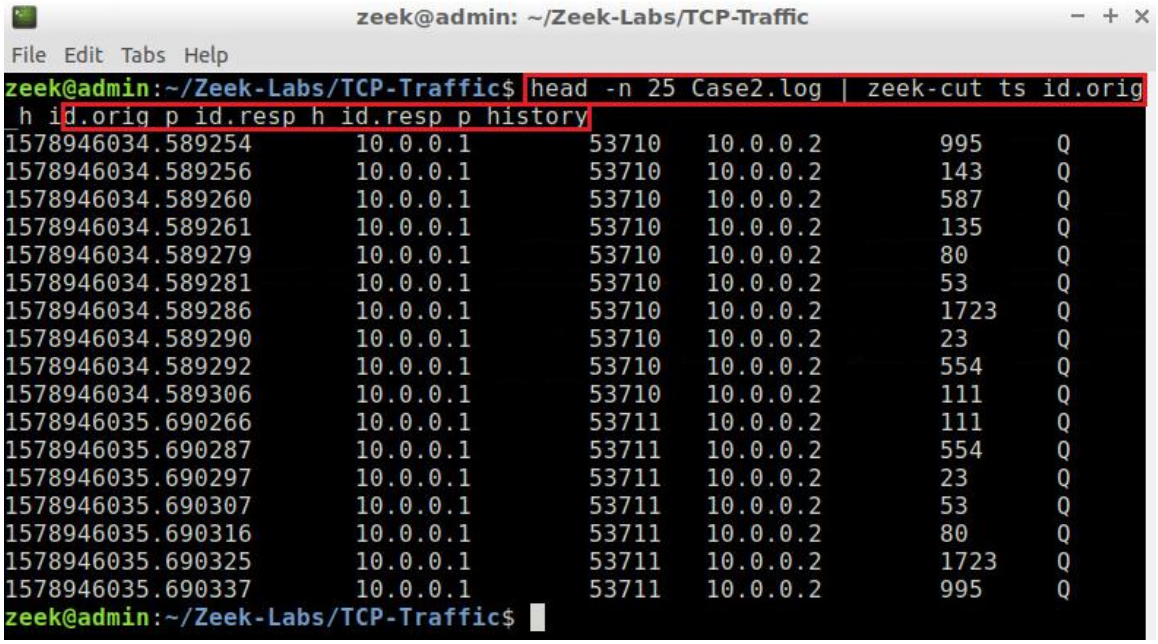
The Terminal command is explained as follows:

- `head -n 25 Case1.log`: returns the top 25 rows of the *conn.log* file, specified by the `-n` option.
- `| zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p history`: uses the `zeek-cut` utility to only return the specified columns, and removes padding.

Unlike the default example, we can see the `history` column contains the exact same flag. Our filter was successful in organizing the traffic related to the `Case1` exploit.

Step 6: Display the contents of the *Case2.log* file.

```
head -n 25 Case2.log | zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p
history
```



The Terminal command is explained as follows:

- `head -n 25 Case2.log`: returns the top 25 rows of the *conn.log* file, specified by the `-n` option.
- `| zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p history`: uses the `zeek-cut` utility to only return the specified columns, and removes padding.

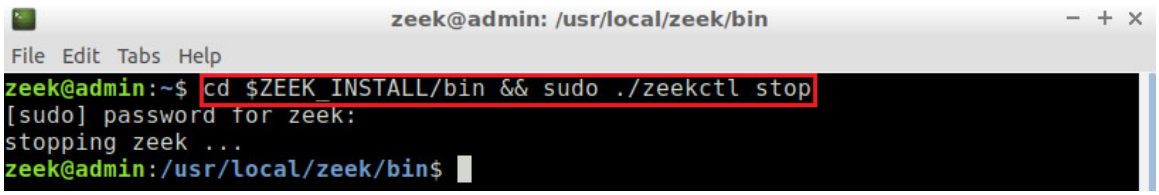
Unlike the default example, we can see the `history` column contains the exact same flag. Our filter was successful in organizing the traffic related to the `Case2` exploit.

3.3 Closing the current instance of Zeek

After you have finished the lab, it is necessary to terminate the currently active instance of Zeek. Shutting down a computer while an active instance persists will cause Zeek to shut down improperly and may cause errors in future instances.

Step 1. Stop Zeek by entering the following command on the terminal. If required, type `password` as the password. If the Terminal session has not been terminated or closed, you may not be prompted to enter a password. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key

```
cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
```



Concluding this lab, we introduced default frameworks for anomaly-detection scripts. We generated malicious network traffic to simulate a *zero-day* exploit, and then processed the traffic using a customized a Zeek script. With the resulting Zeek log files, these exploits can be studied for additional analysis and mitigation.

References

1. Bilge, Leyla, and Tudor Dumitraş. "Before we knew it: an empirical study of zero-day attacks in the real world." *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.
2. “Writing scripts”, Zeek user manual, [Online], Available: Zeek, https://docs.zeek.org/en/stable/examples/scripting/#the-event-queue-and-event-handlers.