



The University of Texas at San Antonio™

The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 1: Introduction to the Capabilities of Zeek

Document Version: **03-13-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to Zeek	4
1.1 The Zeek event engine	5
1.1.1 State management.....	5
1.1.2 Transport layer analyzers.....	5
1.1.3 Application layer analyzers	5
1.1.4 Infrastructure	5
1.2 The Zeek policy script interpreter	6
1.3 Zeek analyzers	6
1.4 Signatures.....	6
1.5 ZeekControl	7
2 Using ZeekControl to update the status of Zeek	7
2.1 Starting a new instance of Zeek	9
2.2 Stopping the active instance of Zeek	10
3 Introduction to Zeek's traffic analysis capabilities	10
3.1 Processing offline packet capture files	11
3.1.1 Command format for processing packet capture files	11
3.1.2 Leveraging a script to detect brute force attacks present in a pcap file	11
3.2 Launching Mininet.....	12
3.3 Generating and analyzing live network traffic capture	15
3.3.1 Leveraging the Tcpcmdump command utility	16
3.3.2 Capturing live network traffic.....	16
3.3.3 Analyzing the newly captured network traffic	18
References	20

Overview

This lab introduces Zeek, an open-source network analysis framework primarily used in security monitoring and traffic analysis. The primary focus of this lab is to explain Zeek's layered architecture while demonstrating Zeek's capabilities towards performing network traffic analysis.

Objectives

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

1. Understand Zeek's layered architecture.
2. Start and manage a Zeek instance using the *ZeekControl* utility.
3. Use Zeek to process packet captures files.
4. Generate and analyze live network traffic in Zeek.

Lab topology

Figure 1 displays the topology of the lab. This lab utilizes the *Client* machine to host and configure the Zeek IDS. The *zeek1* and *zeek2* virtual machines will be used to generate and collect network traffic.

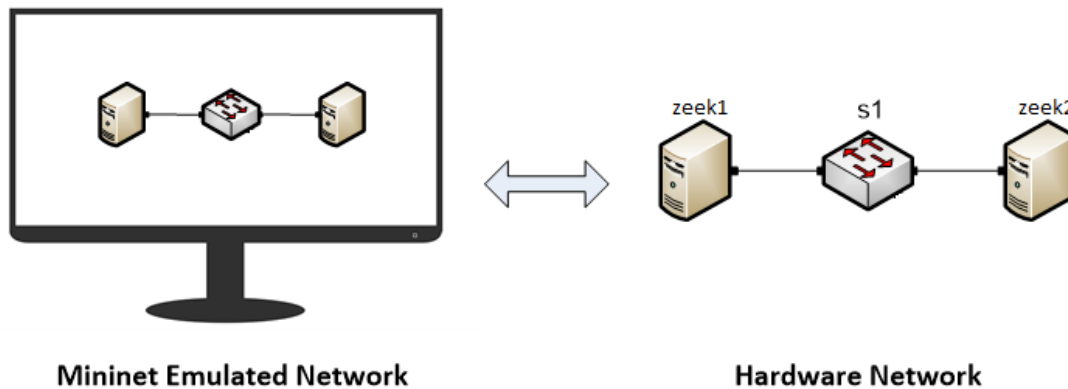


Figure 1. Lab topology.

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: Introduction to Zeek.
2. Section 2: Using *ZeekControl* to update the status of Zeek.
3. Section 3: Introduction to Zeek’s traffic analysis capabilities.

1 Introduction to Zeek

Zeek is a passive, open-source network traffic analyzer. It is primarily used as a security monitor that inspects all traffic on a network link for signs of suspicious activity¹. It can run on commodity hardware with standard UNIX-based systems and can be used as a passive network monitoring tool.

Setting Zeek as a node with an assigned IP address on the monitored network is not mandatory. Figure 2 shows Zeek’s layered architecture. Once Zeek receives packets, its *event engine* converts them into *events*. The *events* are then forwarded to the policy script interpreter, which generates logs, notifications, and/or actions.

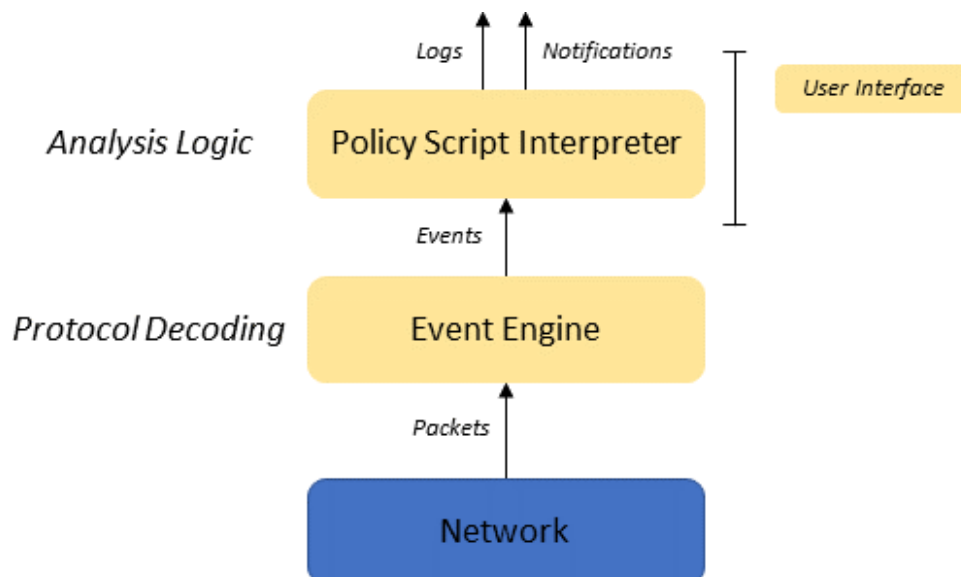


Figure 2. Zeek’s architecture.

Zeek uses the standard `libpcap` library for capturing packets to be used in network monitoring and analysis.

1.1 The Zeek event engine

The event engine layer performs low-level network packets analysis. It receives raw packets from the network layer (packet capture), sorts them by connection, reassembles data streams, and decodes application layer protocols. Whenever it encounters something potentially relevant to the policy layer, it generates an event.

The event engine consists of several analyzers responsible for well-defined tasks. Typical tasks include decoding a specific protocol, performing signature-matching, identifying backdoors, etc. Usually, an analyzer is accompanied by a default script which implements some general policy adjustable to the local environment. The event engine can be divided into four major parts.

1.1.1 State management

Zeek's main data structure is a connection which follows typical flow identification mechanisms, such as 5-tuple approaches. The 5-tuple structure consists of the source IP address/port number, destination IP address/port number, and the protocol in use. For a connection-oriented protocol like TCP, the definition of a connection is more clear-cut, however for others such as UDP and ICMP, Zeek implements a flow-like abstraction to aggregate packets. Each packet belongs to exactly one connection.

1.1.2 Transport layer analyzers

On the transport layer, Zeek analyzes TCP, UDP packets. In TCP, Zeek's associated analyzer closely follows the various state changes, keeps track of acknowledgments, handles retransmissions and much more.

1.1.3 Application layer analyzers

The analysis of the application layer data of a connection depends on the service. There are analyzers for a wide variety of different protocols, e.g. HTTP, SMTP or DNS, that generally conduct detailed analysis of the data stream.

1.1.4 Infrastructure

The general infrastructure of Zeek includes the event and timer management components, the script interpreter, and data structures.

1.2 The Zeek policy script interpreter

While the event engine itself is policy-neutral, the top layer of Zeek defines the environment-specific network security policy. By writing handlers for events that may be raised by the event engine, the user can precisely define the constraints within the given network. If a security breach is detected, the policy layer generates an alert.

New event handlers can be created in Zeek's own scripting language. While providing all expected convenience of a powerful scripting language, it has been designed with network intrusion detection in mind. While it is expected that additional policy scripts are written by the user, there are nevertheless several default scripts included with the initial installation of Zeek. These default scripts already perform a wide range of analyses and are easily customizable.

1.3 Zeek analyzers

The majority of Zeek's analyzers are in its event engine with accompanying policy scripts that can be customized by the user. Sometimes, however, the analyzer is just a policy script implementing multiple event handlers. The analyzers perform application layer decoding, anomaly detection, signature matching and connection analysis. Zeek has been designed so that it is easy to add additional analyzers.

1.4 Signatures

Most network intrusion detection systems (NIDS) match a large set of signatures against the network traffic. Here, a signature is a pattern of bytes that the NIDS tries to locate in the payload of network packets. As soon as a match is found, the system generates an alert.

A well-known IDS system is *Snort*; conversely, Zeek's general approach to intrusion detection has a much broader scope than traditional signature-matching, yet still contains a signature engine providing a functionality that is similar to that of other systems. Furthermore, while Zeek implements its own flexible signature language, there exists a converter which directly translates Snort's signatures into Zeek's syntax, as shown below:

```

alert tcp any any -> [a.b.0.0/16,c.d.e.0/24] 80
( msg:"WEB-ATTACKS conf/httpd.conf attempt";
  nocase; sid:1373; flow:to_server,established;
  content:"conf/httpd.conf"; [...] )
    
```

(a) Snort

```

signature sid-1373 {
  ip-proto == tcp
  dst-ip == a.b.0.0/16,c.d.e.0/24
  dst-port == 80
  # The payload below is actually generated in a
  # case-insensitive format, which we omit here
  # for clarity.
  payload /*.conf\httpd.conf/
  tcp-state established,originator
  event "WEB-ATTACKS conf/httpd.conf attempt"
}%
    
```

(b) Zeek

Figure 3. Example of signature conversion¹. (a) Snort’s signature. (b) Zeek’s signature.

1.5 ZeekControl

ZeekControl, formerly known as **BroControl**, is an interactive shell for easily operating and managing Zeek installations on a single system or across multiple systems in a traffic-monitoring cluster.

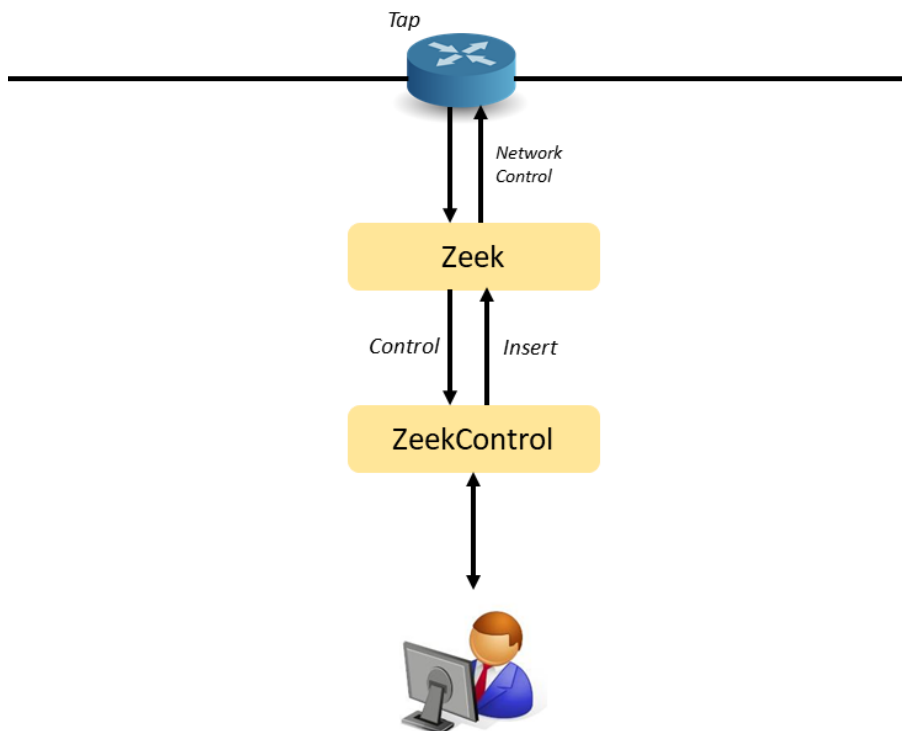
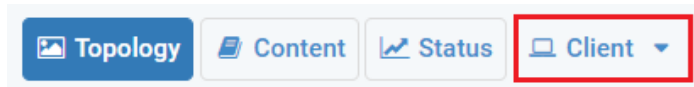


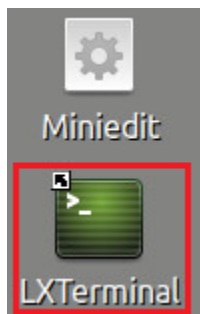
Figure 4. ZeekControl scheme.

2 Using ZeekControl to update the status 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.

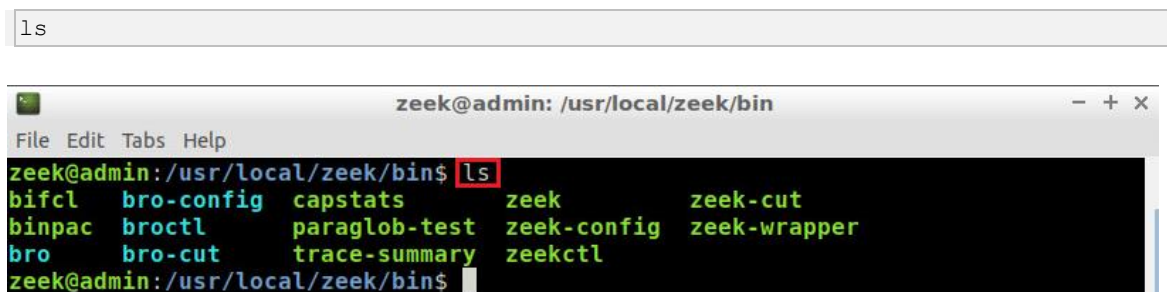


Step 3. Using the Terminal, input the following command to enter the *ZeekControl* directory. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key.



The active directory will change, as seen on the second line of the Terminal. Note that `$ZEEK_INSTALL` variable was substituted by its value (`/usr/local/zeek`) listed in Table 2.

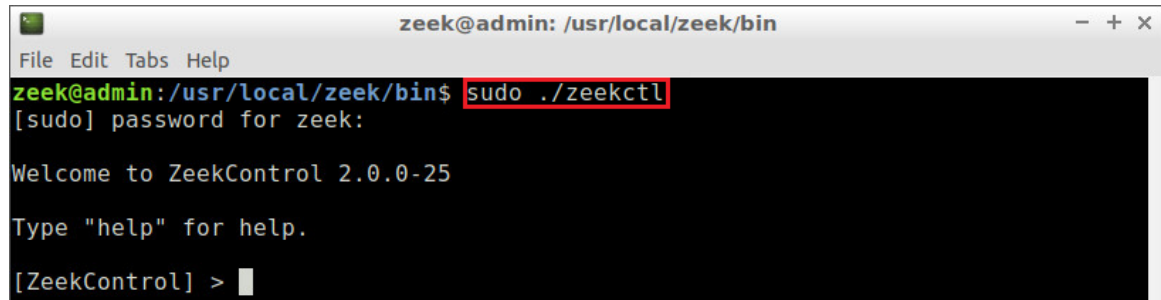
Step 4. Use the following command to view the contents of the active directory.



The directory contents will be displayed. The green file name portrays an executable file.

Step 5. Use the following command to launch the `ZeekControl` tool. When prompted for a password, type `password` and hit `Enter`.


```
sudo ./zeekctl
```



Once active, the `ZeekControl` prompt will be displayed within the Terminal. The `help` command will display additional information regarding `ZeekControl`.

2.1 Starting a new instance of Zeek

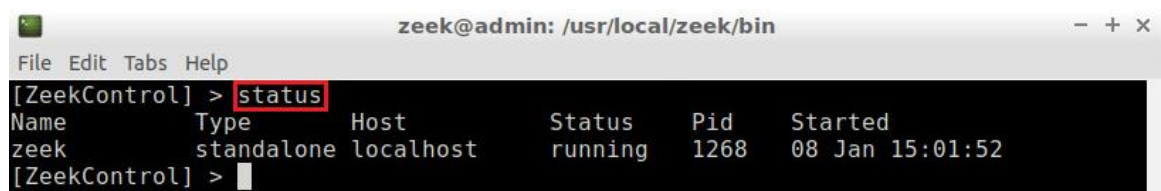
Step 1. To initialize Zeek, enter the following command into the `ZeekControl` prompt.

```
start
```



Step 2. Use the following command to view the status of the currently active Zeek instance to ensure that it is active.

```
status
```



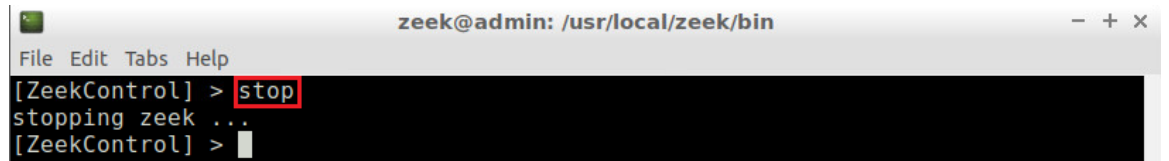
The `running` status indicates that Zeek is currently active and functioning properly. The output of the `status` command includes other useful parameters:

- `Name`: the name of the Zeek instance.
- `Type`: the type of the instance (standalone in our case).
- `Host`: the hostname (localhost).
- `Pid`: the process ID. This ID can be used with other tools like `kill` to send a signal to the process.
- `Started`: the starting date and time of the instance.

2.2 Stopping the active instance of Zeek

Step 1. To stop Zeek, enter the following command into the `ZeekControl` prompt.

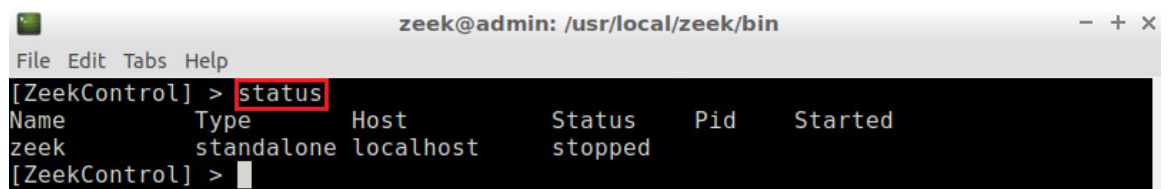
```
stop
```



The screenshot shows a terminal window titled "zeek@admin: /usr/local/zeek/bin". The prompt is "[ZeekControl] >". The user enters "stop", which is highlighted with a red box. The terminal output shows "stopping zeek .." followed by a new prompt "[ZeekControl] >".

Step 2. Use the following command to verify the exit status of Zeek.

```
status
```



The screenshot shows a terminal window titled "zeek@admin: /usr/local/zeek/bin". The prompt is "[ZeekControl] >". The user enters "status", which is highlighted with a red box. The terminal output shows a table with the following content:


Name	Type	Host	Status	Pid	Started
zeek	standalone	localhost	stopped		

The prompt then returns to "[ZeekControl] >".

The `stopped` status indicates that Zeek is currently stopped.

Step 3. To restart Zeek, enter the following command into the `ZeekControl` prompt.

```
start
```



The screenshot shows a terminal window titled "zeek@admin: /usr/local/zeek/bin". The prompt is "[ZeekControl] >". The user enters "start", which is highlighted with a red box. The terminal output shows "starting zeek .." followed by a new prompt "[ZeekControl] >".

Step 4. Use the following command to exit `ZeekControl`.

```
exit
```



The screenshot shows a terminal window titled "zeek@admin: /usr/local/zeek/bin". The prompt is "[ZeekControl] >". The user enters "exit", which is highlighted with a red box. The terminal output shows "zeek@admin: /usr/local/zeek/bin\$" followed by a new prompt.

Note that exiting the `ZeekControl` tool does not stop Zeek. Zeek is only stopped by explicitly using the `stop` command in the `ZeekControl` prompt.

3 Introduction to Zeek's traffic analysis capabilities

Zeek's broad range of traffic analysis capabilities makes it an exceptional intrusion detection system (IDS) and network analysis framework. Zeek is proficient in processing packet capture (pcap) files and logging traffic on a given network interface.

3.1 Processing offline packet capture files

Linux-based systems process packet capture (pcap) files using the `libpcap` library. In Zeek, it is possible to capture live traffic and analyze trace files. In the following example, we analyze a pcap file using a premade script that detects brute force attacks.

3.1.1 Command format for processing packet capture files

The general format for initializing offline packet capture analysis is as follows:

```
zeek -r <pcap_file_location> <script_location>
```

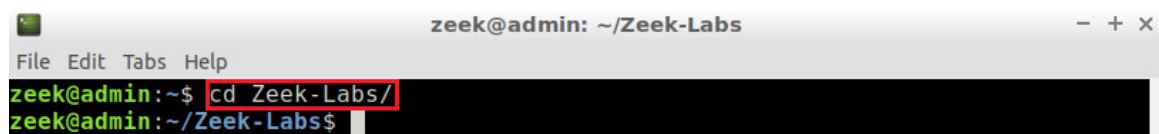
- `zeek`: command to invoke Zeek.
- `-r`: option signifies to Zeek that it will be reading from an offline file.
- `<pcap_file_location>`: indicates the pcap file location.
- `<script_location>`: indicates the script location.

3.1.2 Leveraging a script to detect brute force attacks present in a pcap file

Zeek installs a number of default scripts and trace files that can be used for testing purposes. In this section, we use the `bruteforce.pcap` as the input packet capture file and `ZeekBruteforceDetection.zeek` as the detection script. The packet capture file contains network traffic of a brute force password attack, while the script defines the brute forcing event for the Zeek event engine.

Step 1. Enter the lab workspace directory. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key.

```
cd Zeek-Labs/
```



Step 2. Initialize Zeek offline packet parsing on the packet capture file. It is possible to use the `Tab` key to autocomplete the longer paths.

```
zeek -C -r Sample-PCAP/bruteforce.pcap Lab-Scripts/ZeekDetectBruteForce.zeek
```

```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ zeek -C -r Sample-PCAP/bruteforce.pcap Lab-Scripts/ZeekDetectBruteForce.zeek
zeek@admin:~/Zeek-Labs$
```

The `-C` option is included to prevent Zeek from displaying specific warnings.

Step 3. After running the command, if a brute forcing attack was found, it will be logged in the `notice.log` output log file. We will use the `cat` command to view the file.

```
cat notice.log
```

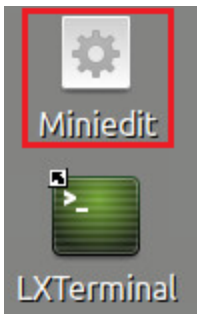
```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ cat notice.log
#separator \x09
#set separator ,
#empty_field (empty)
#unset_field -
#path notice
#open 2020-01-08-15-06-29
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p
p fluid file_mime_type file_desc proto note msg sub s
rc dst p n peer_descr actions suppress_for remote_l
ocation.country_code remote_location.region remote_location.city remote_l
ocation.latitude remote_location.longitude
#types time string addr port addr port string string string e
num enum string string addr addr port count string set[enum
] interval string string string double double
1389721084.522861 - - - - - -
- FTP::Bruteforcing 192.168.56.1 had 20 failed logins on 1 FTP serve
r in 0m37s - 192.168.56.1 - - - Notice::
ACTION_LOG 3600.000000 - - - - -
#close 2020-01-08-15-06-29
zeek@admin:~/Zeek-Labs$
```

Examining the preceding image, a possible brute force attack was detected. The log file shows that the IP address `192.168.56.1` had 20 or more failed login attempts on the hosted FTP server.

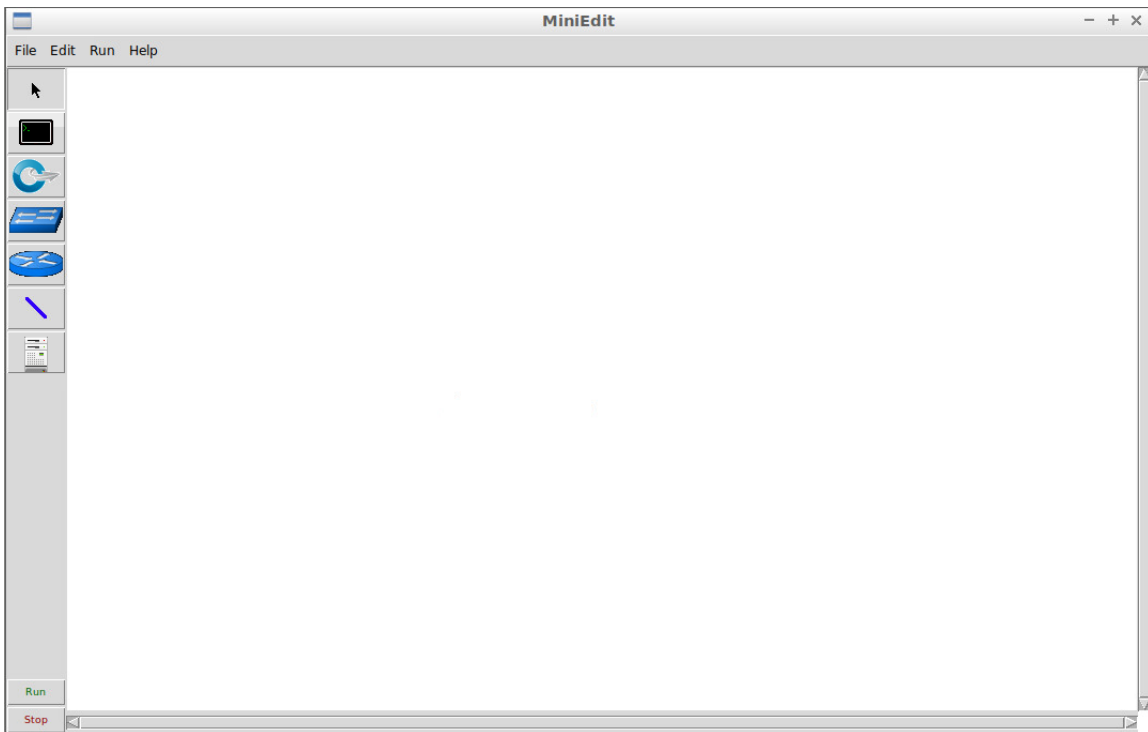
3.2 Launching Mininet

Mininet is a network emulator that creates a network topology consisting of virtual hosts, switches, controllers, and links. Within the Zeek lab series, we will be leveraging Mininet to generate and capture network traffic.

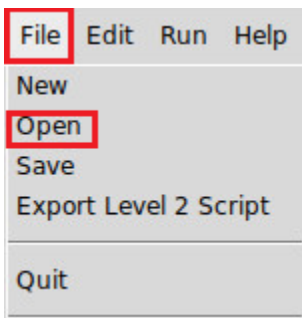
Step 1. From the *Client* machine's desktop, on the left side of the screen, click on the MiniEdit icon as shown below. When prompted for a password, type `password` and hit `Enter`. The MiniEdit editor will now launch.



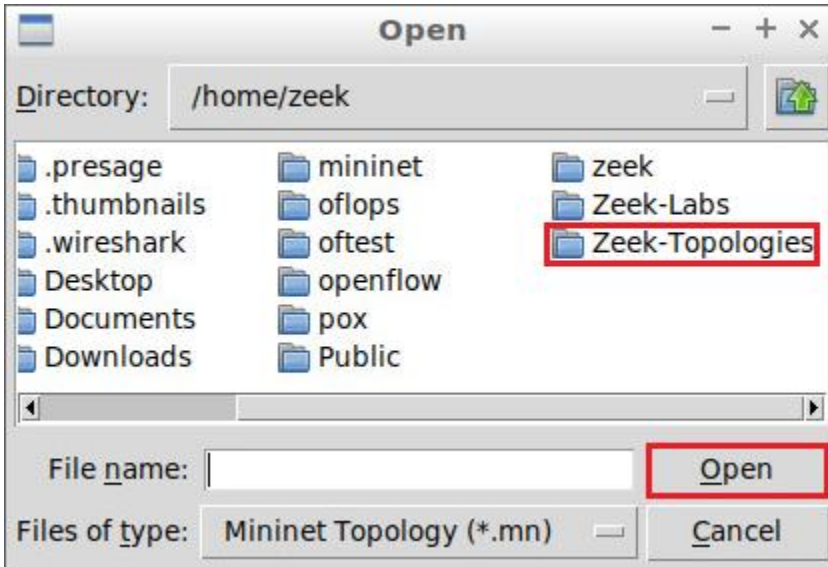
Step 2. The MiniEdit editor will now launch and allow for the creation of new, virtualized lab topologies. The image below shows the default MiniEdit display.



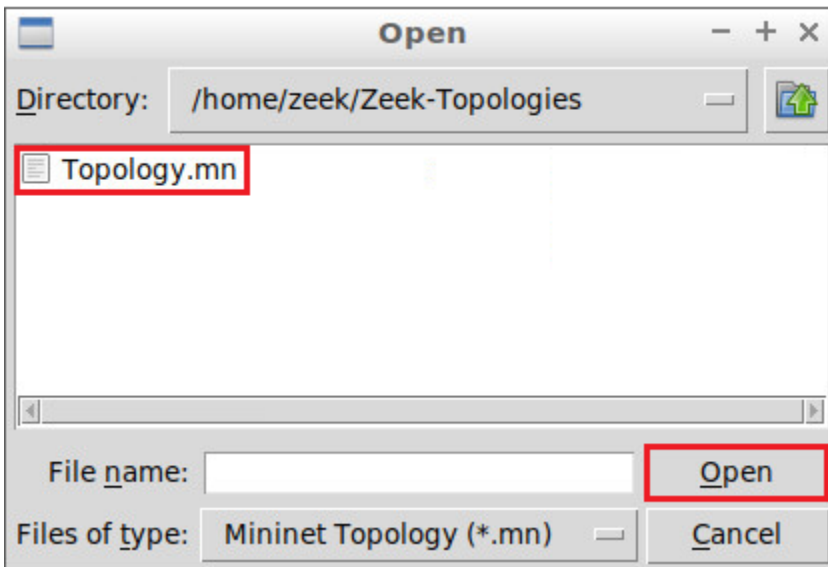
Step 3. A premade topology has already been created for this lab series. To load the correct topology, begin by clicking the **Open** button within the **File** tab on the top left of the MiniEdit editor.



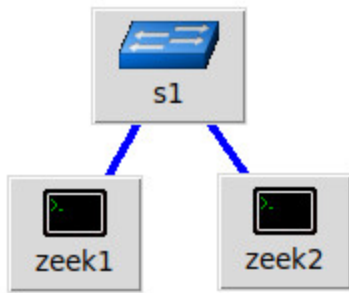
Step 4. Navigate to the Zeek-Topologies directory by scrolling to the right of the active directories and double clicking the Zeek-Topologies icon, or by clicking the **Open** button.



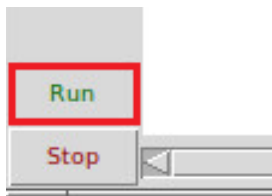
Step 5. Select the *Topology.mn* file by double clicking the *Topologies.mn* icon, or by clicking the Open button.



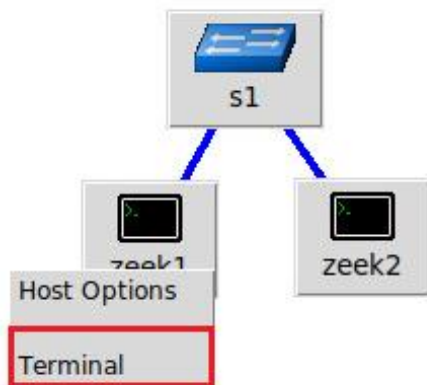
Step 6. The lab topology will contain two virtual machines – *zeek1* and *zeek2*, which are able to connect and communicate with one another through the *s1* switch, as seen in the image below.



Step 7. 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.



Step 8. To access either the `zeek1` or `zeek2` terminals for subsequent steps, hold the right mouse button on the desired machine, which will then display a `Terminal` button. Drag the cursor to the `Terminal` button to launch the terminal, as seen in the image below.



With the Mininet lab topology loaded, we can now begin to generate and analyze live network traffic capture.

3.3 Generating and analyzing live network traffic capture

The `tcpdump` command utility is a famous network packet analyzing tool that is used to display TCP/IP and other network packets being transmitted over the network⁴.

3.3.1 Leveraging the Tcpcmdump command utility

The general format for `tcpdump` is the following command:

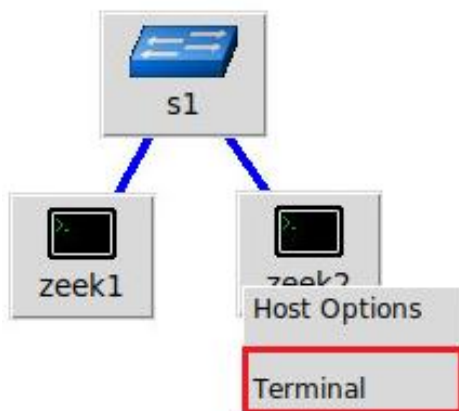
```
sudo tcpdump -i <interface_name> -s <num> -w <pcap_file_location>
```

- `sudo`: option to enable higher level privileges.
- `tcpdump`: program for capturing live network traffic.
- `-i`: option used to specify a network interface.
- `<interface_name>`: denotes the interface name.
- `-s`: option used to specify number of packets to capture.
- `<num>`: denotes the number of packets to capture. 0 equals infinite.
- `-w`: option used to specify that we will be writing to a new file.
- `<pcap_file_location>`: indicates the file location.

3.3.2 Capturing live network traffic

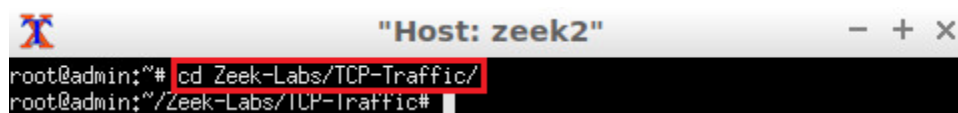
The *zeek2* machine will be used to capture live network traffic, while the *zeek1* machine will be used to generate live network traffic.

Step 1. Open the *zeek2* Terminal by hold the right mouse button on the desired machine, which will then display a `Terminal` button. Drag the cursor to the the `Terminal` button to launch the terminal, as seen in the image below.



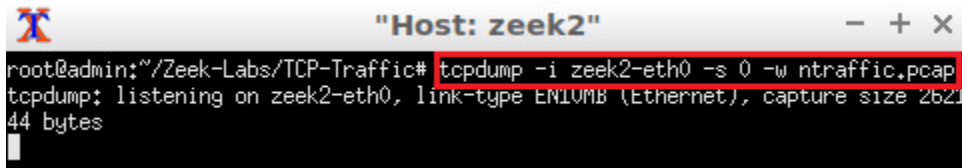
Step 2. Navigate to the TCP-Traffic directory. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key.

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



Step 3. Use the following command to begin live packet capture. If the Terminal session has not been terminated or closed, you may not be prompted to enter the password. If prompted for a password, type `password` and hit `Enter`. Live packet capture will start on interface `zeek2-eth0`.

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

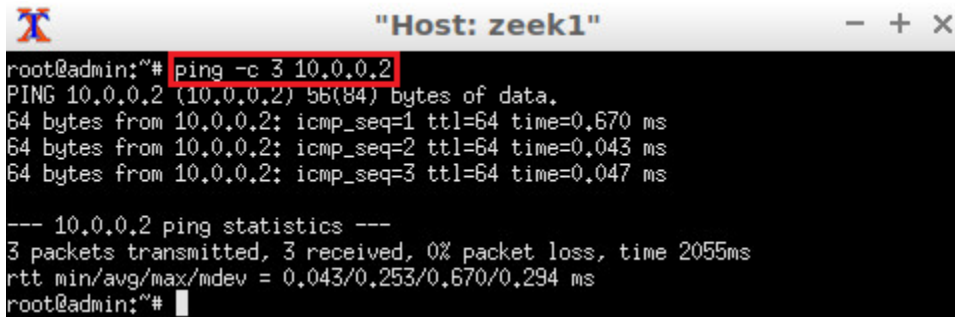


Step 4. Minimize the `zeek2` Terminal and open the `zeek1` Terminal. If necessary, right click within the Miniedit editor to activate your cursor.



Step 5. Generate traffic by using the `ping` utility. `Ping` operates by sending Internet Control Message Protocol (ICMP) echo request packets to the target host and waiting for an ICMP echo reply. Issue the following command on the newly opened `zeek1` Terminal.

```
ping -c 3 10.0.0.2
```

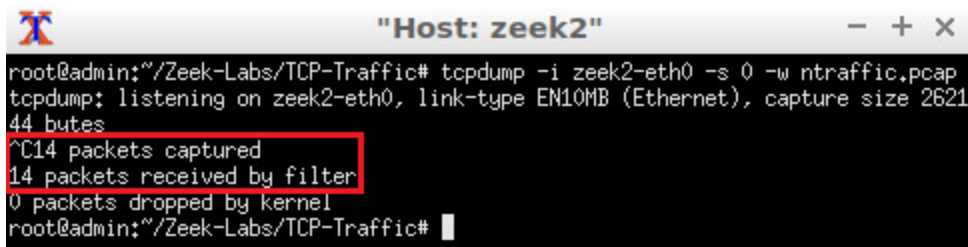


The `-c` option is used to indicate the number of packets to send – in this example, 3 packets.

Step 5. 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 6. Use the `Ctrl+c` key combination to stop live traffic capture. Statistics of the capture session will be displayed. 14 packets were recorded by the interface, which were then captured and stored in the new `ntraffic.pcap` file.

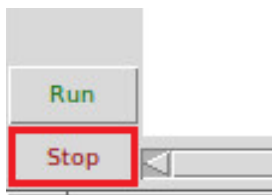


```

root@admin:~/Zeek-Labs/TCP-Traffic# tcpdump -i zeek2-eth0 -s 0 -w ntraffic.pcap
tcpdump: listening on zeek2-eth0, link-type EN10MB (Ethernet), capture size 2621
44 bytes
^C14 packets captured
14 packets received by filter
0 packets dropped by kernel
root@admin:~/Zeek-Labs/TCP-Traffic#

```

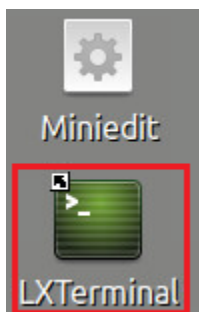
Step 7. Stop the current Mininet session by clicking the `Stop` button on the bottom left of the MiniEdit editor, and close the MiniEdit editor by clicking the `✕` on the top right of the editor.



We will now return to the *Client* machine to process and analyze the newly generated network traffic.

3.3.3 Analyzing the newly captured network traffic

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 *ntraffic.pcap* file. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key.



```

cd Zeek-Labs/TCP-Traffic/
zeek@admin: ~/Zeek-Labs/TCP-Traffic
zeek@admin:~$ cd Zeek-Labs/TCP-Traffic/
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Step 3. View the file contents of the *TCP-Traffic* directory.

```
ls
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
ntraffic.pcap
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

We can see the *ntraffic.pcap* file that was generated by the *zeek2* machine is now accessible.

Step 4. Initialize Zeek offline packet parsing on the packet capture file. The `-r` option is used to read from a given pcap file, and the `-C` option is used to disable checksums validation.

```
zeek -C -r ntraffic.pcap
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek -C -r ntraffic.pcap
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

Step 5. View the newly generated Zeek log files.

```
ls
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
conn.log ntraffic.pcap packet_filter.log
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

The generated log files will contain important information regarding the network traffic. For instance, the *conn.log* file will contain connection-based information, specifically the hosts communicating, their IP addresses, protocols and ports. The following labs will offer in-depth insight and examples towards understanding these Zeek log files.

Step 6. Viewing the *conn.log* connection-based log file with the `cat` command, we can see that the IP address *10.0.0.1* was detected to generate the captured traffic, corresponding to the *zeek1* virtual machine.

```
cat conn.log
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ cat conn.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn
#open 2020-01-08-15-14-09
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p
p proto service duration orig_bytes resp_bytes conn_state
te local_orig local_resp missed_bytes history orig_pkts
rig_ip_bytes resp_pkts resp_ip_bytes tunnel_parents
#types time string addr port addr port enum string interval
count count string bool bool count string count count count c
ount set[string]
1578514297.655355 CIGSlc4sxnBAyFxfcd 10.0.0.1 8 10.0.0.2
0 icmp - 2.054924 168 168 0TH - - 0
- 3 252 3 252 - - - - -

```

Step 7. 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
```

```

zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin:~/usr/local/zeek/bin$

```

The above command navigates to Zeek’s installation directory and executes the stop command in `zeekctl`.

Concluding this lab, we have reviewed Zeek’s architecture and event-based engine, as well as introduced both offline and live network traffic capture.

References

1. “Zeek documentation”, [Online]. Available: <https://docs.zeek.org/en/stable/intro/index.html>
2. Sommer, Robin, and Vern Paxson, “Enhancing byte-level network intrusion detection signatures with context,” *In Proceedings of the 10th ACM conference on Computer and communications security*, pp. 262-271. ACM, 2003.
3. “Signature-based intrusion detection”, [Online]. Available: <http://www.cs.unc.edu/~jeffay/courses/nidsS05/slides/6-Sig-based-Detection.pdf>.
4. Joseph, D. A., Paxson, V., & Kim, S. (2006). tcpdump Tutorial. University of California, EE122 Fall.



The University of Texas at San Antonio™

The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 2: An Overview of Zeek Logs

Document Version: **03-13-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to Zeek Logs	4
1.1 Zeek Logs generated by packet analysis	4
1.2 Zeek Logs generated by recurrent network analysis	4
1.3 Typical uses of Zeek Logs	5
2 Starting a new instance of Zeek.....	5
3 Parsing packet capture files into Zeek log files.....	6
3.1 Overview of Zeek command options	7
3.2 Using Zeek to process offline packet capture files	7
3.3 Understanding Zeek log files	8
3.4 Basic viewing of Zeek logs	9
4 Analyzing Zeek log files	10
4.1 Leveraging zeek-cut for a more refined view of log files	10
4.1.1 Using zeek-cut in conjunction with cat and head command utilities.....	10
4.1.2 Printing the output of zeek-cut to a text file	12
4.1.3 Printing the output of zeek-cut to a csv file	12
4.2 Closing the current instance of Zeek.....	13
References	14

Overview

This lab covers Zeek's logging files. Zeek's event-based engine will generate log files based on signatures or events found during network traffic analysis. The focus in this lab is on explaining each logging file and introducing some basic analytic functions and tools.

Objectives

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

1. Generate Zeek log files.
2. Use Linux Terminal tools combined with Zeek's *zeek-cut* utility to customize the output of logs files for analysis.

Lab topology

Figure 1 displays the topology of the lab. This lab will primarily use the *Client* machine for offline packet capture processing and analysis.

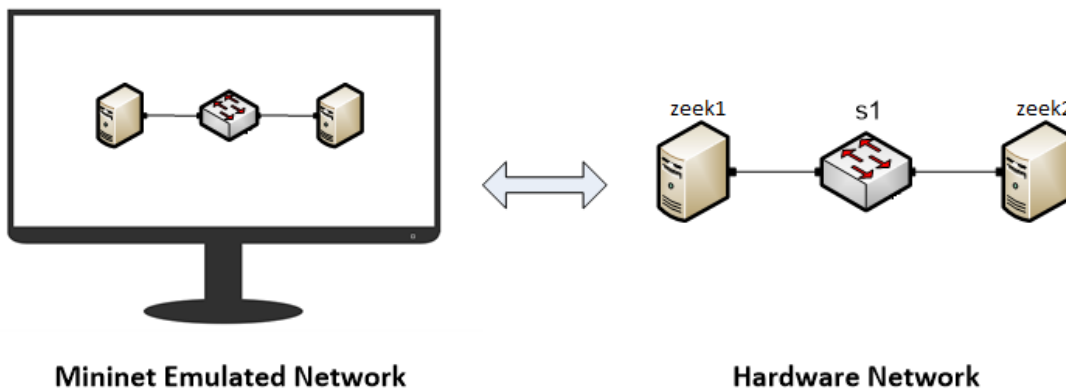


Figure 1. Lab topology.

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: Introduction to Zeek logs.
2. Section 2: Starting a new instance of Zeek.
3. Section 3: Parsing packet capture files into Zeek log files.
4. Section 4: Analyzing Zeek log files.

1 Introduction to Zeek Logs

Zeek's generated log files include a comprehensive record of every connection seen on the wire; this includes application-layer protocols and fields (e.g., Hyper-Text Transfer Protocol (HTTP) sessions, Uniform Resource Locator (URL), key headers, Multi-Purpose Internet Mail Extensions (MIME) types, server responses, etc.), Domain Name Server (DNS) requests and responses, Secure Socket Layer (SSL) certificates, key content of Simple Mail Transfer Protocol (SMTP) sessions, and others.

1.1 Zeek Logs generated by packet analysis

A Zeek log is a stream of high-level entries that correspond to network activities, such as a login to SSH or an email sent using SMTP. In Zeek, each event stream has a dedicated file with its own set of features, fields, or columns.

During capture or analysis, Zeek generates a log determined by the protocol type. Due to this architecture, a Session Initiation Protocol (SIP) log for instance, does not contain any other protocols' packets information like HTTP. Furthermore, each log file contains case-relative fields (e.g., *from* and *subject* fields in an SMTP log). Some of these log files are large and contain entries that can be either benign or malicious, whereas others are smaller and contain more actionable information.

1.2 Zeek Logs generated by recurrent network analysis

With every session of packet analysis, either through live packet analysis or the parsing of an offline packet capture file, Zeek generates session-specific log files. In addition to these session-based log files, Zeek creates network-reliant log files as well. These network-reliant files are continually generated and updated when a new session is initialized and started.

The following Zeek log files are updated daily:

- *known_hosts.log*: Log file containing information for hosts that completed TCP handshakes.
- *known_services.log*: Log file containing a list of services running on hosts.
- *known_certs.log*: Log file containing a list of Secure Socket Layer (SSL) certificates.
- *software.log*: Log file containing information about Software being used on the network.

Additionally, a list of detection-based log files is created during each session. The log files relevant to this lab are:

- *notice.log* (Zeek notices): When Zeek detects an anomaly, a corresponding notice will be raised in this file.
- *intel.log* (Intelligence data matches): When Zeek detects traffic flagged with known malicious indicators, a corresponding reference will be logged in this file.
- *signatures.log* (Signature matches): When Zeek detects traffic flagged with known malicious or faulty packet signatures, a corresponding reference will be logged in this file.

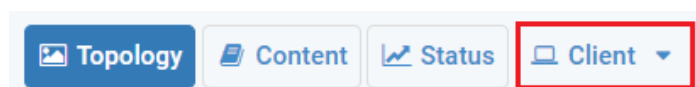
1.3 Typical uses of Zeek Logs

By default, Zeek logs all information into well-structured, tab-separated text files suitable for postprocessing. Users can also choose from a set of alternative output formats and backends such as external databases.

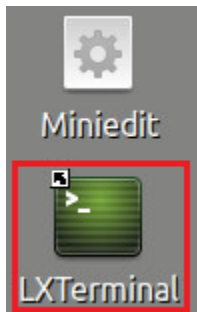
The Zeek-native `zeek-cut` utility can be leveraged to further specify and parse the information within the generated log files.

2 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. When prompted for a password, type `password` and hit `Enter`. 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 start
```

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

3 Parsing packet capture files into Zeek log files

In this section we introduce Zeek's capability of generating and viewing log files. Packet capture files used in this lab are preloaded onto the *Client* machine, and can be found with the following path:

```
Zeek-Labs/Sample-PCAP/
```

These packet capture files were downloaded from `Tcp replay's` sample capture collection. 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.

```
http://tcpreplay.appneta.com/wiki/captures.html
```

`Tcpreplay` is a suite of free Open Source utilities for editing and replaying previously captured network traffic and can be used to test transmissions and network health.

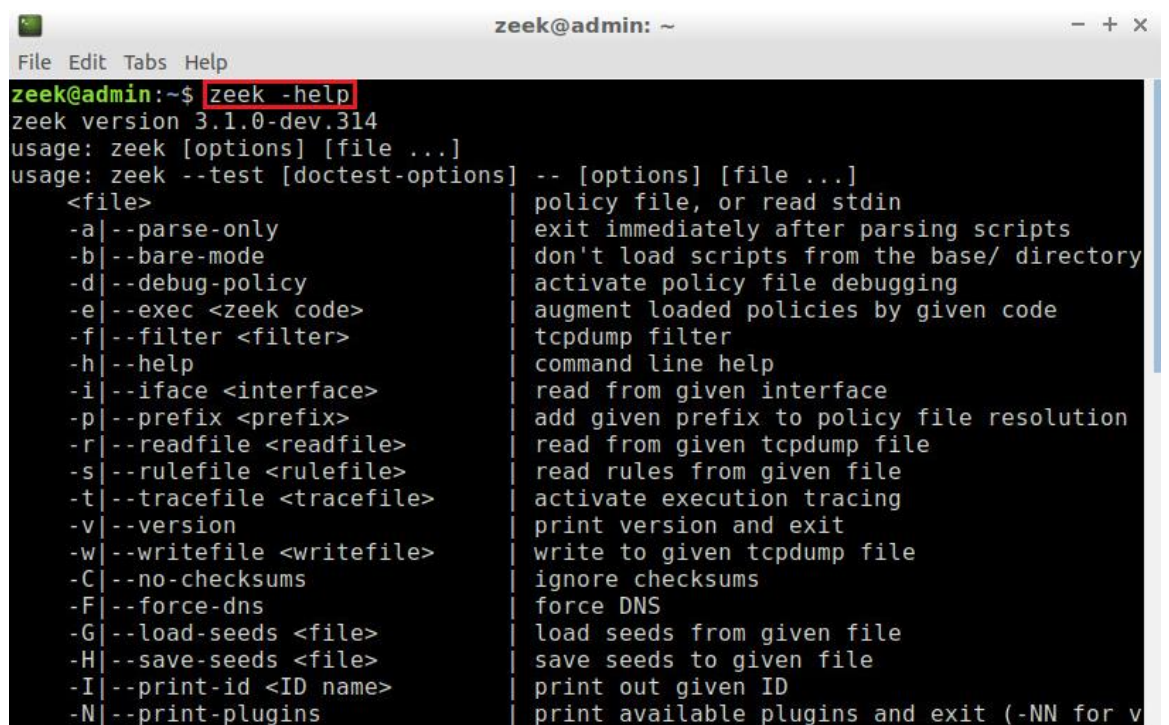
3.1 Overview of Zeek command options

When using Zeek, the user specifies a running state option. In this lab, three primarily options are used:

- `-C`: specifies to ignore checksum warnings, specifically to avoid redundancy since we are focusing on TCP traffic only.
- `-r`: specifies offline packet capture file analysis.
- `-w`: specifies live network capture.

Additional Zeek options can be found by passing the `-help` option to the `zeek` command:

```
zeek -help
```



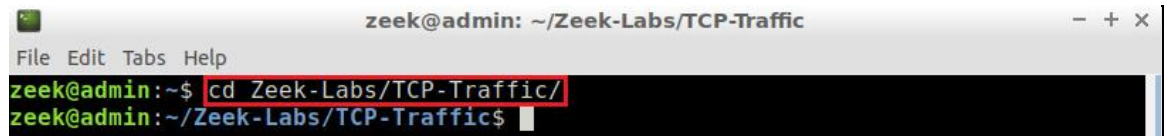
```
zeek@admin:~$ zeek -help
zeek version 3.1.0-dev.314
usage: zeek [options] [file ...]
usage: zeek --test [doctest-options] -- [options] [file ...]
<file> | policy file, or read stdin
-a|--parse-only | exit immediately after parsing scripts
-b|--bare-mode | don't load scripts from the base/ directory
-d|--debug-policy | activate policy file debugging
-e|--exec <zeek code> | augment loaded policies by given code
-f|--filter <filter> | tcpdump filter
-h|--help | command line help
-i|--iface <interface> | read from given interface
-p|--prefix <prefix> | add given prefix to policy file resolution
-r|--readfile <readfile> | read from given tcpdump file
-s|--rulefile <rulefile> | read rules from given file
-t|--tracefile <tracefile> | activate execution tracing
-v|--version | print version and exit
-w|--writefile <writefile> | write to given tcpdump file
-C|--no-checksums | ignore checksums
-F|--force-dns | force DNS
-G|--load-seeds <file> | load seeds from given file
-H|--save-seeds <file> | save seeds to given file
-I|--print-id <ID name> | print out given ID
-N|--print-plugins | print available plugins and exit (-NN for v
```

3.2 Using Zeek to process offline packet capture files

In this subsection we will use Zeek to process the existing offline packet capture file `smallFlows.pcap`. By specifying the `-r` option and the directory path to the pcap file, Zeek can generate the corresponding log files.

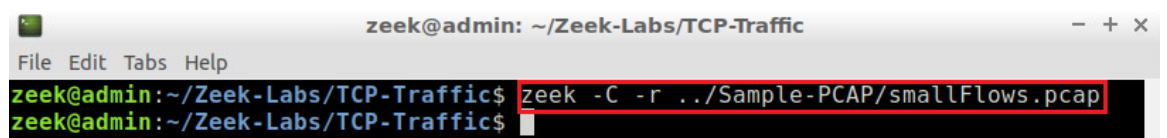
Step 1. Navigate to the lab workspace directory. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key.

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



Step 2. Use the following command to process the *smallFlows.pcap* file. It is possible to use the `tab` key to autocomplete the longer paths.

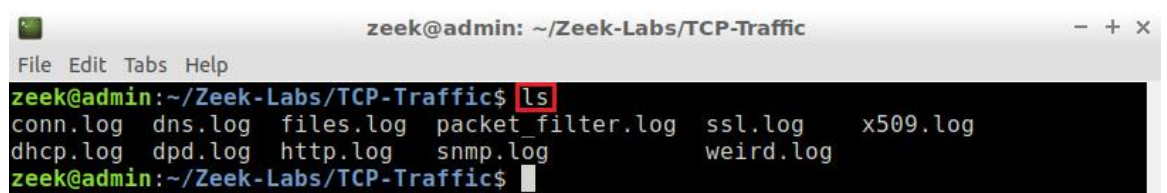
```
zeek -C -r ../Sample-PCAP/smallFlows.pcap
```



After Zeek finishes processing the packet capture file, it will generate a number of log files.

Step 3. Use the following command to list the generated log files.

```
ls
```



3.3 Understanding Zeek log files

Zeek's generated log files can be summarized as follows:

- *conn.log*: A file containing information pertaining to all TCP/UDP/ICMP connections, this file contains most of the information gathered from the packet capture.
- *files.log*: A file consisting of analytic results of packets' counts and sessions' durations.
- *packet_filter.log*: A file listing the active filters applied to Zeek upon reading the packet capture file.
- *x509.log*: A file containing public key certificates used by protocols.

- *weird.log*: A file containing packet data non-conformant with standard protocols. It also contains packets with possibly corrupted or damaged packet header fields.
- *(protocol).log* (*dns.log*, *dhcp.log*, *http.log*, *snmp.log*): These are files containing information for packets found in each respective protocol. For instance, *dns.log* will only contain information generated by Domain Name Service (DNS) packets.

More information regarding log files is available in the Zeek official documentation, which can be viewed online using an external Internet-connected machine through this link:

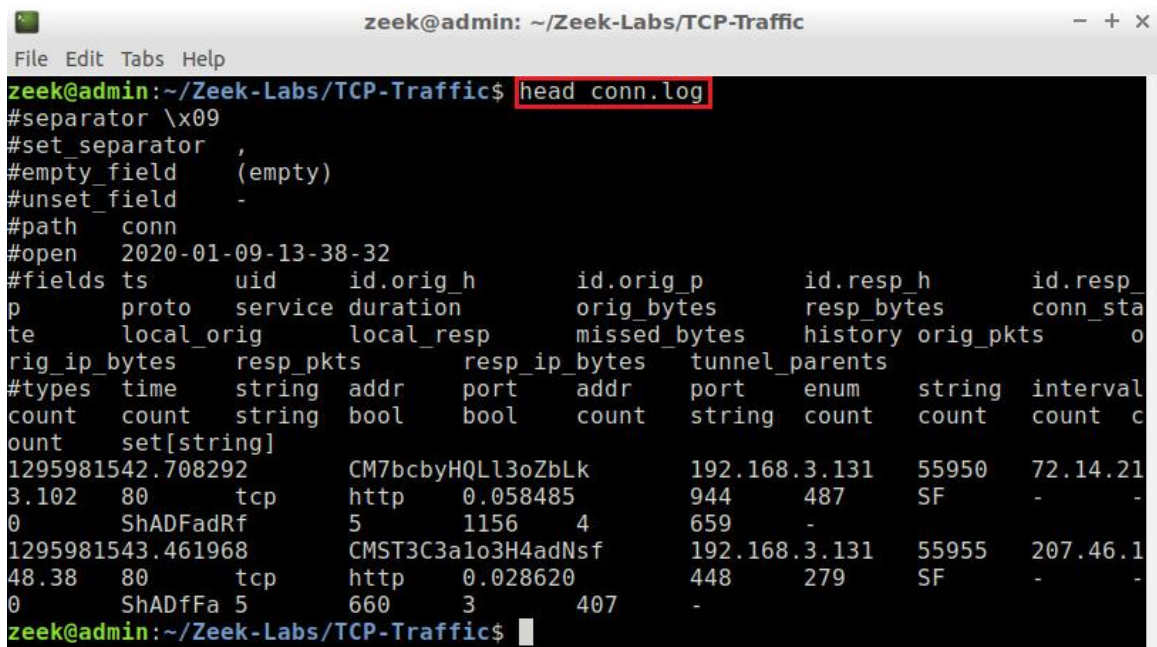
```
https://docs.zeek.org/en/stable/script-reference/log-files.html
```

3.4 Basic viewing of Zeek logs

In this subsection we examine the generated log files and their contents.

Step 1. Use the following command to display the contents of the *conn.log* file using the `head` command.

```
head conn.log
```



```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ head conn.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn
#open 2020-01-09-13-38-32
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_
p proto service duration orig_bytes resp_bytes conn_sta
te local_orig local_resp missed_bytes history orig_pkts o
rig_ip_bytes resp_pkts resp_ip_bytes tunnel_parents
#types time string addr port addr port enum string interval
count count string bool bool count string count count count c
ount set[string]
1295981542.708292 CM7bcbyHQLl3oZbLk 192.168.3.131 55950 72.14.21
3.102 80 tcp http 0.058485 944 487 SF - -
0 ShADfAdRf 5 1156 4 659 - -
1295981543.461968 CMST3C3a1o3H4adNsf 192.168.3.131 55955 207.46.1
48.38 80 tcp http 0.028620 448 279 SF - -
0 ShADfFa 5 660 3 407 - -
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

The topmost rows within the *conn.log* file will be displayed in the Terminal; however, the current formatting wraps around multiple lines, making it unclear and hard to understand. In the following section we introduce the `zeek-cut` utility for enhancing the output of these log files.

4 Analyzing Zeek log files

In this section, we review the utilities that help in displaying log files with well-formatted outputs, as well as saving output to text files.

4.1 Leveraging zeek-cut for a more refined view of log files

Although the produced log file is tab delimited, it is difficult to visualize and parse information from the terminal. The `zeek-cut` utility can be used to parse the log files by specifying which column data to be displayed in a more organized output.

4.1.1 Using zeek-cut in conjunction with cat and head command utilities

Generally, the `zeek-cut` utility is typically coupled with `cat` using the `pipe |` command. In Linux, the `pipe` command sends the output of one command as input to another. Essentially, the output of the left command is passed as input to that on its right, and multiple commands can be chained together.

Step 1. Use the following command to pipe the contents of `cat` into `zeek-cut`.

```
cat conn.log | zeek-cut id.orig_h id.orig_p id.resp_h id.resp_p
```

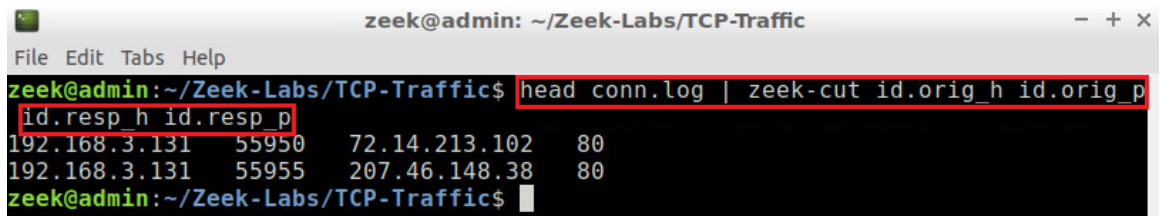
The options passed into the `zeek-cut` utility represent the column headers to be extracted from the log file:

- `id.orig_h`: Column containing the source IP address.
- `id.orig_p`: Column containing the source port.
- `id.resp_h`: Column containing the destination IP address.
- `id.resp_p`: Column containing the destination port.

Alternatively, instead of using the `cat` command, the `head` command can be used to display the topmost rows of the log file, which can be very useful to view a large file's contents.

Step 2. Use the following command to pipe the contents of `head` into `zeek-cut`.

```
head conn.log | zeek-cut id.orig_h id.orig_p id.resp_h id.resp_p
```

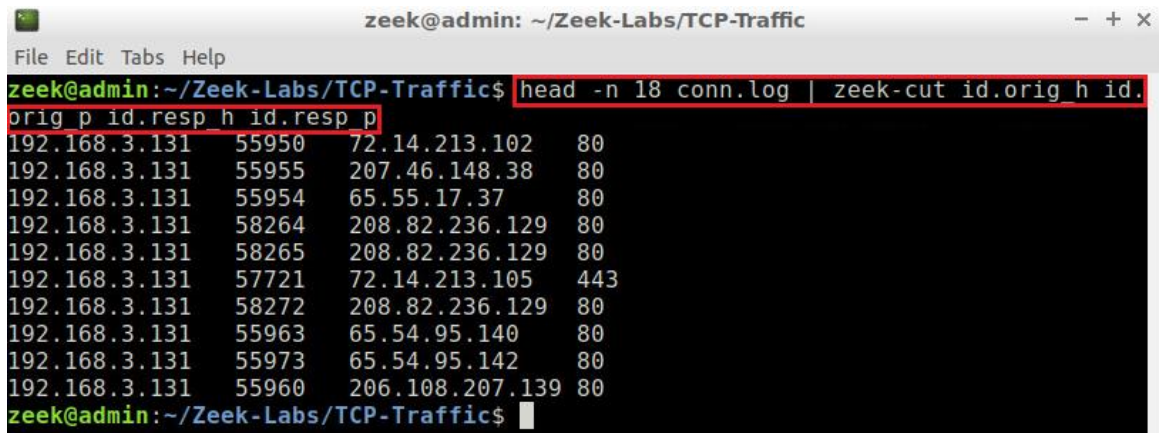


Notice that only two records are shown. This is caused by the `head` command taking the 10 topmost rows of `conn.log`, regardless of what that entails, and passing it as input to `zeek-cut`.

Since the log file contains 8 lines of header padding used for displaying the file's format, we will have to specify the first 18 rows of file in order to successfully display the first 10 packets of the log file.

Step 3. Use the following command to pipe the contents of `head` into `zeek-cut`.

```
head -n 18 conn.log | zeek-cut id.orig_h id.orig_p id.resp_h id.resp_p
```



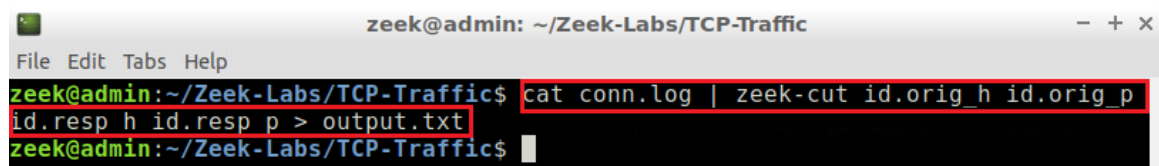
The `-n` option can be passed to the `head` utility to specify the desired number of rows.

4.1.2 Printing the output of zeek-cut to a text file

While the results displayed in the Terminal after using the `zeek-cut` utility can be easily viewed for smaller datasets, it is often necessary to save the output into a separate file. Using the `>` character, we can send the output to a new file for further processing by other applications.

Step 1. Use the following command to change the output location of `zeek-cut`.

```
cat conn.log | zeek-cut id.orig_h id.orig_p id.resp_h id.resp_p > output.txt
```

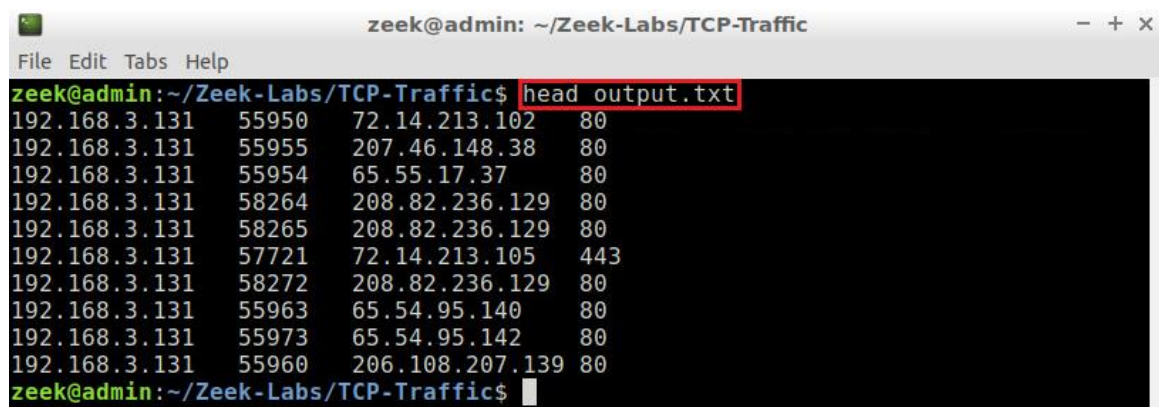


```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ cat conn.log | zeek-cut id.orig_h id.orig_p
id.resp_h id.resp_p > output.txt
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

By including the file extension in `output.txt`, we are choosing to print the output into a plain text file.

Step 2. We can display the topmost contents of the new `output.txt` file by using the `head` command.

```
head output.txt
```



```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ head output.txt
192.168.3.131 55950 72.14.213.102 80
192.168.3.131 55955 207.46.148.38 80
192.168.3.131 55954 65.55.17.37 80
192.168.3.131 58264 208.82.236.129 80
192.168.3.131 58265 208.82.236.129 80
192.168.3.131 57721 72.14.213.105 443
192.168.3.131 58272 208.82.236.129 80
192.168.3.131 55963 65.54.95.140 80
192.168.3.131 55973 65.54.95.142 80
192.168.3.131 55960 206.108.207.139 80
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

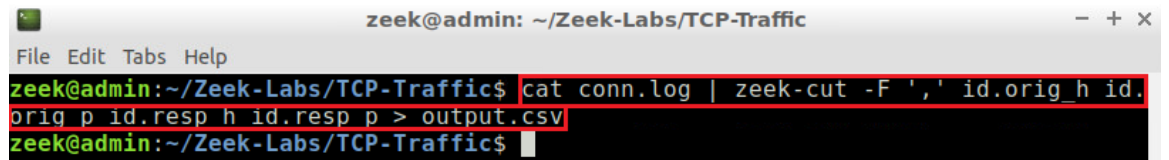
The `output.txt` file contains the same tab-delimited format as shown in previous `zeek-cut` examples.

4.1.3 Printing the output of zeek-cut to a csv file

In some situations, it is helpful to save the output of `zeek-cut` in a csv file. In a csv file, data may be imported into other applications, such as databases or machine learning classifiers.

Step 1. The exported output file by `zeek-cut` is tab-delimited due to the default `zeek-cut` settings. To export a file with another delimiter, the `-F` option is used.

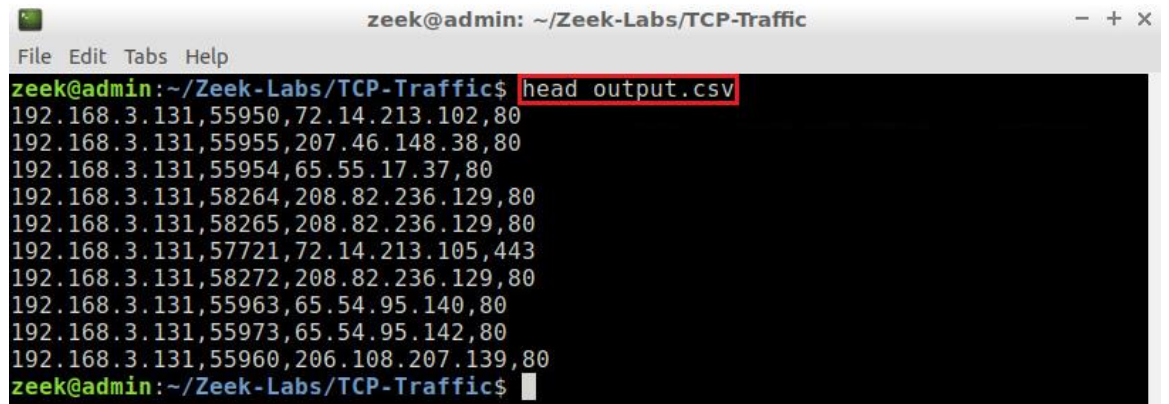
```
cat conn.log | zeek-cut -F ',' id.orig_h id.orig_p id.resp_h id.resp_p > output.csv
```



```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ cat conn.log | zeek-cut -F ',' id.orig_h id.orig_p id.resp_h id.resp_p > output.csv
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

Step 4. We can now display the topmost contents of the `output.csv` file.

```
head output.csv
```



```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ head output.csv
192.168.3.131,55950,72.14.213.102,80
192.168.3.131,55955,207.46.148.38,80
192.168.3.131,55954,65.55.17.37,80
192.168.3.131,58264,208.82.236.129,80
192.168.3.131,58265,208.82.236.129,80
192.168.3.131,57721,72.14.213.105,443
192.168.3.131,58272,208.82.236.129,80
192.168.3.131,55963,65.54.95.140,80
192.168.3.131,55973,65.54.95.142,80
192.168.3.131,55960,206.108.207.139,80
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

As shown in the image, the `output.csv` file is in a comma-delimited format, rather than the previous tab-delimited format.

In conclusion, `zeek-cut` is a flexible tool that can be called to format Zeek log files depending on the user's needs. The `zeek-cut` utility can be utilized with more advanced commands to further increase customization.

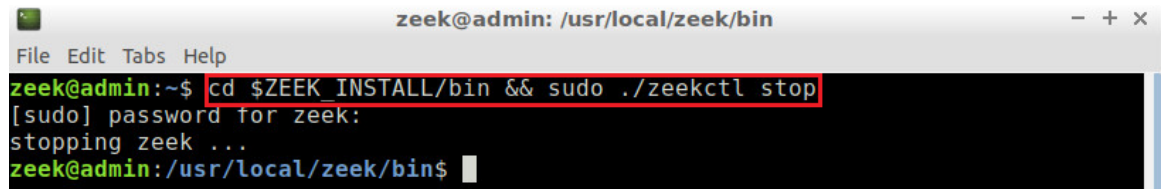
4.2 Closing the current instance of Zeek

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
```



```
zeek@admin:~/zeek/bin$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin:~/zeek/bin$
```

Concluding this lab, we have reviewed Zeek’s output log files in more depth while introducing some of the more relevant network-based log files and introduced some basic utilities to view these log files.

References

1. “Log files”, Zeek user manual, [Online]. Available: [Zeek, docs.zeek.org/en/stable/script-reference/log-files.html](https://docs.zeek.org/en/stable/script-reference/log-files.html).
2. “Sample captures”, *Tcpreplay*, [Online]. Available: tcpreplay.appneta.com/wiki/captures.html



The University of Texas at San Antonio™

The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION

Lab 3: Parsing, Reading and Organizing Zeek Log Files

Document Version: **03-13-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to shell scripts	4
1.1 Ubuntu Linux text editors	4
1.2 Creating a shell script	5
2 Advanced zeek-cut log file analysis	7
2.1 Example 1	8
2.2 Example 2	9
2.3 Example 3	10
2.4 Example 4	12
3 Incorporating the AWK scripting language for log file analysis	13
3.1 Example 1	13
3.2 Example 2	15
3.3 Example 3	16
3.4 Closing the current instance of Zeek.....	18
References	18

Overview

This lab explains how to format and organize Zeek's log files by combining zeek-cut utility with basic Linux shell commands. Utilities and tools introduced in this lab provide practical examples for logs customization in a real network environment.

Objectives

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

1. Use Linux tools and commands for text files processing.
2. Practice Linux shell scripts and the AWK scripting language.
3. Incorporate AWK with zeek-cut to provide formatted logs.

Lab topology

Figure 1 shows the lab workspace topology. This lab primarily uses the *Client* machine for offline packet capture processing and analysis.

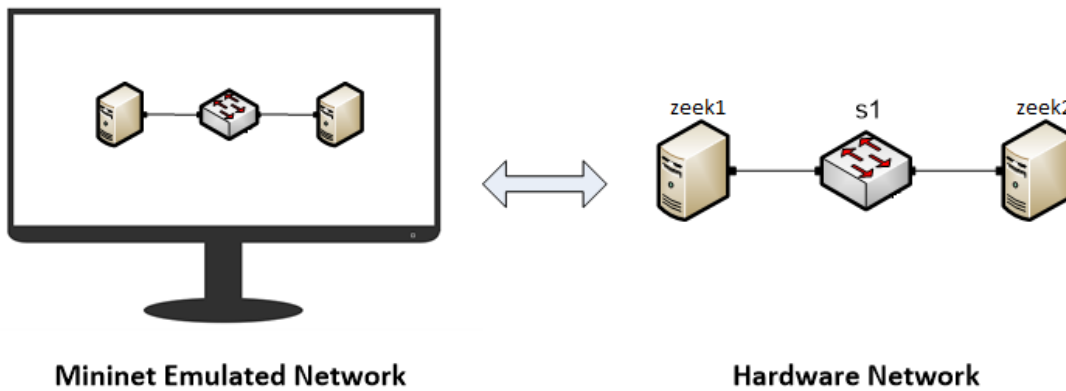


Figure 1. Lab topology.

Lab settings

The information (case-sensitive) in the table below provides the credentials 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: Introduction to shell scripts.
2. Section 2: Advanced zeek-cut log file analysis.
3. Section 3: Incorporating the AWK scripting language for log file analysis.

1 Introduction to shell scripts

A shell script is a text file containing commands to be executed by the Unix command-line interpreter. Shell scripts provide a convenient way to manipulate files and automate programs' executions. Selection and repetition are incorporated into scripts to branch control based on conditioning and looping statements. Running a shell script can immensely save time and prevent manually entering repetitive commands in recurrent tasks.

1.1 Ubuntu Linux text editors

Linux-based distributions include pre-installed text editors like `nano`, `vi`, `vim`, `gedit`, etc. `nano` is a keyboard-oriented lightweight text editor with a simple Command Line Interface (CLI). Other editors such as `vi` and `vim` are highly customizable and extensible, making them attractive for users that demand a large amount of control and flexibility over their text editing environment. Alternatively, the Graphical User Interface (GUI) text editor `gedit` can be used to visually work outside of the terminal. More information on these text editors can be found on the Ubuntu help pages. To access the following links, users must have access to an external computer connected to the Internet, because the Zeek Lab topology does not have an active Internet connection.

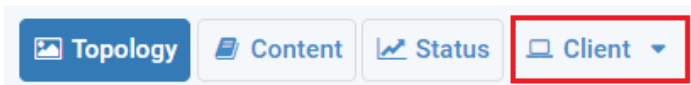
- `Nano` – <https://help.ubuntu.com/community/Nano>
- `Vim` – <https://help.ubuntu.com/community/VimHowto>
- `Gedit` – <https://help.ubuntu.com/community/gedit>

For simplicity, in this lab we use `nano` text editor to view, create and edit text files.

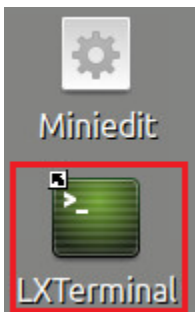
1.2 Creating a shell script

Shell scripts are effective in executing repetitive terminal commands. Unlike executing commands manually in the terminal, scripts can be saved and executed whenever needed simple by invoking their names. We will begin this lab by writing some basic shell scripts.

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



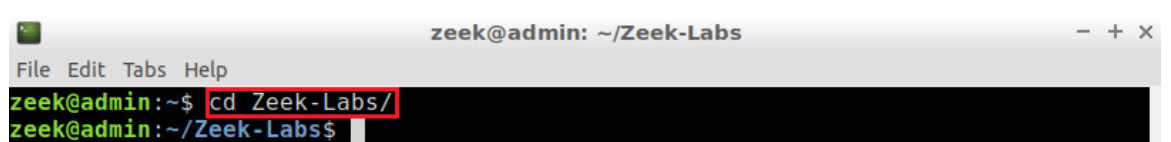
Step 2. The *Client* machine will now open, and the desktop will be displayed. On the left side of the screen, click on the LXTerminal icon as shown below.



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

Step 4: In the Linux terminal, navigate to the lab workspace directory by typing the following command:

```
cd Zeek-Labs/
```

A terminal window titled 'zeek@admin: ~/Zeek-Labs'. The terminal shows the command 'cd Zeek-Labs/' being entered and executed. The prompt changes from 'zeek@admin:~\$' to 'zeek@admin:~/Zeek-Labs\$'. The command and the new prompt are highlighted with red boxes.

Step 3: Use the `nano` text editor to create the `lab3script.sh` file.

```
sudo nano lab3script.sh
```

```

zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ sudo nano lab3script.sh
[sudo] password for zeek:
zeek@admin:~/Zeek-Labs$

```

Step 4: Edit the *lab3script.sh* file contents.

Once the text editor has opened, we will be able to enter the following commands. Each new line will denote a new Terminal command being passed. 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 start
cd Zeek-Labs/TCP-Traffic/
zeek -C -r ../Sample-PCAP/smallFlows.pcap

```

```

zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
GNU nano 2.9.3 lab3script.sh
cd $ZEEK_INSTALL/bin
sudo ./zeekctl start
cd Zeek-Labs/TCP-Traffic/
zeek -C -r ../Sample-PCAP/smallFlows.pcap

```

The file's content is explained as follows:

- Line 1: changes the current directory to the Zeek's installation directory.
- Line 2: starts a new instance of Zeek through `zeekctl`.
- Line 3: changes the current directory to the lab workspace.
- Line 4: invokes the `zeek` command with the `-r` option to begin processing the *smallFlows.pcap* capture file located in the *Sample-PCAP* directory.

Step 5: When using `Nano`, the following keyboard shortcuts are used to save a file and then exit the workspace.

- `CTRL + o` – save the file
- `CTRL + x` – save and exit the file, return to terminal

After completing Step 4 and adding the correct commands with proper formatting, we will save and exit the text editor. Press `CTRL + o` and hit `Enter` to save the file's contents, then `CTRL + x` to exit `nano` and return to the terminal.

Step 6: Use the following command to modify the permissions of the script file to make it executable. When prompted for a password, type `password` and hit `Enter`.

```

sudo chmod +x lab3script.sh

```



```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ sudo chmod +x lab3script.sh
[sudo] password for zeek:
zeek@admin:~/Zeek-Labs$
```

Step 7: Execute the *lab3script.sh* shell script by typing the following command.

```
./lab3script.sh
```

```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ ./lab3script.sh
starting zeek ...
zeek@admin:~/Zeek-Labs$
```

Step 8: Navigate to the lab workspace directory.

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

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ cd Zeek-Labs/TCP-Traffic/
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

Step 9: Verify that the *smallFlows.pcap* file was processed successfully.

```
ls
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
conn.log  dns.log  files.log  packet_filter.log  ssl.log  x509.log
dhcp.log  dpd.log  http.log  snmp.log           weird.log
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

The above output shows the list of log files generated by Zeek's processing, verifying that the script executed without errors.

2 Advanced zeek-cut log file analysis

This section introduces more advanced `zeek-cut` functionality to analyze packet capture statistics. These statistics can be used for planning and anomaly analysis. For instance, if a single port has been targeted and received a large number of network traffic, it may highlight a possible vulnerability. We can use the `zeek-cut` utility to determine if a host sends an abnormal number of packets to a specific destination and further analyze this event.

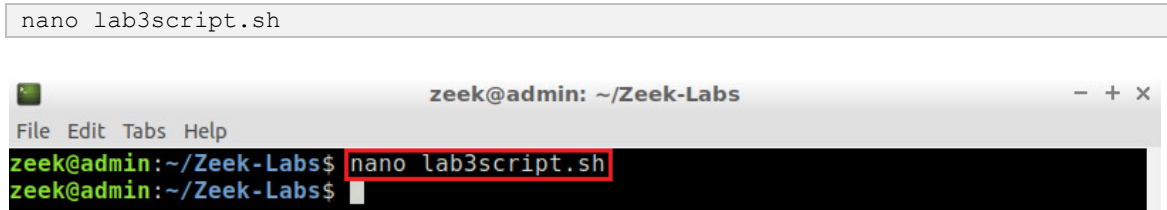
2.1 Example 1

Example 1: Show the 10 source IP addresses that generated the most network traffic, organized in descending order.

To solve this example, we will be looking at the `id.orig_h` column because it contains the source IP addresses from the packet capture file.

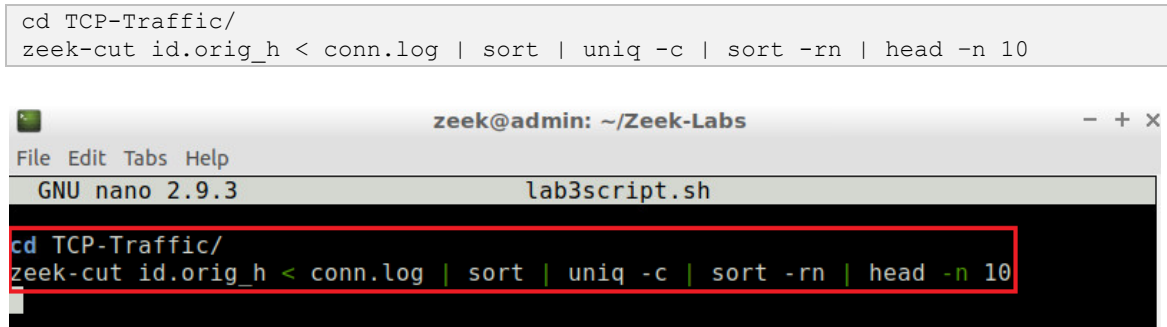
Step 1: Open the `lab3script.sh` file with `nano` text editor.

```
nano lab3script.sh
```



Step 2: Modify the script file's contents. Delete all the previous content and type the following command:

```
cd TCP-Traffic/
zeek-cut id.orig_h < conn.log | sort | uniq -c | sort -rn | head -n 10
```



Press `CTRL + o` and hit `Enter` to save the file's contents, then `CTRL + x` to exit `nano` and return to the terminal. The above command is explained as follows:

- `zeek-cut id.orig_h < conn.log`: selects the `id.orig_h` column from the `conn.log` file.
- `| sort`: uses the `sort` command to organize the rows in alphabetical order.
- `| uniq -c`: uses the `uniq` command with the `-c` option to remove duplicates while returning unique instances and their counts.
- `| sort -rn`: uses the `sort` command with the `-rn` option to organize the rows in reverse numerical order.
- `| head -n 10`: uses the `head` command with the `-n` option to display the 10 topmost values.

Step 3: Execute the modified shell script.

```
./lab3script.sh
```

```

zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ ./lab3script.sh
397 192.168.3.131
201 172.16.255.1
92 10.0.2.15
2 10.0.2.2
1 66.209.190.254
1 65.55.57.251
1 65.55.25.60
1 174.36.30.111
zeek@admin:~/Zeek-Labs$

```

The number of duplicates is seen in the left column, while the matching source IP address is seen in the right column. Only 8 unique source addresses were found, and each was returned. From this output, we can conclude that the majority of network traffic was generated by the top 3 source IP addresses.

2.2 Example 2

Example 2: Show the 10 destination ports that received the most network traffic, organized in descending order.

To solve this example, we will be looking at the `id.resp_p` column because it contains the destination ports from the packet capture file.

Step 1: Open the `lab3script.sh` file with `nano` text editor.

```
nano lab3script.sh
```

```

zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ nano lab3script.sh
zeek@admin:~/Zeek-Labs$

```

Step 2: Modify the script file's contents. Delete all the previous content and type the following command:

```
cd TCP-Traffic/
zeek-cut id.resp_p < conn.log | sort | uniq -c | sort -rn | head -n 10
```

```

zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
GNU nano 2.9.3 lab3script.sh
cd TCP-Traffic/
zeek-cut id.resp_p < conn.log | sort | uniq -c | sort -rn | head -n 10

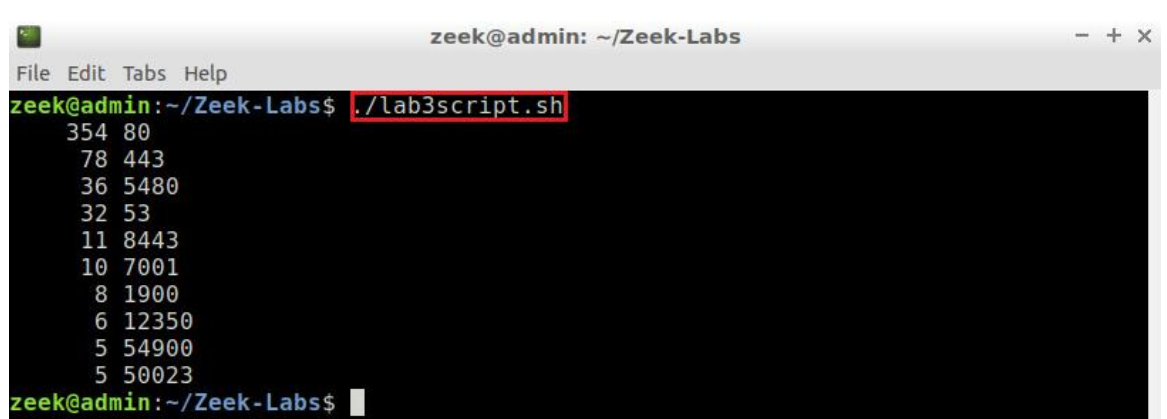
```

Press `CTRL + o` and hit `Enter` to save the file's contents, then `CTRL + x` to exit `nano` and return to the terminal. The above command is explained as follows:

- `zeek-cut id.resp p < conn.log`: selects the `id.resp p` column from the `conn.log` file.
- `| sort`: uses the `sort` command to organize the rows in alphabetical order.
- `| uniq -c`: uses the `uniq` command with the `-c` option to remove duplicates while returning unique instances and their counts.
- `| sort -rn`: uses the `sort` command with the `-rn` option to organize the rows in reverse numerical order.
- `| head -n 10`: uses the `head` command with the `-n` option to display the 10 topmost values.

Step 3: Execute the modified shell script.

```
./lab3script.sh
```



```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ ./lab3script.sh
354 80
78 443
36 5480
32 53
11 8443
10 7001
8 1900
6 12350
5 54900
5 50023
zeek@admin:~/Zeek-Labs$
```

The number of duplicates is seen in the left column, while the matching destination port is seen in the right column. More than 10 unique destination ports were found, so only the top 10 were returned. From this output we can conclude that port 80 received the most traffic.

2.3 Example 3

Example 3: Show the number of connections per protocol service.

To solve this example, we will be looking at the `service` column because it contains the destination ports from the packet capture file.

Step 1: Open the `lab3script.sh` file with `nano` text editor.

```
nano lab3script.sh
```

```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ nano lab3script.sh
zeek@admin:~/Zeek-Labs$
```

Step 2: Modify the script file's contents. Delete all the previous content and type the following command:

```
cd TCP-Traffic/
zeek-cut service < conn.log | sort | uniq -c | sort -n
```

```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
GNU nano 2.9.3 lab3script.sh
cd TCP-Traffic/
zeek-cut service < conn.log | sort | uniq -c | sort -n
```

Press `CTRL + o` and hit `Enter` to save the file's contents, then `CTRL + x` to exit `nano` and return to the terminal. The above command is explained as follows:

- `zeek-cut service < conn.log`: selects the `service` column from the `conn.log` file.
- `| sort`: uses the `sort` command to organize the rows in alphabetical order.
- `| uniq -c`: uses the `uniq` command with the `-c` option to remove duplicates while returning unique instances and their counts.
- `| sort -n`: uses the `sort` command with the `-n` option to organize the rows in numerical order.

Step 3: Execute the modified shell script.

```
./lab3script.sh
```

```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ ./lab3script.sh
 1 snmp
 2 smb
 3 dhcp
38 dns
57 ssl
264 http
331 -
zeek@admin:~/Zeek-Labs$
```

The number of duplicates is seen in the left column, while the matching destination port is seen in the right column. From this output we can see that 331 packets did not have a marked protocol. This can be caused by a number of anomalies and is an example of how you can use the `zeek-cut` utility to return anomalies that require further identification.

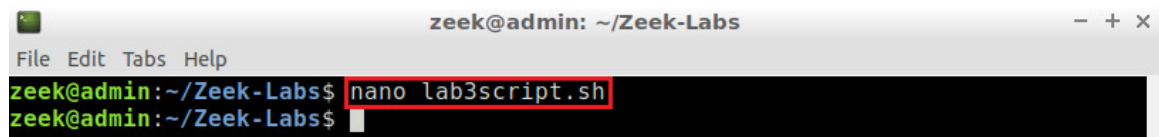
2.4 Example 4

Example 4: Print the distinct browsers used by the hosts in this packet capture file to a separate file.

To solve this example, we will be looking at the `user_agent` column because it contains the browser and connection-related information from the packet capture file.

Step 1: Open the `lab3script.sh` file with `nano` text editor.

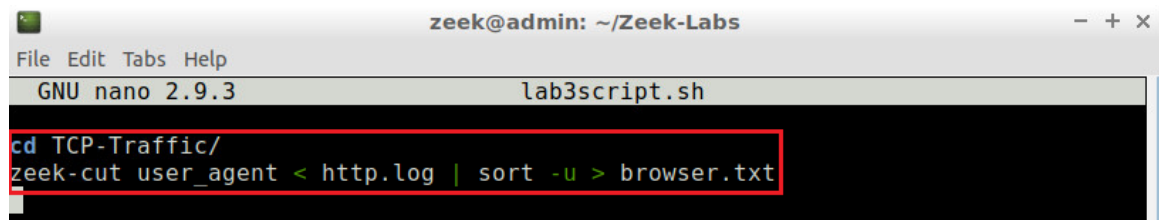
```
nano lab3script.sh
```



```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ nano lab3script.sh
zeek@admin:~/Zeek-Labs$
```

Step 2: Modify the script file's contents.

```
cd TCP-Traffic/
zeek-cut user_agent < http.log | sort -u > browser.txt
```



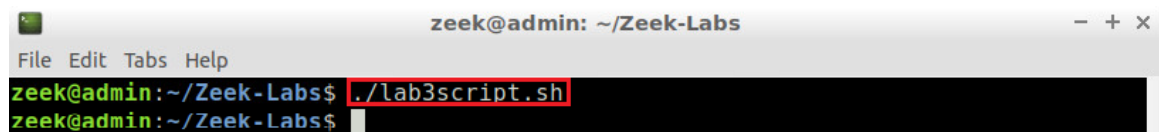
```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
GNU nano 2.9.3 lab3script.sh
cd TCP-Traffic/
zeek-cut user_agent < http.log | sort -u > browser.txt
```

Press `CTRL + o` and hit `Enter` to save the file's contents, then `CTRL + x` to exit `nano` and return to the terminal. The above command is explained as follows:

- `zeek-cut user_agent < http.log` selects the `user_agent` column from the `http.log` file.
- `| sort -u > browser.txt` uses the `sort` command to sort the lines in the file and the `-u` option checks for strict ordering. The output is then saved into the `browser.txt` file.

Step 3: Execute the modified shell script.

```
./lab3script.sh
```



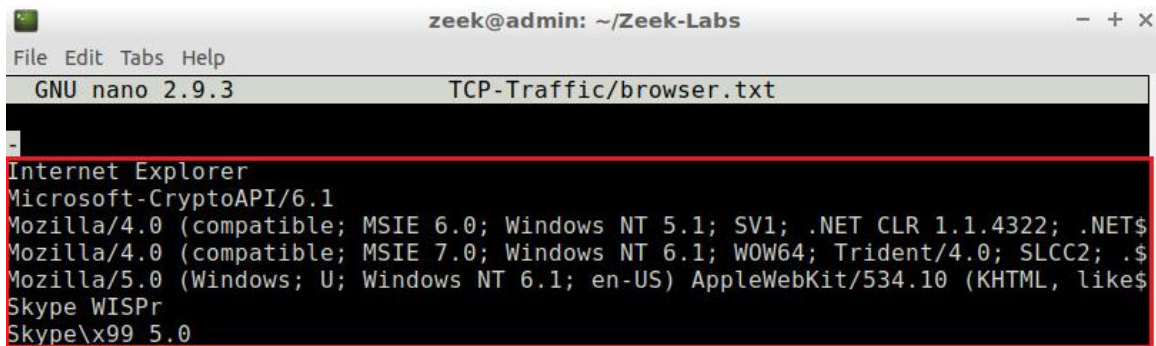
```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ ./lab3script.sh
zeek@admin:~/Zeek-Labs$
```

Step 4: Use a text editor to view the contents of the *browser.txt* file.

```
nano TCP-Traffic/browser.txt
```



Step 5: View the distinct browser information.



```
GNU nano 2.9.3 TCP-Traffic/browser.txt
-
Internet Explorer
Microsoft-CryptoAPI/6.1
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NETS
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .S
Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.10 (KHTML, like$
Skype WISPr
Skype\x99 5.0
```

Each browser found within the packet capture file is printed with related information extracted from the traffic by Zeek.

3 Incorporating the AWK scripting language for log file analysis

AWK is a terminal scripting language used to parse, filter and modify text files. AWK is specifically useful when processing rows and columns found in a Comma Separated Value (CSV) file. Additionally, AWK's integrated string manipulation functions allow for the searching and modifying of specific output.

Like `cat` and `head` commands, AWK output can be piped into the `zeek-cut` utility, allowing more advanced parsing and formatting options. AWK reads each column in a file through its position. The first input column is accessed using `$1` while the second column is accessed using `$2` and so on. AWK also allows creating simple variables to store and read script values. AWK reads the input data as a loop, starting from the top of the file and finishing at the end of the file. Each row is considered an instance within the script.

3.1 Example 1

Example 1: Find the source and destination IP address of all UDP and TCP connections that lasted more than one minute.

Step 1: Open the *lab3script.sh* file with `nano` text editor.

```
nano lab3script.sh
```

Step 2: Modify the script file’s contents. Delete all the previous content and type the following command:

```
cd TCP-Traffic/
awk '$9 > 60' conn.log | zeek-cut id.orig_h id.resp_h
```

Press `CTRL + o` and hit `Enter` to save the file’s contents, then `CTRL + x` to exit `nano` and return to the terminal. The above command is explained as follows:

- `awk '$9 > 60' conn.log` selects the rows that have their 9th column value greater than 60 from the `conn.log` file. The 9th field represents the connection duration, and we are checking if the value is greater than 60 seconds (or 1 minute).
- `| zeek-cut id.orig_h id.resp_h` returns the source and destination IP addresses.

Step 3: Execute the modified shell script.

```
./lab3script.sh
```

```
192.168.3.131 72.14.213.101
192.168.3.131 65.55.118.107
192.168.3.131 207.46.0.109
192.168.3.131 63.215.202.48
192.168.3.131 206.108.207.163
192.168.3.131 206.108.207.155
192.168.3.131 63.215.202.49
172.16.255.1 204.9.163.184
10.0.2.15 65.54.189.173
192.168.3.131 66.220.149.32
172.16.255.1 194.192.199.252
```

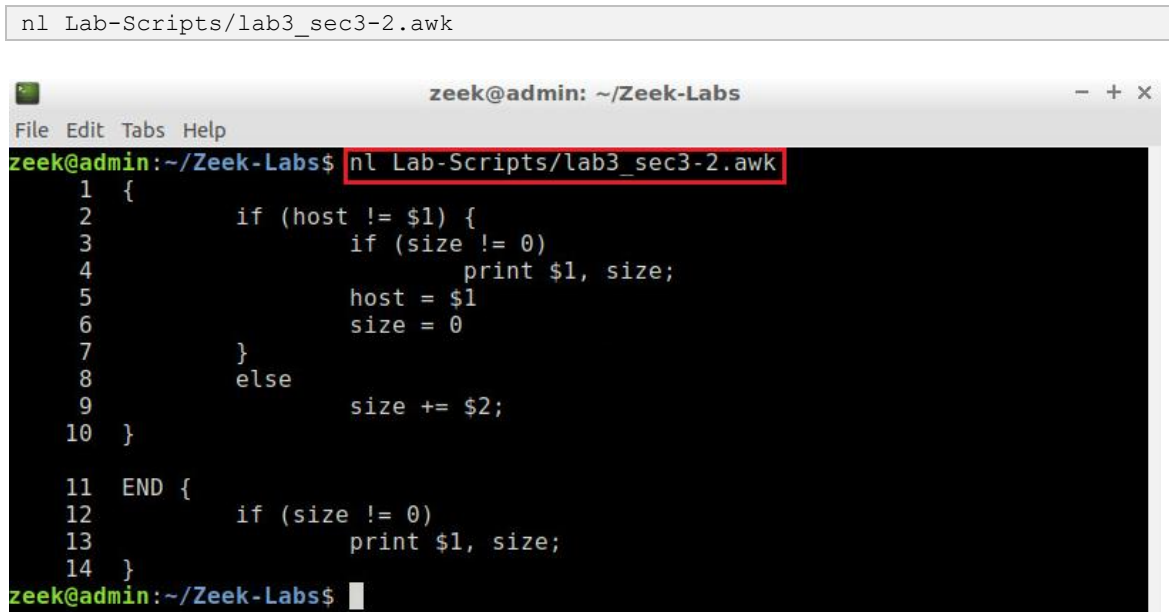

The source IP address is seen in the left column, while the matching destination IP address is seen in the right column. The pairs will only be displayed if the connection lasted at least one minute.

3.2 Example 2

Example 2: Show the top source host addresses in terms of total traffic (in bytes) sent in descending order.

The *Lab-Scripts* directory contains an AWK script named *lab3_sec3-2.awk* that can be viewed with the following command:

```
n1 Lab-Scripts/lab3_sec3-2.awk
```



```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ n1 Lab-Scripts/lab3_sec3-2.awk
1 {
2     if (host != $1) {
3         if (size != 0)
4             print $1, size;
5         host = $1
6         size = 0
7     }
8     else
9         size += $2;
10 }
11 END {
12     if (size != 0)
13         print $1, size;
14 }
zeek@admin:~/Zeek-Labs$
```

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

1. The `{` character is used to begin nested statements. This instance is the main functionality of the script.
2. The `host` variable, which will be used to store the source IP addresses found in the first column (`$1`), is checked against the current data entry in the column. If it is not equal, we will enter the next statement. Because we only want one instance of each source IP address, but the summed value of bytes sent, we will use this check to prevent duplicate entries.
3. This line contains a check to make sure the current packet is not empty and does contain a payload. If the current packet contains a payload of more than 0 bytes, we will proceed to line 4.
4. The current source IP address and its byte payload will be printed or returned to the next statements.

5. Now that we know the current source IP address is not yet stored in the host variable, we will create a new entry into the variable.
6. The size variable is reset back to zero
7. The `}` character is used to end nested statements. Therefore, the first case of a source IP address not being contained in host is complete.
8. If the host variable contains the current data entry, we will proceed to line 9.
9. Here we will sum the unique source IP address' total bytes by adding the payload from the second column (`$2`).
10. The `}` character is used to end nested statements. This is the ending of the main functionality of the script.
11. The `END` statement denotes what the script will do once it has reached the end of the file, and there are no more input data rows to be read.
12. If a source IP address contains a total payload of more than 0 bytes, we will proceed to line 13.
13. AWK will return the source IP address found in the first column, as well as the size variable, containing the total payload in relation to that source IP address.

Step 1: Input the following command.

```
zeek-cut id.orig_h orig_bytes < TCP-Traffic/conn.log | sort | awk -f Lab-Scripts/lab3_sec3-2.awk | sort -k 2 | head -n 10
```

```
zeek@admin: ~/Zeek-Labs
File Edit Tabs Help
zeek@admin:~/Zeek-Labs$ zeek-cut id.orig_h orig_bytes < TCP-Traffic/conn.log | s
ort | awk -f Lab-Scripts/lab3_sec3-2.awk | sort -k 2 | head -n 10
65.55.25.60 1062064
174.36.30.111 128479
172.16.255.1 1605
10.0.2.2 76751
zeek@admin:~/Zeek-Labs$
```

- `zeek-cut id.orig_h orig_bytes < conn.log`: selects the `id.orig_h` and `orig_bytes` columns from the `conn.log` file.
- `| sort`: uses the `sort` command to organize the rows in alphabetical order.
- `| awk -f lab3_sec3-2.awk`: will execute awk with the `-f` option to denote using the script find within the `lab3_sec3-2.awk` file.
- `| sort -k 2`: uses the `sort` command with the `-k` option to organize the rows based on the values found in the second column – the total number of bytes.
- `| head -n 10`: uses the `head` command with the `-n` option to display the 10 topmost values.

The left column contains the source IP address, while the right column contains the number of bytes produced by the paired source IP address.

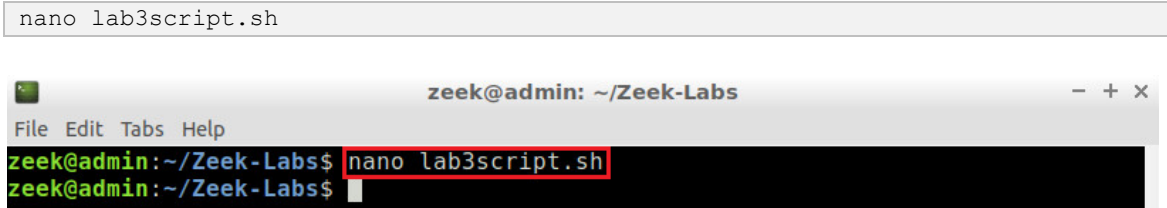
3.3 Example 3

Example 3: Are there any web servers operating on non-standardized ports?

To solve this example, we will be looking at the `service` column to view the packets using the Hyper Text Transport Protocol (HTTP) protocol. The standard ports for the HTTP protocol are 80 and 8080, so we will be searching for the network traffic that does not reach those ports.

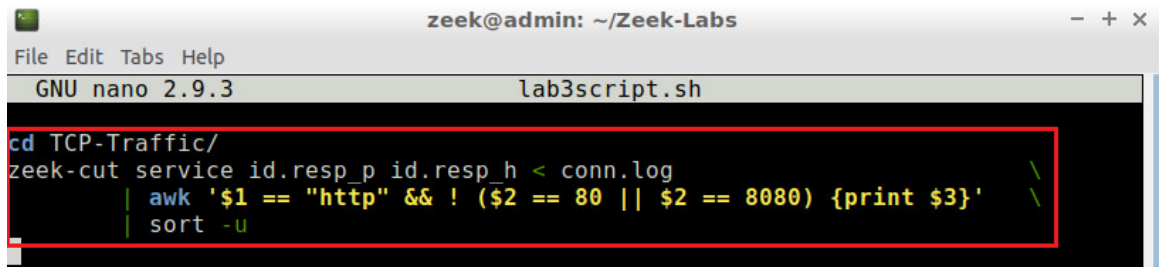
Step 1: Open the `lab3script.sh` file with `nano` text editor.

```
nano lab3script.sh
```



Step 2: Modify the script file's contents. Delete all the previous content and type the following command:

```
cd TCP-Traffic/
zeek-cut service id.resp_p id.resp_h < conn.log \
| awk '$1 == "http" && ! ($2 == 80 || $2 == 8080) {print $3}' \
| sort -u
```



```
GNU nano 2.9.3 lab3script.sh
cd TCP-Traffic/
zeek-cut service id.resp_p id.resp_h < conn.log \
| awk '$1 == "http" && ! ($2 == 80 || $2 == 8080) {print $3}' \
| sort -u
```

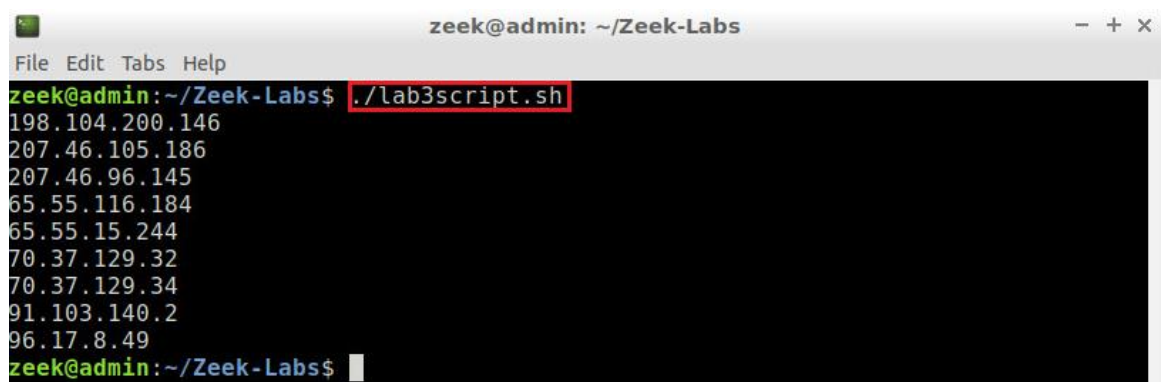
Press `CTRL + o` and hit `Enter` to save the file's contents, then `CTRL + x` to exit `nano` and return to the terminal. The above command is explained as follows:

- `zeek-cut service id.resp_p id.resp_h < conn.log`: selects the `service`, `id.resp_p` and `id.resp_h` columns from the `conn.log` file.
- `| awk`: passes the input into the following AWK command:
 - `$1 == "http"`: performs a check on the first column to make sure the active data entry is running on the http service.
 - `&& ! ($2 == 80 || $2 == 8080)`: performs a second check if the first check is successfully passed. The ports will be checked and if they are not equal to either of the standard http ports (80 and 8080), they will be passed to the print statement
 - `{print $3}`: prints the destination IP address of any host that passes both of the previous checks.

- `| sort -u`: uses the `sort` command to sort the lines in the file and the `-u` option checks for strict ordering.

Step 3: Execute the modified shell script.

```
./lab3script.sh
```



The screenshot shows a terminal window titled "zeek@admin: ~/Zeek-Labs". The command `./lab3script.sh` is entered and highlighted with a red box. The output displays a list of destination IP addresses: 198.104.200.146, 207.46.105.186, 207.46.96.145, 65.55.116.184, 65.55.15.244, 70.37.129.32, 70.37.129.34, 91.103.140.2, and 96.17.8.49. The prompt returns to `zeek@admin:~/Zeek-Labs$`.

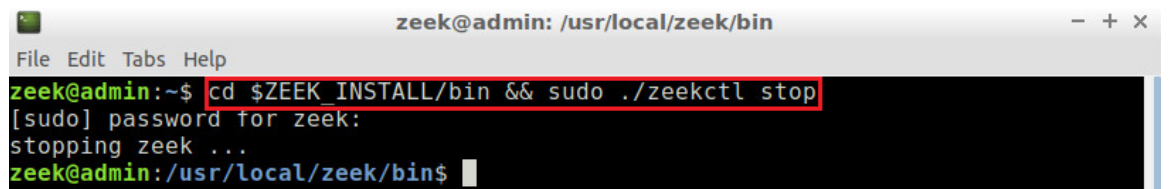
The destination IP addresses that received traffic on non-standardized ports are displayed.

3.4 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
```



The screenshot shows a terminal window titled "zeek@admin: /usr/local/zeek/bin". The command `cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop` is entered and highlighted with a red box. The output shows the prompt for the password: `[sudo] password for zeek:`, followed by `stopping zeek ...`. The prompt returns to `zeek@admin: /usr/local/zeek/bin$`.

Concluding this lab, we have reviewed the process of creating shell scripts to be used for network analysis. We introduced more complex commands for the `zeek-control` utility, as well as used the AWK scripting language to retrieve information from Zeek log files.

References

1. "Logging", Zeek user manual, [Online], Available: docs.zeek.org/en/stable/examples/logs.
2. "Exercise: understanding and examining bro logs", Zeek user manual, [Online], Available: <https://www.zeek.org/bro-workshop-2011/solutions/logs/index.html>.



The University of Texas at San Antonio™
The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 4: Generating, Capturing and Analyzing Network Scanner Traffic

Document Version: **03-13-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	4
Objective	4
Lab topology.....	4
Lab settings	4
Lab roadmap	5
1 Introduction to Internet scanning and probing.....	5
2 Generating real time network scans.....	5
2.1 Starting a new instance of Zeek	6
2.2 Launching Mininet.....	6
2.3 Setting up the zeek2 virtual machine for live network capture	8
2.4 Using the zeek1 virtual machine for network scanning activities	9
2.4.1 nmap options	10
2.4.2 TCP SYN scans	10
2.4.3 TCP connect scans.....	11
2.4.4 TCP NULL scans	11
2.4.5 TCP XMAS scans	12
2.4.6 Terminating live network capture	13
3 Analyzing collected network traffic	13
3.1 Example Query 1	15
3.2 Example Query 2	15
3.3 Closing the current instance of Zeek.....	16
References	17

Overview

This lab is designed to provide an in-depth guide to scanning and probing network traffic. The lab demonstrates the generation of scan-based traffic and uses Zeek to process the collected traffic.

Objective

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

1. Perform Internet scanning and probing events.
2. Utilize the Nmap software.
3. Generate and collect scan traffic.

Lab topology

Figure 1 shows the lab workspace topology. This lab primarily uses the *zeek1* virtual machine to generate scan-based traffic, and the *zeek2* virtual machine to perform live network capture.

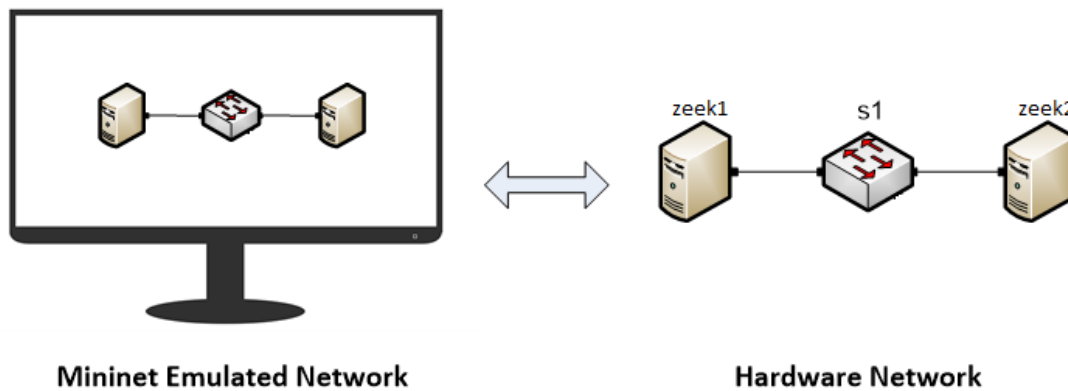


Figure 1. Lab topology.

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: Introduction to Internet scanning and probing.
2. Section 2: Generating real time network scans.
3. Section 3: Analyzing collected network traffic.
4. Section 4: Detailing the importance of the Zeek interface topology.

1 Introduction to Internet scanning and probing

Internet scanning is the process of generating crafted traffic used to identify active devices on a network. A variety of software utilities and tools are used to replicate scan-related traffic for testing purposes. These crafted packets can be both stealthy and versatile. It is hard to determine scan-like activities when scanning traffic follows protocols' standards and specifications.

Malicious scanning is a reconnaissance technique used to collect information about a target's machine or network to facilitate an attack against it. Scanning is used by attackers to discover what ports are open, what services are running and identify system software, all to enable an attacker to more easily detect and exploit known vulnerabilities within a target machine¹.

This lab uses [nmap](https://www.nmap.org/), and its documentation can be found on the [nmap](https://www.nmap.org/) website. 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://www.nmap.org/
```

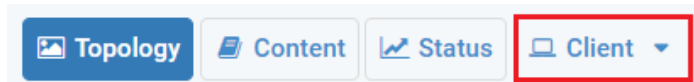
[nmap](https://www.nmap.org/) has a wide array of scan-related functionalities such as the customization of a scan's transport protocol, ports, IP ranges, etc.

2 Generating real time network scans

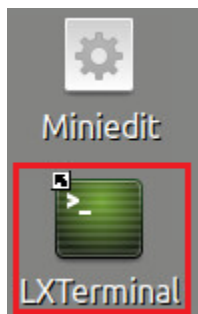
Zeek’s default packet capture processing generates log files containing organized network traffic statistics. By leveraging the *zeek1* virtual machine to scan the *zeek2* virtual machine, we can better define and understand the steps it takes to both generate and capture scan traffic.

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`.

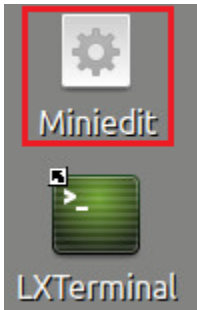
```
cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
```

```
zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
[sudo] password for zeek:
starting zeek ...
zeek@admin: /usr/local/zeek/bin$
```

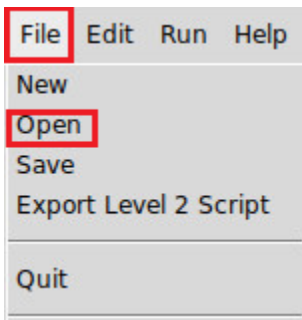
A new instance of Zeek is now active, and we are ready to proceed to the next section of the lab.

2.2 Launching Mininet

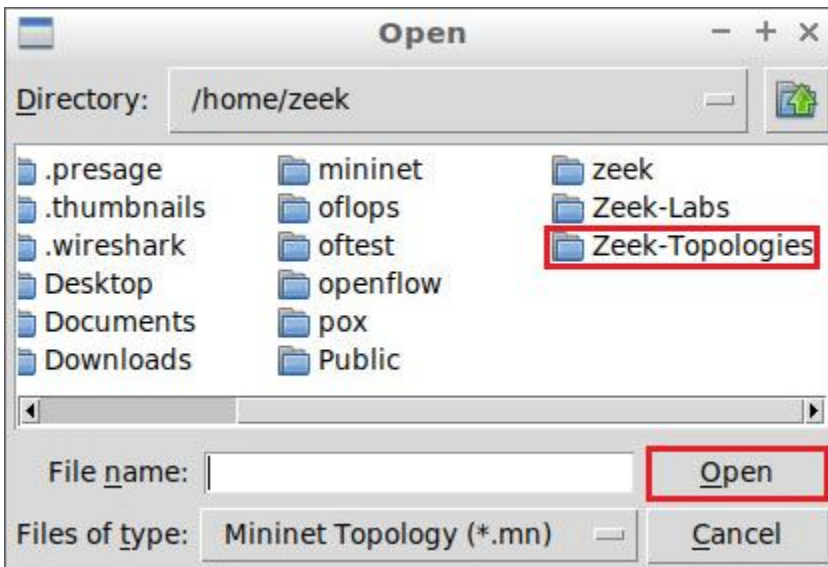
Step 1. From the *Client* machine's desktop, on the left side of the screen, click on the MiniEdit icon as shown below. When prompted for a password, type `password` and hit `Enter`. The MiniEdit editor will now launch.



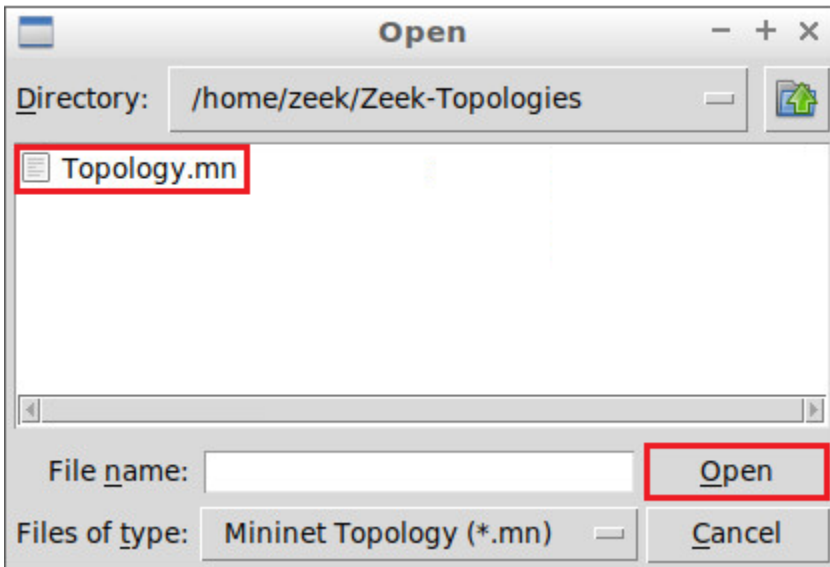
Step 2. The MiniEdit editor will now launch and allow for the creation of new, virtualized lab topologies. Load the correct topology by clicking the `Open` button within the `File` tab on the top left of the MiniEdit editor.



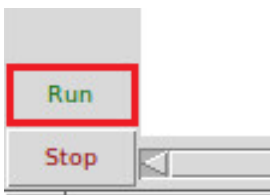
Step 3. Navigate to the Zeek-Topologies directory by scrolling to the right of the active directories and double clicking the Zeek-Topologies icon, or by clicking the `Open` button.



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

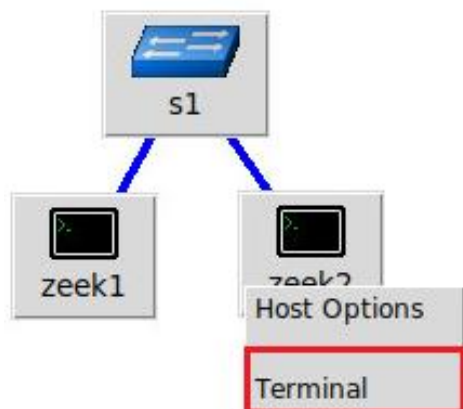


Step 5. To begin running the virtual machines, navigate to the **Run** button, found on the bottom left of the Miniedit editor, and select the **Run** button, as seen in the image below.



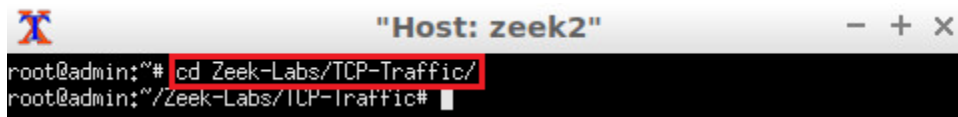
2.3 Setting up the zeek2 virtual machine for live network capture

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



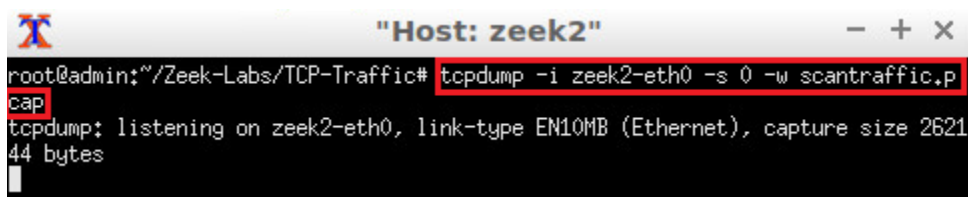
Step 2. From the *zeek2* terminal, navigate to the TCP-Traffic directory.

```
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*.

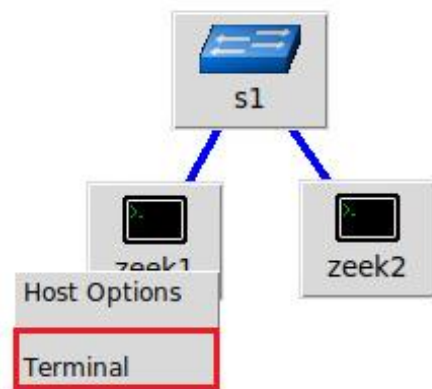
```
tcpdump -i zeek2-eth0 -s 0 -w scantraffic.pcap
```



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.

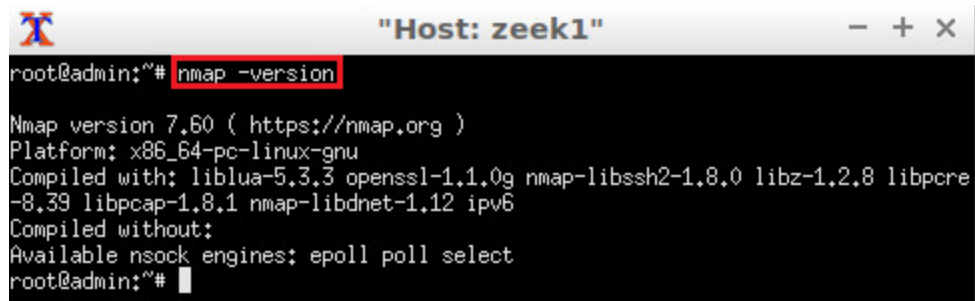
2.4 Using the *zeek1* virtual machine for network scanning activities

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. On a machine running Linux, `nmap` is executed through the Terminal. Verify that `nmap` is functioning properly by viewing the currently installed version.

```
nmap -version
```



```

Host: zeek1
root@admin:~# nmap -version
Nmap version 7.60 ( https://nmap.org )
Platform: x86_64-pc-linux-gnu
Compiled with: liblua-5.3.3 openssl-1.1.0g nmap-libssh2-1.8.0 libz-1.2.8 libpcr
-8.39 libpcap-1.8.1 nmap-libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
root@admin:~#

```

The figure above shows that the currently installed version of nmap is 7.60. With both the *zeek2* and *zeek1* virtual machines configured correctly, we can proceed with the exercises.

2.4.1 nmap options

`nmap` is used to discover hosts and services on a computer network by sending packets and analyzing the responses. `nmap` command has a list of options for every scan type and covers several protocols. This lab focuses on TCP scans with their default settings. Two additional options that can be used during this lab are:

- `-A`: enables operating system and version detection.
- `-T4`: faster execution, can strain the initiator's machine on larger scans.

More information is available on the following `nmap` documentation page:

```
https://nmap.org/book/man-briefoptions.html
```

2.4.2 TCP SYN scans

TCP SYN scans are one of the most common types of scans used for vulnerability detection. During SYN scanning, the initiating host sends a single TCP SYN packet to the destination. The receiving host interprets the request as a new TCP connection where the standard three-way TCP handshake is to be established. If a SYN/ACK packet is sent back, the initiator can infer that the port is open. The initiator can then send an RST (reset) packet to terminate the established connection.

Step 1. Use the following command to conduct a TCP SYN scan.

```
nmap -sS 10.0.0.2
```

The `-sS` option is used to indicate a TCP SYN scan.

```

"Host: zeek1"
root@admin:~# nmap -sS 10.0.0.2
Starting Nmap 7.60 ( https://nmap.org ) at 2020-01-10 13:46 EST
Nmap scan report for 10.0.0.2
Host is up (0.000019s latency).
All 1000 scanned ports on 10.0.0.2 are closed
MAC Address: 8E:A5:CB:04:7F:90 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 14.61 seconds
root@admin:~#

```

After the scan is completed, `nmap` produces a report on the performed scan. This includes the scan starting time, the number of ports, the total time, etc. We can see here the TCP SYN scan took 14.61 seconds, and none of the scanned ports were open.

2.4.3 TCP connect scans

TCP Connect scans are an alternative to TCP SYN scans. Rather than starting a TCP handshake, the initiator's operating system attempts to establish a connection with the target victim through a system call. If a connection is successfully created, the initiator can infer that the receiver is open.

Step 1. Use the following command to conduct a TCP connect scan.

```
nmap -sT 10.0.0.2
```

The `-sT` option is used to indicate a TCP Connect scan.

```

"Host: zeek1"
root@admin:~# nmap -sT 10.0.0.2
Starting Nmap 7.60 ( https://nmap.org ) at 2020-01-10 13:47 EST
Nmap scan report for 10.0.0.2
Host is up (0.000075s latency).
All 1000 scanned ports on 10.0.0.2 are closed
MAC Address: 8E:A5:CB:04:7F:90 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 13.34 seconds
root@admin:~#

```

The report in the above figure shows that the scan was completed in 13.34 seconds, and none of the scanned ports were open.

2.4.4 TCP NULL scans

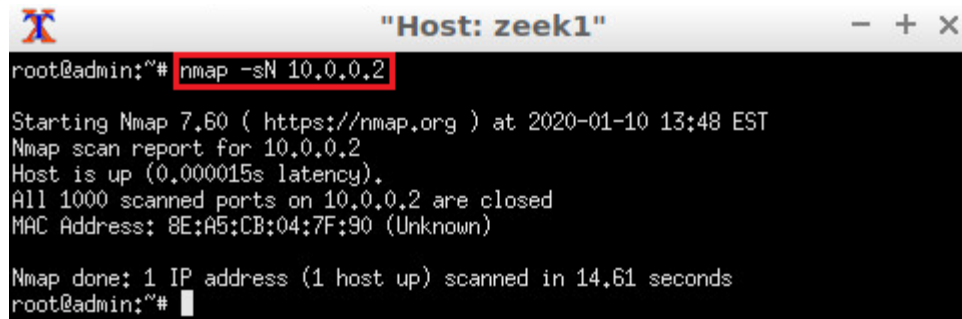
TCP NULL scans are another form of TCP scanning. In general, all TCP packets contain flags. Firewalls are configured to drop packets containing certain flags. The TCP NULL scan attempts to bypass these firewalls by excluding the header. With a sequence number of

0, packets in a TCP NULL scan will have no flags and can potentially infiltrate a network's firewall.

Step 1. Use the following command to conduct a TCP NULL scan.

```
nmap -sN 10.0.0.2
```

The `-sN` option is used to indicate a TCP NULL scan.



```
root@admin:~# nmap -sN 10.0.0.2
Starting Nmap 7.60 ( https://nmap.org ) at 2020-01-10 13:48 EST
Nmap scan report for 10.0.0.2
Host is up (0.000015s latency).
All 1000 scanned ports on 10.0.0.2 are closed
MAC Address: 8E:A5:CB:04:7F:90 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 14.61 seconds
root@admin:~#
```

The report in the above figure shows that the scan was completed in 14.61 seconds, and none of the scanned ports were open.

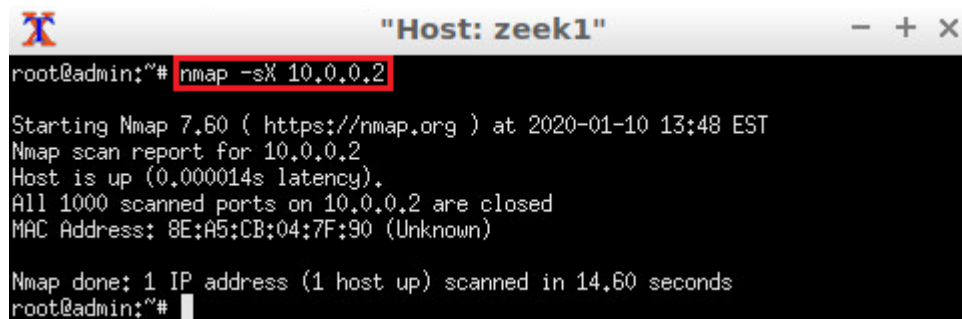
2.4.5 TCP XMAS scans

TCP Xmas scans, also known as Christmas tree scans, have their name derived from their set flags. In TCP Xmas scans, the PSH, URG and FIN flags are all set in the TCP header. This combination of flags is used in an attempt to infiltrate a strict network's firewall.

Step 1. Use the following command to conduct a TCP XMAS scan.

```
nmap -sX 10.0.0.2
```

The `-sX` option is used to indicate a TCP XMAS scan.



```
root@admin:~# nmap -sX 10.0.0.2
Starting Nmap 7.60 ( https://nmap.org ) at 2020-01-10 13:48 EST
Nmap scan report for 10.0.0.2
Host is up (0.000014s latency).
All 1000 scanned ports on 10.0.0.2 are closed
MAC Address: 8E:A5:CB:04:7F:90 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 14.60 seconds
root@admin:~#
```

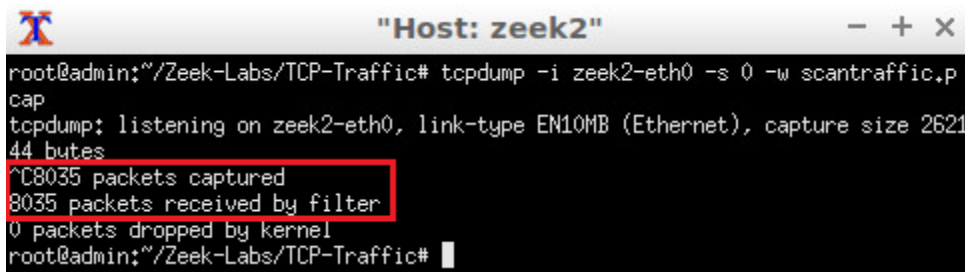
The report in the above figure shows that the scan was completed in 14.60 seconds, and none of the scanned ports were open or vulnerable.

2.4.6 Terminating live network capture

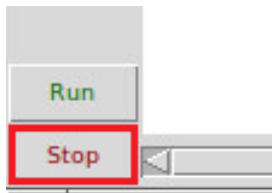
Step 1. Minimize the *zeek1* Terminal and open the *zeek2* Terminal using the navigation bar at the bottom of the screen. If necessary, right click within the Miniedit editor to activate your cursor.



Step 2. Use the `Ctrl+c` key combination to stop live traffic capture. Statistics of the capture session will be displayed. 8035 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 `x` on the top right of the editor.

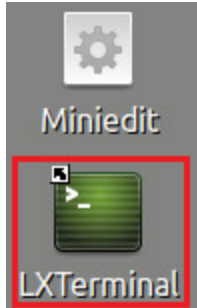


We will now return to the *Client* machine to process and analyze the newly generated network traffic.

3 Analyzing collected network traffic

After successfully conducting a number of TCP-based scans, the *scanpackets.pcap* packet capture file now contains the required network 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.



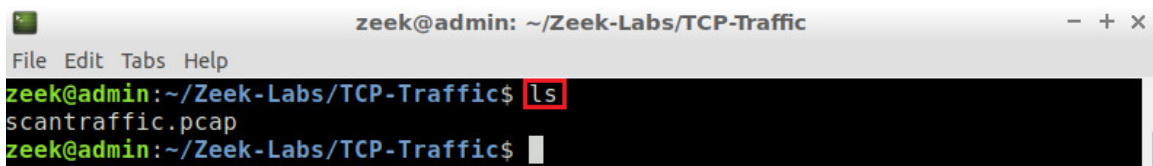
Step 2. Navigate to the *TCP-Traffic* directory to find the *scantraffic.pcap* file.

```
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. Use the following Zeek command to process the packet capture file.

```
zeek -C -r scantraffic.pcap
```



Similarly to the previous labs, Zeek will process the *scantraffic.pcap* file and generate resulting log files based off of the default Zeek configurations.

Step 5. List the generated Zeek log files.

```
ls
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
conn.log packet filter.log reporter.log scantraffic.pcap
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

With the log files generated, we can now use the `zeek-cut` utility for further analysis.

3.1 Example Query 1

Example 1: Show the source IP addresses that generated the most network traffic, organized in descending order.

Step 1. Enter the following command.

```
zeek-cut id.orig_h < conn.log | sort | uniq -c | sort -rn | head -n 10
```

The above command is explained as follows:

- `zeek-cut id.orig_h < conn.log`: selects the `id.orig_h` column from the `conn.log` file.
- `| sort`: uses the `sort` command to organize the rows in alphabetical order.
- `| uniq -c`: uses the `uniq` command with the `-c` option to remove duplicates while returning unique instances and their counts.
- `| sort -rn`: uses the `sort` command with the `-rn` option to organize the rows in reverse numerical order.
- `| head -n 10`: uses the `head` command with the `-n` option to display the 10 topmost values.

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek-cut id.orig_h < conn.log | sort | uni
q -c | sort -rn | head -n 10
4003 10.0.0.1
  3 fe80::2809:bff:fee3:47e2
  2 fe80::e89b:c7ff:fe34:d52c
  2 fe80::8ca5:cbff:fe04:7f90
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

We can see the majority of the packets were received from the `zeek1` machine denoted by the IP address 10.0.0.1.

3.2 Example Query 2

Example 2: Show the 10 destination ports that received the most network traffic, organized in descending order.

Step 1. Enter the following command.

```
zeek-cut id.resp_p < conn.log | sort | uniq -c | sort -rn | head -n 10
```

The above command is explained as follows:

- `zeek-cut id.orig_h < conn.log`: selects the `id.orig_h` column from the `conn.log` file.
- `| sort`: uses the `sort` command to organize the rows in alphabetical order.
- `| uniq -c`: uses the `uniq` command with the `-c` option to remove duplicates while returning unique instances and their counts.
- `| sort -rn`: uses the `sort` command with the `-rn` option to organize the rows in reverse numerical order.
- `| head -n 10`: uses the `head` command with the `-n` option to display the 10 topmost values.

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek-cut id.resp_p < conn.log | sort | uniq -c | sort -rn | head -n 10
 7 134
 5 8888
 5 7007
 5 113
 4 9999
 4 9998
 4 999
 4 9968
 4 995
 4 9944
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

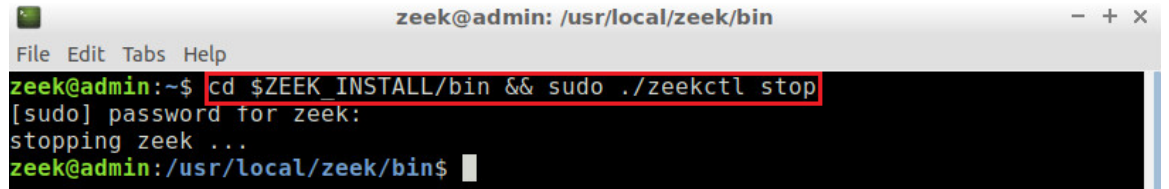
The number of duplicates is seen in the left column, while the matching destination port is seen in the right column. More than 10 unique destination ports were found, so only the top 10 were returned. These destination ports may be variable due to `nmap's` scanning configurations.

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
```



```
zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin: /usr/local/zeek/bin$
```

Concluding this lab, we have reviewed the steps required to generate scan traffic as well as enable live traffic capture using Zeek. Once collected, the trace files can be studied, and empirical data can be investigated regarding the current state of a network and its devices.

References

1. Bou-Harb, Elias, Mourad Debbabi, and Chadi Assi. "A systematic approach for detecting and clustering distributed cyber scanning." *Computer Networks* 57.18 (2013): 3826-3839.
2. Pour, Morteza Safaei, and Elias Bou-Harb. "Implications of theoretic derivations on empirical passive measurements for effective cyber threat intelligence generation." *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018.
3. "Options summary", nmap, [Online], Available: nmap, <https://nmap.org/book/man-briefoptions.html>.
4. "Port scanning techniques", nmap, [Online], Available: nmap, <https://nmap.org/book/man-port-scanning-techniques.html>.



The University of Texas at San Antonio™

The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 5: Generating, Capturing and Analyzing DoS and DDoS-centric Network Traffic

Document Version: **03-13-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objective	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to DoS and DDoS activity	4
1.1 DoS attack characteristics	4
1.2 DDoS attack characteristics.....	5
2 Generating real-time DoS traffic.....	5
2.1 Starting a new instance of Zeek	5
2.2 Launching Mininet.....	6
2.3 Setting up the zeek2 machine for live network capture.....	8
2.4 Launching LOIC.....	9
2.5 Using the zeek1 virtual machine to launch a TCP-based DoS attack.....	11
2.6 Using the zeek1 virtual machine to launch a UDP-based DoS attack.....	12
3 Analyzing collected network traffic	14
3.1 Analyzing TCP-based traffic.....	14
3.1.1 TCP Example Query 1.....	15
3.1.2 TCP Example Query 2.....	15
3.2 Analyzing UDP-based traffic.....	16
3.2.1 UDP Example Query 1.....	17
3.3 Closing the current instance of Zeek.....	18
References	19

Overview

This lab covers Denial of Service (DoS)-based network traffic. The lab introduces the generation of DoS-based traffic for testing purposes and uses Zeek to process the collected traffic.

Objective

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

1. Generate real-time DoS and DDoS traffic.
2. Experiment with the Low Orbit Ion Canon (LOIC) software.
3. Analyze collected DDoS traffic.

Lab topology

Figure 1 shows the lab workspace topology. This lab primarily uses the *zeek1* virtual machine to generate DoS-based traffic, and the *zeek2* virtual machine to perform live network capture.

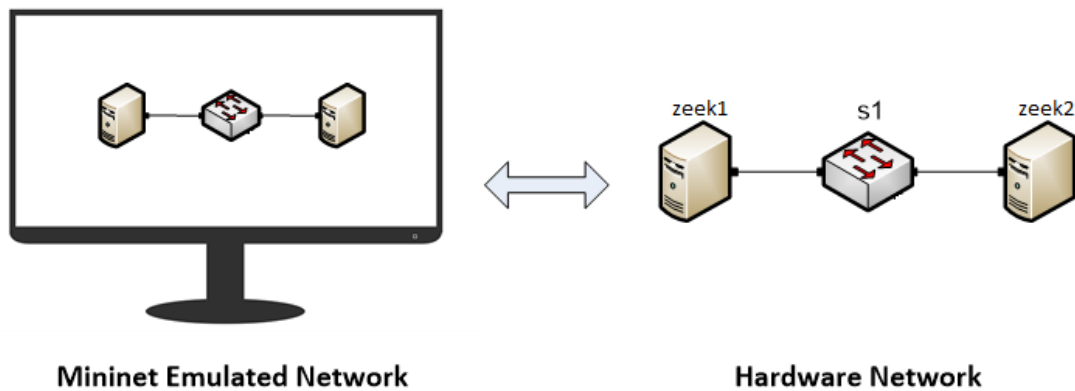


Figure 1. Lab topology.

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: Introduction to DoS and DDoS activity.
2. Section 2: Generating real-time DoS traffic.
3. Section 3: Analyzing collected network traffic.

1 Introduction to DoS and DDoS activity

Denial-of-Service (DoS) is an attack launched by a malicious user to render a target machine or network resource unavailable to its intended users. Distributed Denial-of-Service (DDoS) is an attack originated from different sources to flood the victim's resources. A DDoS attack is more effective than a normal DoS and is harder to mitigate since unlike DoS, it is impossible to stop the attack simply by blocking a single source.

The different types of DoS attacks can be grouped by the traffic they generate, the bandwidth they consume, the services they disrupt, etc. Traffic-based DoS attacks aim at flooding the target with a large volume unsolicited traffic. Bandwidth-based DoS attacks involve transmitting a massive amount of junk data to overload the victim and render its network equipment congested.

1.1 DoS attack characteristics

DoS attacks generally involve flooding a targeted victim with network traffic to cause a crash and make it unavailable to benign users. In this lab we explore two common DoS attacks:

- **SYN flood**: an attacker attempts to overwhelm the server machine by sending a constant stream of TCP connection requests, forcing the server to allocate resources for each new connection until all resources are exhausted¹.
- **ICMP flood**: the attacker abuses ICMP **ping** and floods the victim computer with Echo Request messages. When a computer receives an ICMP Echo Request message it responds with an ICMP Echo Reply message².

1.2 DDoS attack characteristics

DDoS attacks involve using a large number of devices to flood a victim. With an increased number of exploited machines, the amount of resources available to the attacker is far higher. Some relevant DDoS attacks are:

- **HTTP flood**: simple attack but requires a large number of resources. An attacker who controls several devices (botnet) can continually flood a server with HTTP requests until the server becomes unavailable and unable to respond to additional incoming requests.
- **SYN flood**: similar to the DoS SYN flood, a botnet initiates several sessions without completing a TCP handshake, causing the victim to consume its available resources.
- **Amplification attack**: attackers abuse UDP-based network protocols to launch DDoS attacks that exceed hundreds of Gbps in traffic volume. This is achieved via reflective DDoS attacks where an attacker does not directly send traffic to the victim but sends spoofed network packets to a large number of systems that reflect the traffic to the victim³. Domain Name System (DNS) and Network Time Protocol (NTP) are examples of application-layer protocols that act as potential amplification attack vectors.

DoS and DDoS attacks can cause catastrophic fallout and monetary losses to a victim.

2 Generating real-time DoS traffic

This lab uses the Low Orbit Ion Canon (LOIC), open-source network stress testing and DoS attack generator. LOIC can be found in the following Github repository. 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.

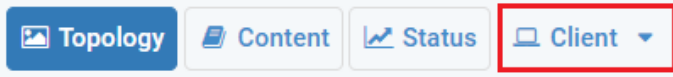
```
https://github.com/NewEraCracker/LOIC
```

Similar to the **nmap** utility, **LOIC** can be used to replicate DoS or DDoS activity for testing purposes. **LOIC** has a Graphical User Interface (GUI), which facilitates the attack's customization.

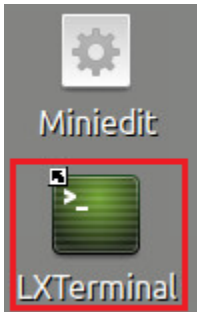
In this lab, Zeek's default packet capture processing will generate log files containing organized network traffic statistics. In this section, *zeek2* virtual machine is used for live capture and *zeek1* virtual machine is used to generate DoS-related traffic.

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`.

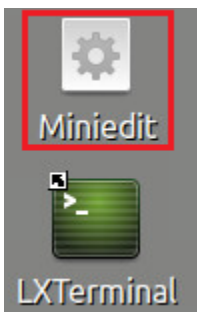
```
cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
```

```
zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
[sudo] password for zeek:
starting zeek ...
zeek@admin: /usr/local/zeek/bin$
```

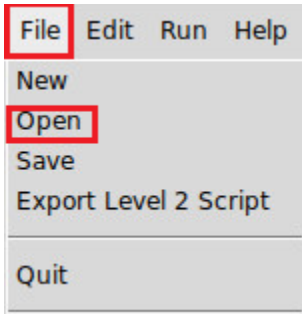
A new instance of Zeek is now active, and we are ready to proceed to the next section of the lab.

2.2 Launching Mininet

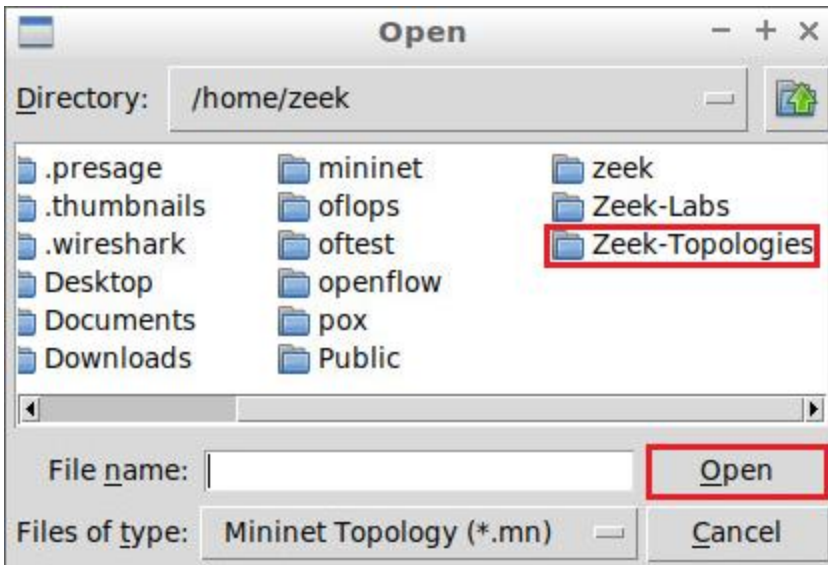
Step 1. From the *Client* machine’s desktop, on the left side of the screen, click on the MiniEdit icon as shown below. When prompted for a password, type `password` and hit `Enter`. The MiniEdit editor will now launch.



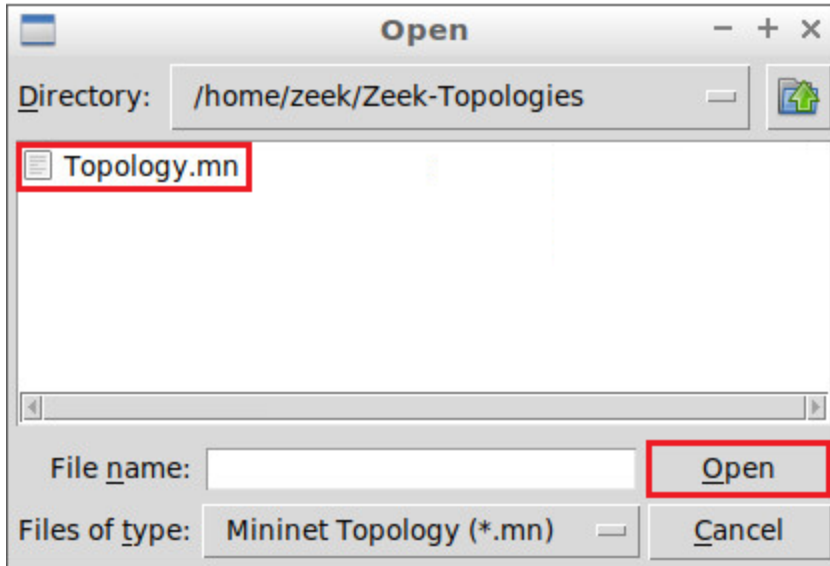
Step 2. The MiniEdit editor will now launch and allow for the creation of new, virtualized lab topologies. Load the correct topology by clicking the **Open** button within the **File** tab on the top left of the MiniEdit editor.



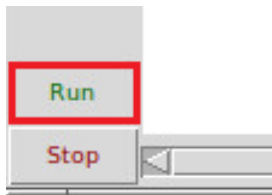
Step 3. Navigate to the Zeek-Topologies directory by scrolling to the right of the active directories and double clicking the Zeek-Topologies icon, or by clicking the **Open** button.



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

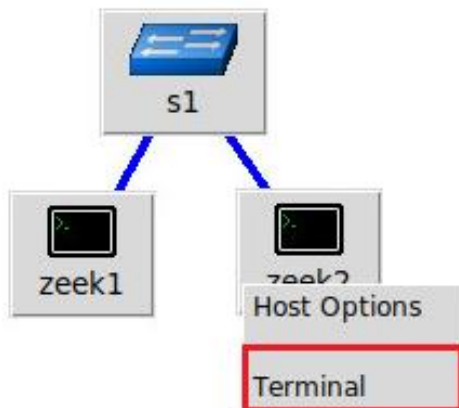


Step 5. To begin running the virtual machines, navigate to the `Run` button, found on the bottom left of the Miniedit editor, and select the `Run` button, as seen in the image below.



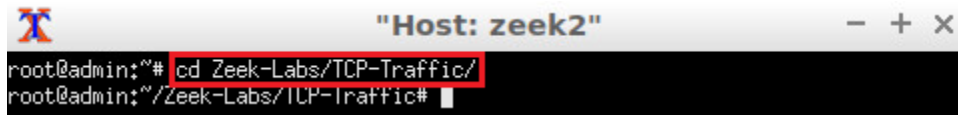
2.3 Setting up the zeek2 machine for live network capture

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



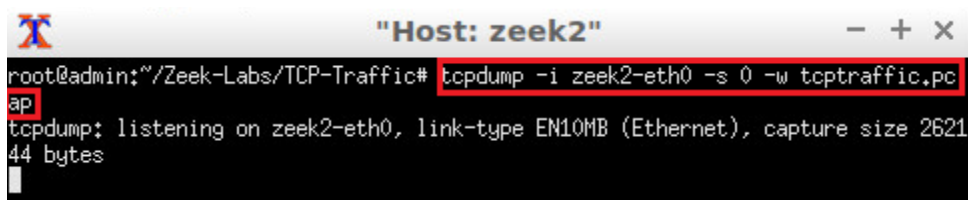
Step 2. From the *zeek2* terminal, navigate to the TCP-Traffic directory.

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



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

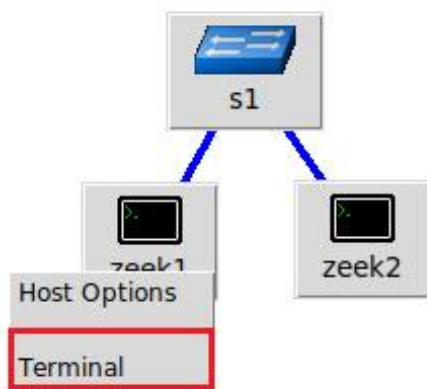
```
tcpdump -i zeek2-eth0 -s 0 -w tcptraffic.pcap
```



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 Launching LOIC

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. Navigate to the *Zeek-Labs/Lab-Tools/LOIC* directory.

```
cd Zeek-Labs/Lab-Tools/LOIC
```

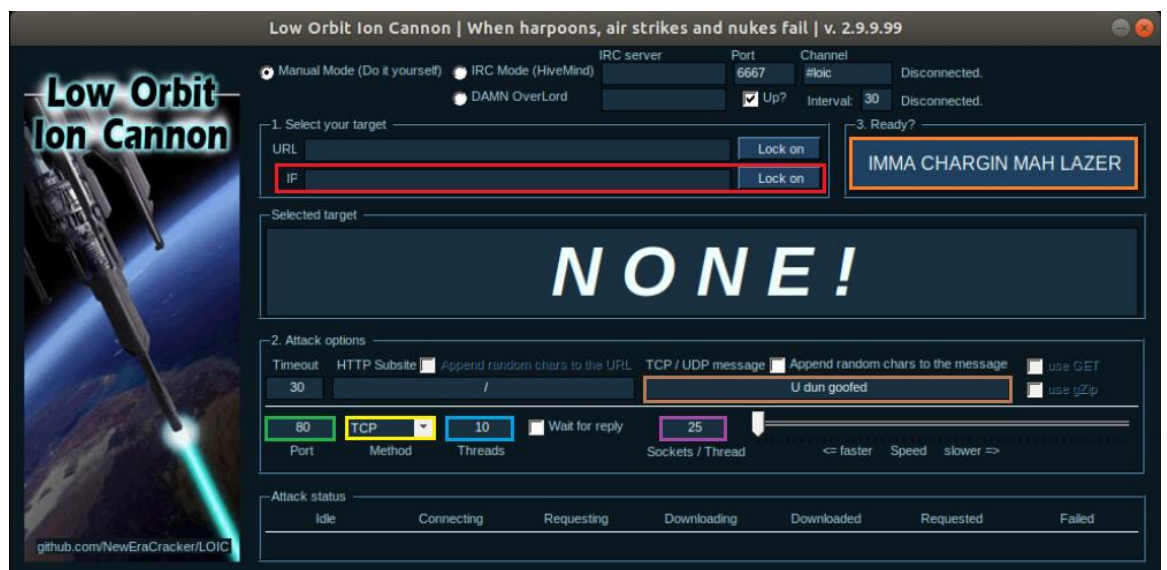


Step 3. Execute the *loic.sh* shell script by entering the following command in the terminal.

```
./loic.sh run
```



Step 4. View the LOIC GUI. If necessary, scale the GUI to a smaller size to fit on the *zeek1* virtual machine's display.



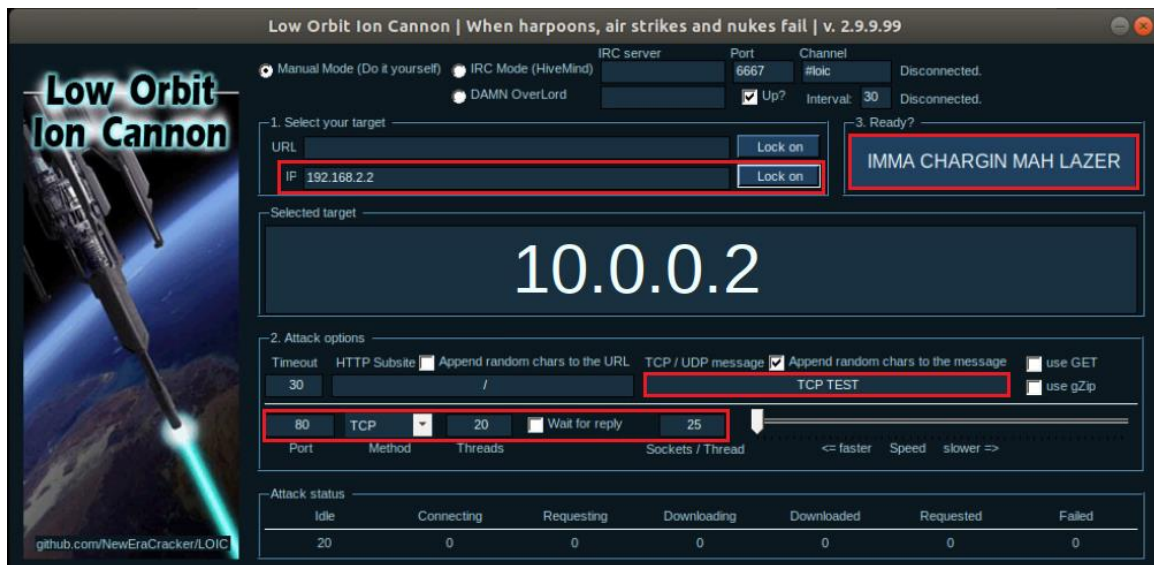
The figure above shows the LOIC interface. Important features highlighted with colored boxes are explained as follows:

1. **Red Box**: target IP address. After entering an IP address, clicking the *Lock on* button will select the IP as the target destination address.
2. **Green Box**: target port. Can be changed depending on which method is used to launch the DoS attack.
3. **Yellow Box**: target method. Can be changed to define which protocol is used to launch the DoS attack.
4. **Blue Box**: number of threads. Indicates the amount of resources *LOIC* will allocate on the host machine.
5. **Purple Box**: number of sockets per thread. Increasing the number of sockets per thread will exponentially increase the speed of the DoS attack; however, it also requires more resources on the host machine.
6. **Brown Box**: packet payload. Used to define what each packet will contain as payload.
7. **Orange Box**: start button. After customizing a desired attack, this button is used to launch the attack.

2.5 Using the zeek1 virtual machine to launch a TCP-based DoS attack

Step 1: Customize the DoS attack by entering the following values in their respective input boxes.

```
IP:          10.0.0.2
Port:       80
Method:     TCP
Threads:    20
Sockets:    25
Payload:    TCP TEST
```

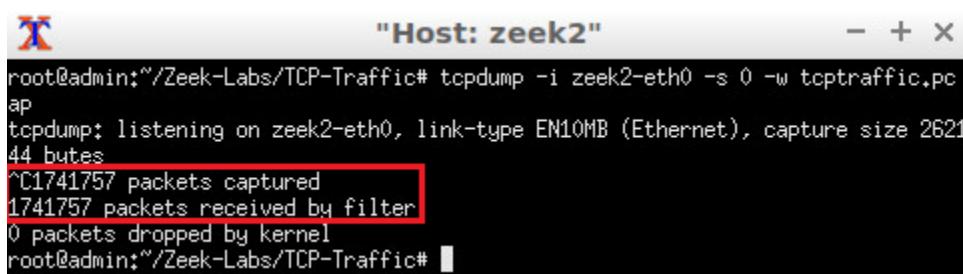


Step 2. Click the *Lock on* button to save the current configurations. Click the **Start** (*IMMA CHARGIN MAH LAZER*) button to begin the DoS attack. Wait roughly 10 seconds and click the **Stop** (*Stop flooding*) button to stop the DoS attack.

Step 3. 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 4. Use the **Ctrl+c** key combination to stop live traffic capture. Statistics of the capture session will be displayed with network packets being stored in the new *tcptraffic.pcap* file.

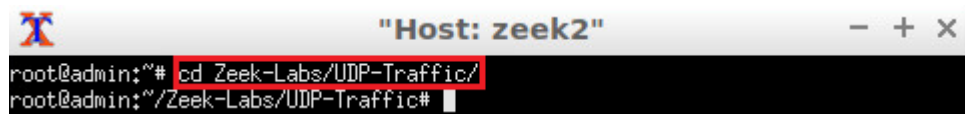


Within the 10 seconds timeframe, 1,741,757 packets were generated and collected. This number of packets verifies that DoS attacks generate an immense amount of network traffic and can be compared against the much smaller number of packets generated during the previous scan events.

2.6 Using the zeek1 virtual machine to launch a UDP-based DoS attack

Step 1. Using the *zeek2* virtual machine, navigate to the lab workspace directory and enter the *UDP-Traffic* directory.

```
cd Zeek-Labs/UDP-Traffic/
```



Step 2. Start live packet capture on interface *zeek2-eth0* and save the output to a file named *udptraffic.pcap*.

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

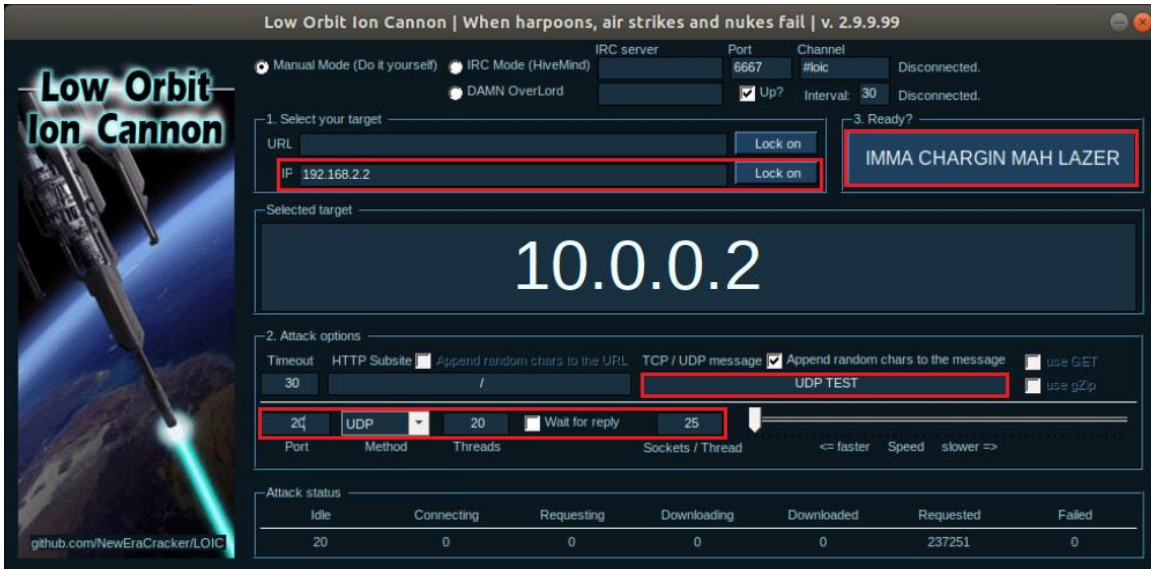


Step 3. Minimize the *zeek2* Terminal and open the *LOIC* GUI using the navigation bar at the bottom of the screen. If necessary, right click within the Miniedit editor to activate your cursor.



Step 4. Customize the DoS attack by entering the following values in their respective input boxes.

```
IP:          10.0.0.2
Port:        20
Method:      UDP
Threads:     20
Sockets:     25
Payload:     UDP TEST (Must be changed before updating Method feature)
```

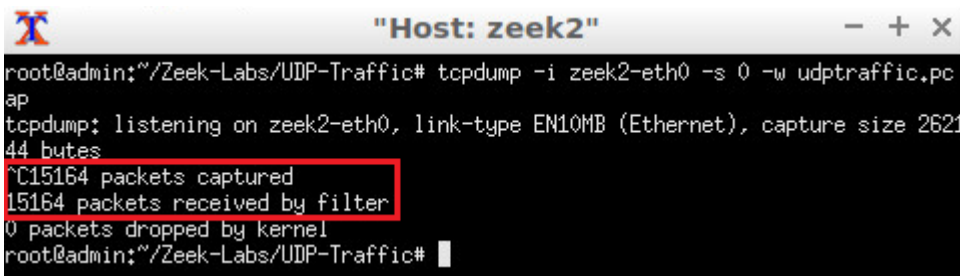


Step 4. Click the *Lock on* button to save the current configurations. Click the *Start (IMMA CHARGIN MAH LAZER)* button to begin the DoS attack. Wait for 10 seconds and click the *Stop (Stop flooding)* button to stop the DoS attack.

Step 5. 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.

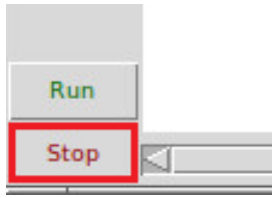


Step 6. Use the `Ctrl+c` key combination to stop live traffic capture. Statistics of the capture session will be displayed. 15,164 packets were recorded by the interface, which were then captured and stored in the new *tcptraffic.pcap* file.



While the UDP-based DoS attack did not generate as much network traffic as the TCP-based DoS attack, heavy amounts of traffic were generated by a single machine. Scaled to a large-scale attack, DoS attacks are extremely debilitating.

Step 7. 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 *x* on the top right of the editor.



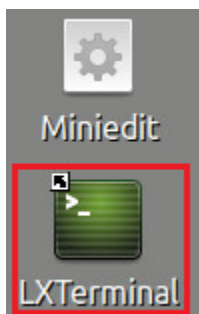
We will now return to the *Client* machine to process and analyze the newly generated network traffic.

3 Analyzing collected network traffic

After successfully conducting both a TCP-based and UDP-based DoS attack, we can begin to analyze the collected network traffic using Zeek and the `zeek-cut` utility commands to display the capture traffic.


3.1 Analyzing TCP-based traffic

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 *tcptraffic.pcap* file.

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

A screenshot of a terminal window. The window title is 'zeek@admin: ~/Zeek-Labs/TCP-Traffic'. The terminal shows the command 'cd Zeek-Labs/TCP-Traffic/' being entered and executed. The prompt changes from 'zeek@admin:~\$' to 'zeek@admin:~/Zeek-Labs/TCP-Traffic\$'. The command and the resulting prompt are highlighted with a red rectangular border.

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

```
ls
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
tcptraffic.pcap
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Step 4. Use the following Zeek command to process the packet capture file.

```
zeek -C -r tcptraffic.pcap
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek -C -r tcptraffic.pcap
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Step 5. List the generated Zeek log files.

```
ls
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
conn.log packet_filter.log tcptraffic.pcap
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

3.1.1 TCP Example Query 1

Example 1: Show the source IP addresses that generated the most network traffic, organized in descending order.

```
zeek-cut id.resp_p < conn.log | sort | uniq -c | sort -rn | head -n 10
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek-cut id.resp h < conn.log | sort | uni
q -c | sort -rn | head -n 10
870871 10.0.0.2
8 ff02::2
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

The *zeek2* virtual machine received 870,871 TCP packets. This command, or a similar one, can be useful in real-world environments to detect vulnerable hosts within a network – allowing for the process of securing and mitigating possible threats.

3.1.2 TCP Example Query 2

Example 1: Show the destination ports that received the most traffic, organized in descending order.

```
zeek-cut id.resp_p < conn.log | sort | uniq -c | sort -rn | head -n 10
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek-cut id.resp_p < conn.log | sort | uniq -c | sort -rn | head -n 10
870871 80
8 134
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

We can see that 870,871 packets were received by the *zeek2* virtual machine on port 80, which is the port we specified for the *zeek1* virtual machine to target. Additional ports may be discovered during processing, slightly variable due to LOIC attempting to establish connections; however, it is clear the most targeted port is the one we specified in the DoS attack.

3.2 Analyzing UDP-based traffic

Step 1. Navigate to the *UDP-Traffic* directory to find the *udptraffic.pcap* file.

```
cd Zeek-Labs/UDP-Traffic/
```

```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~$ cd Zeek-Labs/UDP-Traffic/
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

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

```
ls
```

```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ ls
udptraffic.pcap
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

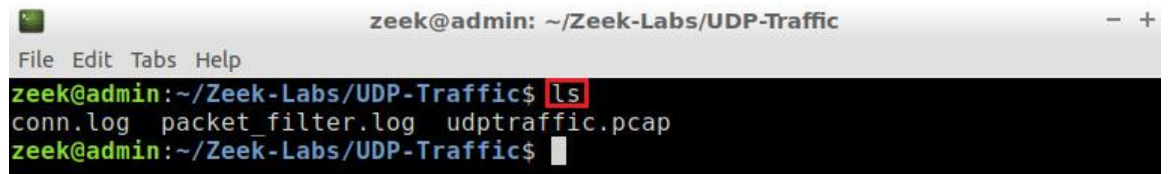
Step 4. Use the following Zeek command to process the packet capture file.

```
zeek -C -r udptraffic.pcap
```

```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ zeek -C -r udptraffic.pcap
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

Step 5. List the generated Zeek log files.

```
ls
```



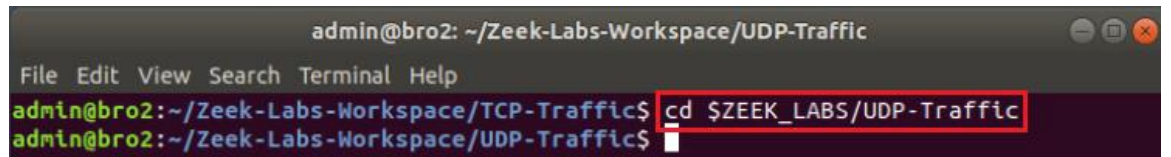
```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ ls
conn.log packet_filter.log udptraffic.pcap
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

3.2.1 UDP Example Query 1

Example 1: Show the list of ports that received any amount of network traffic.

Step 1. Navigate to the lab workspace directory and enter the *UDP-Traffic* directory.

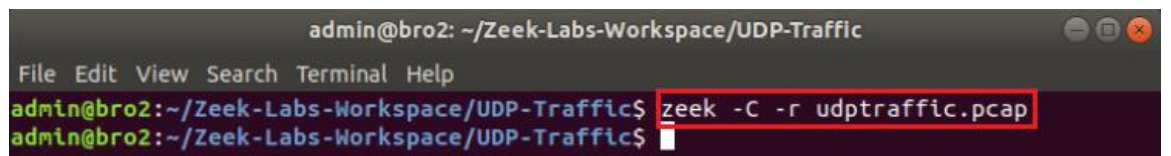
```
cd $ZEEK_LABS/UDP-Traffic
```



```
admin@bro2: ~/Zeek-Labs-Workspace/UDP-Traffic
File Edit View Search Terminal Help
admin@bro2:~/Zeek-Labs-Workspace/TCP-Traffic$ cd $ZEEK_LABS/UDP-Traffic
admin@bro2:~/Zeek-Labs-Workspace/UDP-Traffic$
```

Step 2. Process the *udptraffic.pcap* packet capture file using Zeek. The `-r` option indicates that Zeek will be reading from an offline pcap file, and the `-C` is used to disable checksum verification.

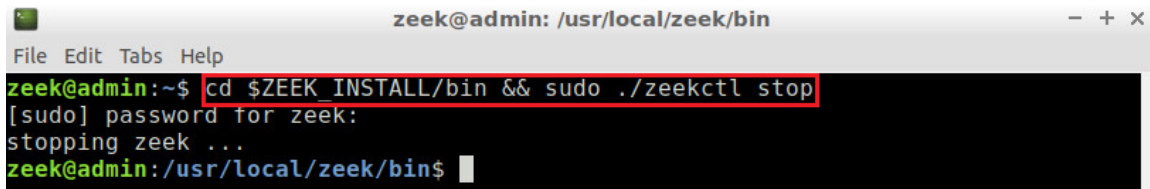
```
zeek -C -r udptraffic.pcap
```



```
admin@bro2: ~/Zeek-Labs-Workspace/UDP-Traffic
File Edit View Search Terminal Help
admin@bro2:~/Zeek-Labs-Workspace/UDP-Traffic$ zeek -C -r udptraffic.pcap
admin@bro2:~/Zeek-Labs-Workspace/UDP-Traffic$
```

Step 3. Show the list of ports that received network traffic.

```
cat conn.log | zeek-cut id.resp_p
```


A terminal window titled 'zeek@admin: /usr/local/zeek/bin' with a menu bar 'File Edit Tabs Help'. The terminal shows the command 'cd \$ZEEK INSTALL/bin && sudo ./zeekctl stop' being executed. The output is '[sudo] password for zeek:', 'stopping zeek ...', and the prompt 'zeek@admin: /usr/local/zeek/bin\$'.

```
zeek@admin:~$ cd $ZEEK INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin: /usr/local/zeek/bin$
```

Concluding this lab, we have introduced DoS and DDoS events, as well as generated and captured DoS traffic in the lab workspace environment. Networks require some form of denial-of-service mitigation or prevention tools since attacks can devastate unsecured networks.

References

1. Lemon, Jonathan, "Resisting SYN flood DoS attacks with a SYN cache," In BSDCon, vol. 2002, pp. 89-97. 2002.
2. Junior, R. B., & Kumar, S. (2014), "Apple's lion vs microsoft's windows 7: comparing built-in protection against ICMP flood attacks," Journal of information security, 5(03), 123.
3. Kührer, M., Hupperich, T., Rossow, C., & Holz, T. (2014). "Exit from hell? reducing the impact of amplification DDoS attacks,". In 23rd {USENIX} Security symposium ({USENIX} Security 14) (pp. 111-125).
4. Fachkha, Claude, Elias Bou-Harb, and Mourad Debbabi. "Fingerprinting internet DNS amplification DDoS activities." *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2014.
5. Fachkha, Claude, Elias Bou-Harb, and Mourad Debbabi. "Towards a forecasting model for distributed denial of service activities." *2013 IEEE 12th International Symposium on Network Computing and Applications*. IEEE, 2013.



The University of Texas at San Antonio™
The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 6: Introduction to Zeek Scripting

Document Version: **03-13-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to scripting with Zeek	4
1.1 Zeek script events.....	4
1.2 Zeek module workspace	5
1.3 Zeek log streams	5
2 Log file analysis using Zeek scripts.....	6
2.1 Starting a new instance of Zeek	6
2.2 Executing a UDP Zeek script.....	7
2.3 Executing a TCP Zeek script.....	8
3 Modifying Zeek log streams.....	10
3.1 Renaming the conn.log stream	10
3.2 Updating the conn.log stream	12
3.3 Closing the current instance of Zeek.....	13
References	14

Overview

This lab covers Zeek’s scripting language. It introduces the major keywords and components required in a Zeek script. The lab then uses these scripts to analyze processed log files.

Objectives

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

1. Develop scripts using Zeek’s scripting language.
2. Analyze processed log files using Zeek scripts.
3. Modify log streams for creating additional events and notices.

Lab topology

Figure 1 shows the lab workspace topology. This lab primarily uses the *Client* machine for offline Zeek script development and offline packet capture processing and analysis.

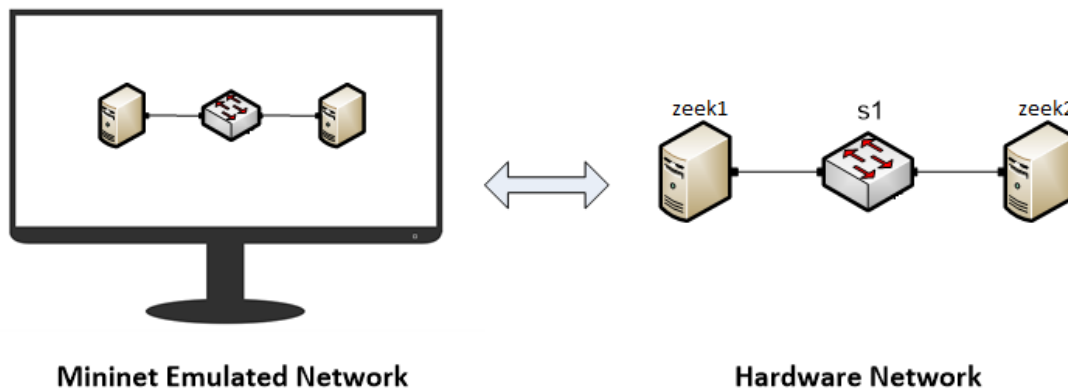


Figure 1. Lab topology.

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: Introduction to scripting with Zeek.
2. Section 2: Log file analysis using Zeek scripts.
3. Section 3: Modifying Zeek log streams.

1 Introduction to scripting with Zeek

Zeek includes its own event-driven scripting language which provides the primary means for an organization to extend and customize Zeek's functionality. By modifying Zeek's log streams, a more in-depth analysis can be performed on network events.

Since Zeek's scripting language is event-driven, we define which events we need Zeek to respond to when encountered during network traffic analysis.

1.1 Zeek script events

The script below shows events that will be explored during this lab. When developing a Zeek script, the script's functionalities are wrapped within respective events.

```

1 event zeek_init(){
2     /* code */
3 }
4 event zeek_done(){
5     /* code */
6 }
7 event tcp_packet(){
8     /* code */
9 }
10 event udp_request(){
11     /* code */
12 }
13 event udp_reply(){
14     /* code */
15 }

```

- `zeek_init` event: activated when Zeek is first initialized.
- `zeek_done` event: activated before Zeek is terminated.
- `tcp_packet` event: activated when a packet containing a TCP header is processed.

- `udp_request` event: activated when a packet containing a UDP request header is processed.
- `udp_reply` event: activated when a packet containing a UDP reply header is processed.

Additional events and their required parameters are outlined and explained in Zeek's official documentation. 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/current/examples/scripting/>

1.2 Zeek module workspace

The script below uses the `module` keyword which assigns the script to a *namespace*. Codes from other scripts can be accessed by including a matching module. The `export` keyword is used to export the code entered in its block with the module workspace.

```
1 module ZeekScript;
2
3 export {
4     /* Append a new Log stream */
5     /* Define a new data type to format new Log stream */
6 }
```

- `module ZeekScript`: changes the module workspace to ZeekScript.
- `export` block: code entered here will be exported with the module workspace.

Exporting code with a module workspace allows more advanced scripts to be built on top of other scripts.

1.3 Zeek log streams

The script below shows the log stream functionality. When developing a Zeek script, all processed outputs will be sent to a specific log stream. These log streams will contain the format of the corresponding log file output. We can create new streams, modify original streams or append additional parameters to existing streams.

```
1 event connection_established(){
2     Log::create_stream(LOG, format, path);
3     Log::write(Logstream, data);
4 }
```

- `connection_established` event: activated when a host makes a connection to a receiver.
- `Log::create_stream`: creates a new log stream, will a name, format structure and path.

- `Log::write`: writes included data to the specified log stream.

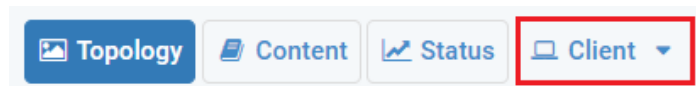
Additional log stream commands are explained in detail in Zeek's official documentation.

2 Log file analysis using Zeek scripts

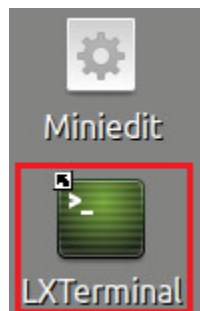
With Zeek's event-driven scripting language, we can create specific event-based filters to be applied during packet capture analysis. This section shows example scripts for network analysis.

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
```

```
zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
[sudo] password for zeek:
starting zeek ...
zeek@admin: /usr/local/zeek/bin$
```

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

2.2 Executing a UDP Zeek script

This lab series includes a *Lab-Scripts* directory, containing all of the relevant Zeek scripts that will be used during the labs.

Step 1. Navigate to the *Lab-Scripts* directory.

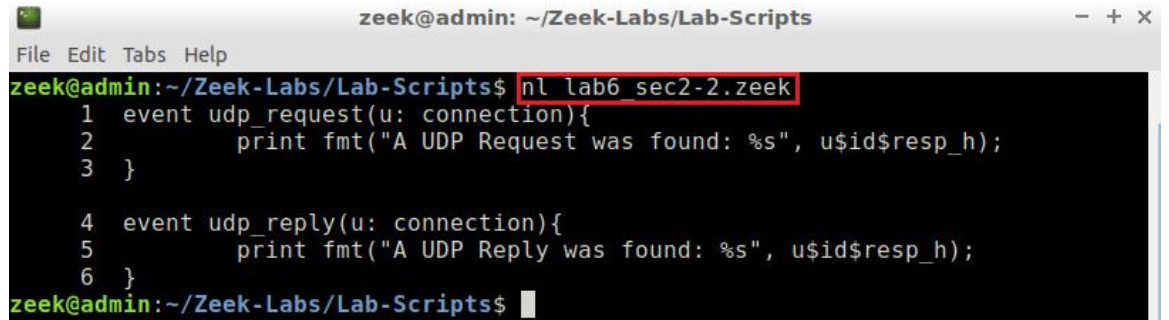
```
cd Zeek-Labs/Lab-Scripts/
```



Within this directory, all lab scripts can be accessed, viewed, and modified.

Step 2. Display the content of the *lab6_sec2-2.zeek* Zeek script using `nl` command. `nl` shows the line numbers in the file.

```
nl lab6_sec2-2.zeek
```



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

1. Event `udp_request` is activated when a packet containing a UDP Request header is processed. The related packet header information is stored in the connection data structure passed to the function through the `u` variable.
2. Prints the specified string. `%s` is a format specifier for strings with `fmt`. It indicates the position of the corresponding variable's information in the string. `uidresp_h` retrieves the destination IP address from the UDP packet.
3. End of the `udp_request` event.
4. Event `udp_reply` activated when a packet containing a UDP Reply header is processed. The related packet header information is stored in the connection data structure passed to the function through the `u` variable.

- Prints the specified string. `uipresp_h` retrieves the destination IP address from the UDP packet.
- End of the `udp_reply` event.

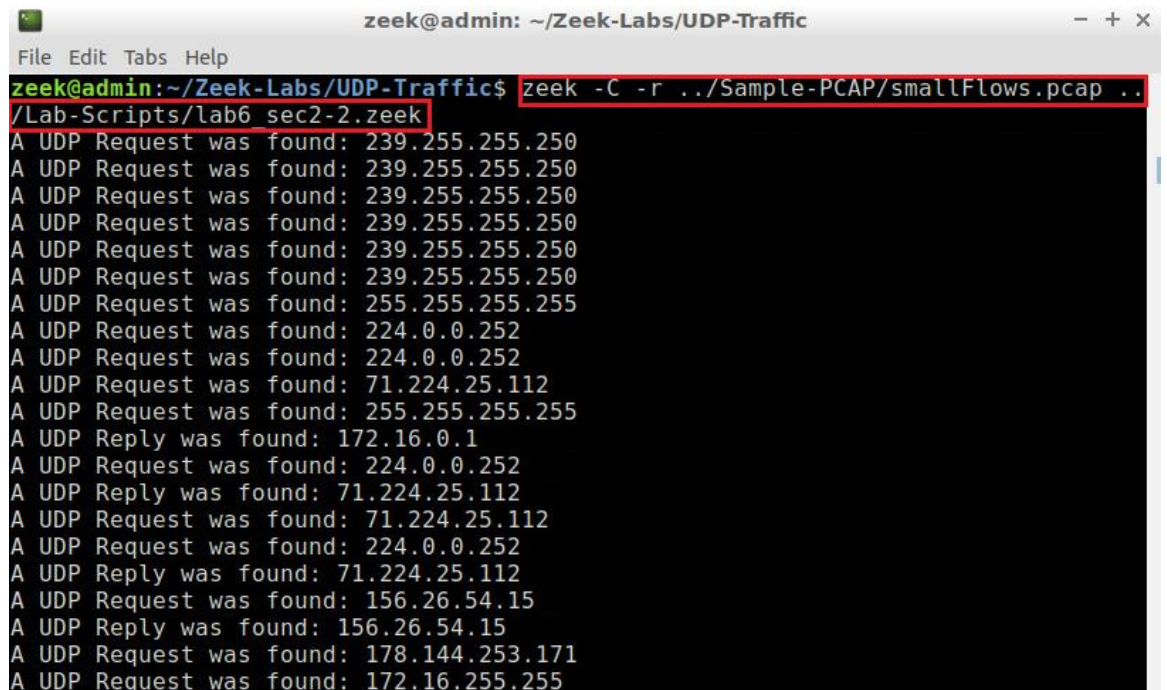
Step 3. Navigate to the *UDP-Traffic* workspace directory.

```
cd Zeek-Labs/UDP-Traffic/
```



Step 4. Process a packet capture file using the Zeek script. It is possible to use the `tab` key to autocomplete the longer paths.

```
zeek -C -r ../Sample-PCAP/smallFlows.pcap ../Lab-Scripts/lab6_sec2-2.zeek
```



```

A UDP Request was found: 239.255.255.250
A UDP Request was found: 239.255.255.250
A UDP Request was found: 239.255.255.250
A UDP Request was found: 239.255.255.250
A UDP Request was found: 239.255.255.250
A UDP Request was found: 239.255.255.250
A UDP Request was found: 255.255.255.255
A UDP Request was found: 224.0.0.252
A UDP Request was found: 224.0.0.252
A UDP Request was found: 71.224.25.112
A UDP Request was found: 255.255.255.255
A UDP Reply was found: 172.16.0.1
A UDP Request was found: 224.0.0.252
A UDP Reply was found: 71.224.25.112
A UDP Request was found: 71.224.25.112
A UDP Request was found: 224.0.0.252
A UDP Reply was found: 71.224.25.112
A UDP Request was found: 156.26.54.15
A UDP Reply was found: 156.26.54.15
A UDP Request was found: 178.144.253.171
A UDP Request was found: 172.16.255.255

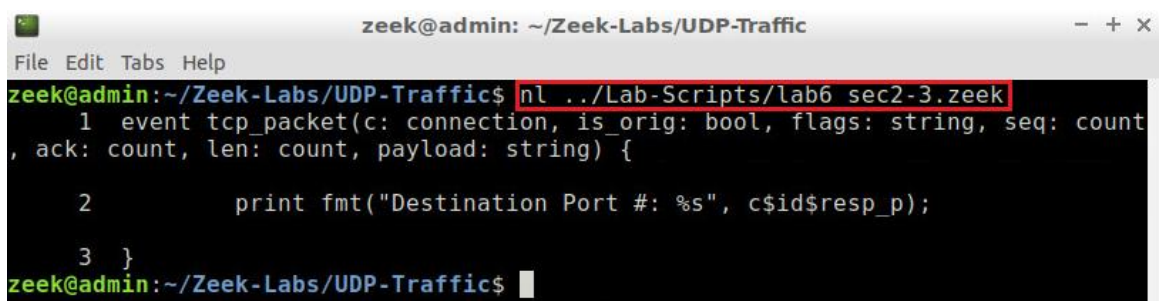
```

The packet capture file is processed into output log files. Since we did not create a new log stream, the script's output is displayed on the standard output (the screen). When `udp_request` or `udp_reply` events are triggered, the resulting packet information is displayed.

2.3 Executing a TCP Zeek script

Step 1. Display the content of the *lab6_sec2-3.zeek* Zeek script using `nl` command. `nl` shows the line numbers in the file. It is possible to use the `tab` key to autocomplete the longer paths.

```
nl ../Lab-Scripts/lab6_sec2-3.zeek
```



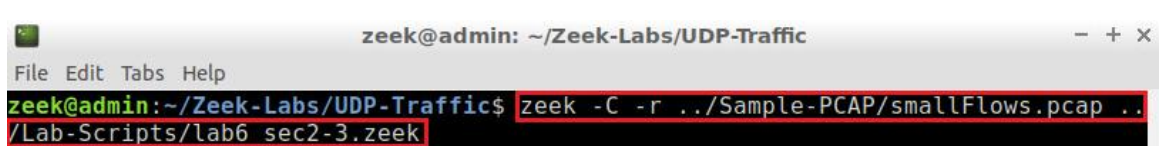
```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ nl ../Lab-Scripts/lab6_sec2-3.zeek
1 event tcp_packet(c: connection, is_orig: bool, flags: string, seq: count
, ack: count, len: count, payload: string) {
2     print fmt("Destination Port #: %s", c$id$resp_p);
3 }
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

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

1. Event `tcp_packet` is activated when a packet containing a TCP header is processed. The related packet header information is stored in the connection data structure passed to the function through the `u` variable. Additional TCP-related information is passed in a similar manner.
2. Prints the specified string. `%s` is a format specifier for strings with `fmt`. It indicates the position of the corresponding variable's information in the string. `uidresp_h` retrieves the destination IP address from the TCP packet.
3. End of the `tcp_packet` event.

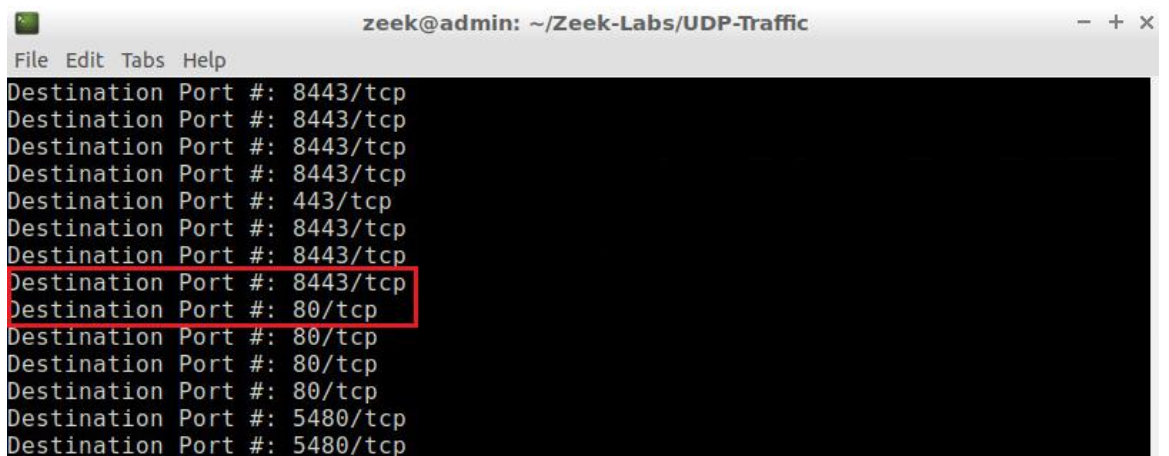
Step 2. Process a packet capture file using the Zeek script. It is possible to use the `tab` key to autocomplete the longer paths.

```
zeek -C -r ../Sample-PCAP/smallFlows.pcap ../Lab-Scripts/lab6_sec2-3.zeek
```



```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ zeek -C -r ../Sample-PCAP/smallFlows.pcap ..
/Lab-Scripts/lab6_sec2-3.zeek
```

The following output is produced:



```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
Destination Port #: 8443/tcp
Destination Port #: 8443/tcp
Destination Port #: 8443/tcp
Destination Port #: 8443/tcp
Destination Port #: 443/tcp
Destination Port #: 8443/tcp
Destination Port #: 8443/tcp
Destination Port #: 8443/tcp
Destination Port #: 80/tcp
Destination Port #: 80/tcp
Destination Port #: 80/tcp
Destination Port #: 80/tcp
Destination Port #: 5480/tcp
Destination Port #: 5480/tcp
```

When the `tcp_packet` event is triggered, the resulting packet information is displayed. Highlighted is an example of Port 8443 and Port 80 traffic.

These examples highlight Zeek's capabilities of tracking specific traffic. For instance, a script can be designed to collect all Port 80 traffic daily and to export it to a log file. In the following section we introduce log streams.

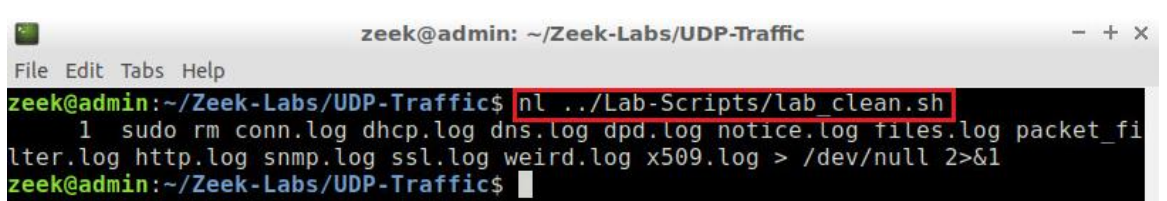
3 Modifying Zeek log streams

Zeek log streams determine where an event's output will be returned, as well as how it is formatted. It is possible to append new streams, modify default streams, or remove streams.

Before continuing, we must clear the lab workspace directory.

Step 1. Display the contents of the `lab_clean.sh` shell script using `nl` command.

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

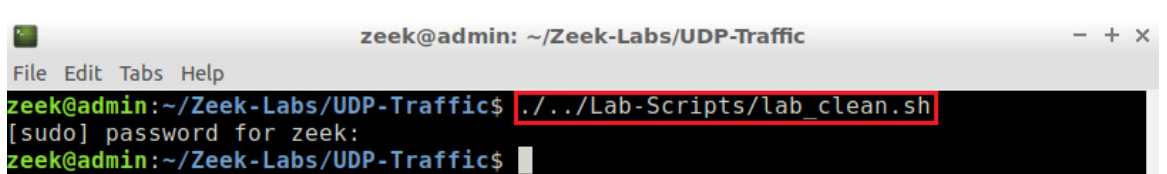


```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ nl ../Lab-Scripts/lab_clean.sh
1 sudo rm conn.log dhcp.log dns.log dpd.log notice.log files.log packet_fi
lter.log http.log snmp.log ssl.log weird.log x509.log > /dev/null 2>&1
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

The shell script removes a list of files expected to be generated by Zeek's processing using default log streams. Executing this shell script will clear the directory of log files generated previously. Output messages from running this script as none displayed in the Terminal, instead the code `> /dev/null 2>&1` will set errors and notices to be sent to a null folder, effectively eliminating them.

Step 2. Execute the `lab_clean.sh` shell script. It is possible to use the `tab` key to autocomplete the longer paths. If required, type `password` as the password.

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



```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ ../Lab-Scripts/lab_clean.sh
[sudo] password for zeek:
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

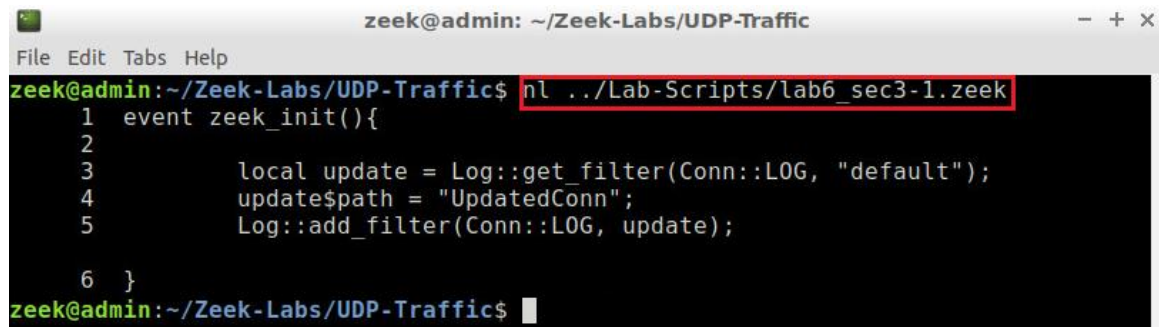
With the workspace directory cleared, we can move to the next section.

3.1 Renaming the conn.log stream

In this example, we will rename the *conn.log* file to be *UpdatedConn.log*. Renaming log streams can help with files organization, especially if a log file has been modified from its original functionality.

Step 1: Display the contents of the *lab6_sec3-1.zeek* Zeek script using the `nl` command. It is possible to use the `tab` key to autocomplete the longer paths.

```
nl ../Lab-Scripts/lab6_sec3-1.zeek
```



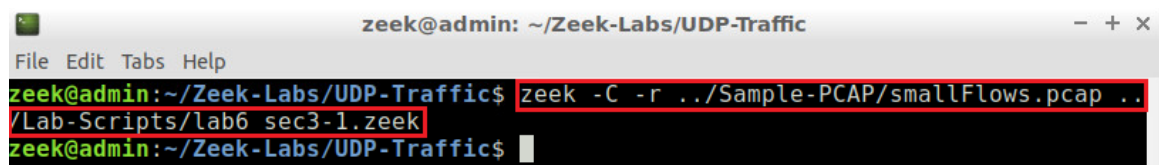
```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ nl ../Lab-Scripts/lab6_sec3-1.zeek
1  event zeek_init(){
2
3      local update = Log::get_filter(Conn::LOG, "default");
4      update$path = "UpdatedConn";
5      Log::add_filter(Conn::LOG, update);
6  }
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

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

1. Event `zeek_init` is activated when Zeek is first initialized.
3. Creates a local variable `update` initialized to the default `Conn::LOG` filter.
4. Sets the `update` variable's path to *UpdatedConn.log*.
5. Appends the new filter to the active log streams.
6. End of the `zeek_init` event.

Step 2. Process a packet capture file using the Zeek script. It is possible to use the `tab` key to autocomplete the longer paths.

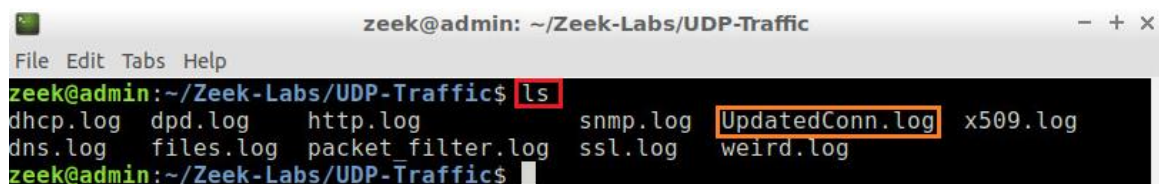
```
zeek -C -r ../Sample-PCAP/smallFlows.pcap ../Lab-Scripts/lab6_sec3-1.zeek
```



```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ zeek -C -r ../Sample-PCAP/smallFlows.pcap ..
../Lab-Scripts/lab6_sec3-1.zeek
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

Step 3. List the generated log files in the current directory.

```
ls
```



```
zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ ls
dhcp.log  dpd.log  http.log  snmp.log  UpdatedConn.log  x509.log
dns.log   files.log  packet  filter.log  ssl.log  weird.log
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

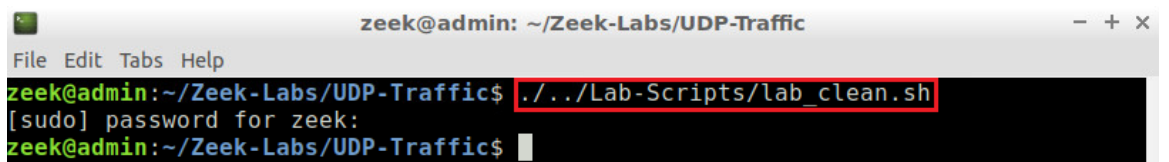
Note the *UpdatedConn.log*, highlighted by the orange box. Since we did not change any formatting, it is an exact replica of the original *conn.log* file.

3.2 Updating the conn.log stream

In this example, we modify the *conn.log* file to generate an additional *conn-http.log* file. This modification will split the *conn.log* contents between two log files, which is useful when organizing specific events – such as splitting UDP traffic from TCP traffic, or reply messages from requests.

Step 1. Execute the included *lab_clean.sh* shell script. If required, type `password` as the password. It is possible to use the `tab` key to autocomplete the longer paths.

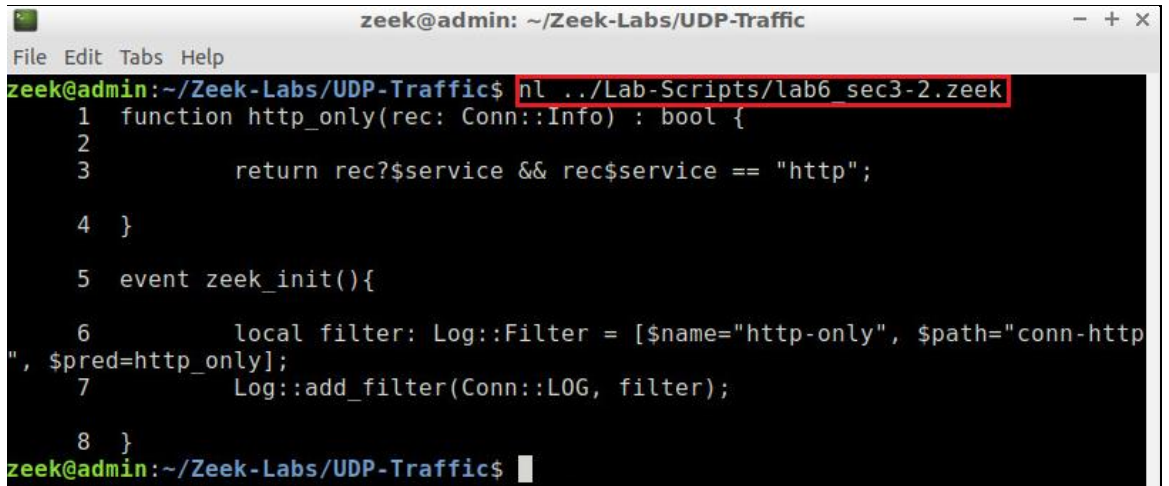
```
./../Lab-Scripts/lab_clean.sh
```



A terminal window titled "zeek@admin: ~/Zeek-Labs/UDP-Traffic" shows the command `./../Lab-Scripts/lab_clean.sh` being executed. The prompt changes to `[sudo] password for zeek:` and then back to `zeek@admin:~/Zeek-Labs/UDP-Traffic$` after the password is entered.

Step 2. Display the contents of *lab6_sec3-1.zeek* Zeek script using the `nl` command.

```
nl ../Lab-Scripts/lab6_sec3-2.zeek
```



A terminal window titled "zeek@admin: ~/Zeek-Labs/UDP-Traffic" shows the command `nl ../Lab-Scripts/lab6_sec3-2.zeek` being executed. The output displays the contents of the script with line numbers:

```
1 function http_only(rec: Conn::Info) : bool {
2
3     return rec?$service && rec$service == "http";
4 }
5 event zeek_init(){
6     local filter: Log::Filter = [$name="http-only", $path="conn-http", $pred=http_only];
7     Log::add_filter(Conn::LOG, filter);
8 }
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

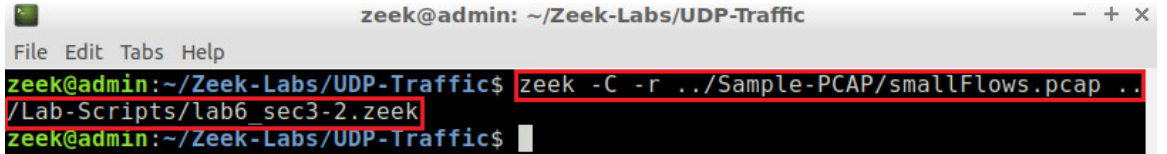
The script is explained as follows. Each number represents the respective line number:

1. Boolean function that has the parameter `rec`, an instance of `Conn::Info`.
3. Returns True if the service stored in `rec` is the HTTP protocol.
4. End of the function.
5. Event `zeek_init` is activated when Zeek is first initialized.
6. Creates a local filter with *http* related naming and pathing.

7. Appends the new filter to the active log streams.
8. End of the `zeek_init` event.

Step 2: Process a packet capture file using the Zeek script. It is possible to use the `tab` key to autocomplete the longer paths.

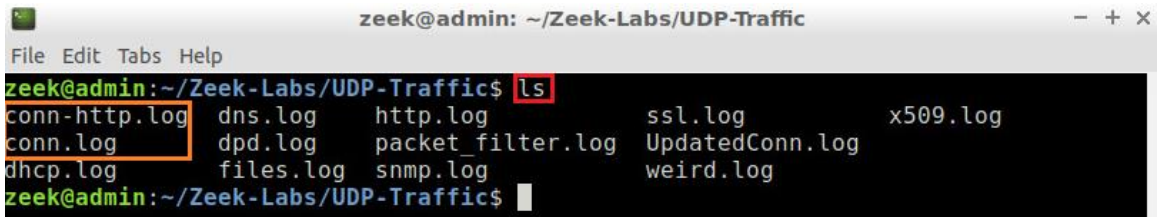
```
zeek -C -r ../Sample-PCAP/ smallFlows.pcap ../Lab-Scripts/lab6_sec3-2.zeek
```



```
zeek@admin:~/Zeek-Labs/UDP-Traffic$ zeek -C -r ../Sample-PCAP/smallFlows.pcap ../Lab-Scripts/lab6_sec3-2.zeek
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

Step 3: List the the generated log files in the current directory.

```
ls
```



```
zeek@admin:~/Zeek-Labs/UDP-Traffic$ ls
conn-http.log  dns.log      http.log      ssl.log       x509.log
conn.log       dpd.log     packet_filter.log UpdatedConn.log
dhcp.log       files.log   snmp.log      weird.log
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

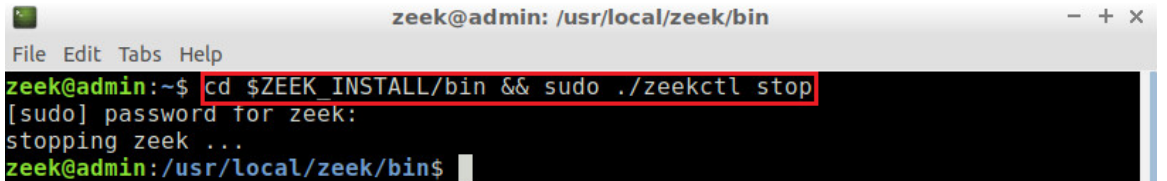
Note the `conn-http.log` file in the first column. This file will have the same formatting as the `conn.log` file; however, it will only contain HTTP traffic. These files are highlighted by the orange box in the proceeding image.

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
```



```
zeek@admin:~/Zeek-Labs/UDP-Traffic$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin:~/Zeek-Labs/UDP-Traffic$
```

Concluding this lab, we have introduced the Zeek scripting language. Using event-driven functionality, Zeek scripts can be used to customize the output log streams. Besides renaming existing files, you can also split the files to generate a more protocol or event-specific log file. Zeek scripts are the backbone of creating an organized workspace for storing and parsing generated log files.

References

1. "Logging framework", Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/frameworks/logging.html#streams>
2. "Monitoring HTTP traffic", Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/examples/httpmonitor/>
3. "Writing scripts", Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/examples/scripting/#the-event-queue-and-event-handlers>.



The University of Texas at San Antonio™
The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 7: Introduction to Zeek Signatures

Document Version: **03-13-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to Zeek signatures.....	4
1.1 Zeek signature format	5
1.2 Creating and using Zeek signatures	5
1.3 Zeek’s default signature framework	6
2 Log file analysis using Zeek signatures.....	8
2.1 Starting a new instance of Zeek	8
2.2 Viewing a premade Zeek signature file	9
2.3 Executing the premade Zeek signature file.....	10
3 Executing Zeek signature matching for network traffic analysis.....	12
3.1 Modifying the premade Zeek signature file	12
3.2 Executing the updated Zeek signature file.....	13
3.3 Closing the current instance of Zeek.....	15
References	16

Overview

This lab covers Zeek’s signature framework language. It introduces what network traffic signatures are and how they are matched to identify specific network events. This lab then reviews premade signature files and provides example usage for analysis.

Objectives

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

1. Develop signatures using Zeek’s signature framework.
2. Analyze processed log files using Zeek signatures.
3. Modify log streams for creating additional events and notices based on signatures.

Lab topology

Figure 1 shows the lab workspace topology. This lab primarily uses the *Client* machine for offline Zeek script development and offline packet capture processing and analysis.

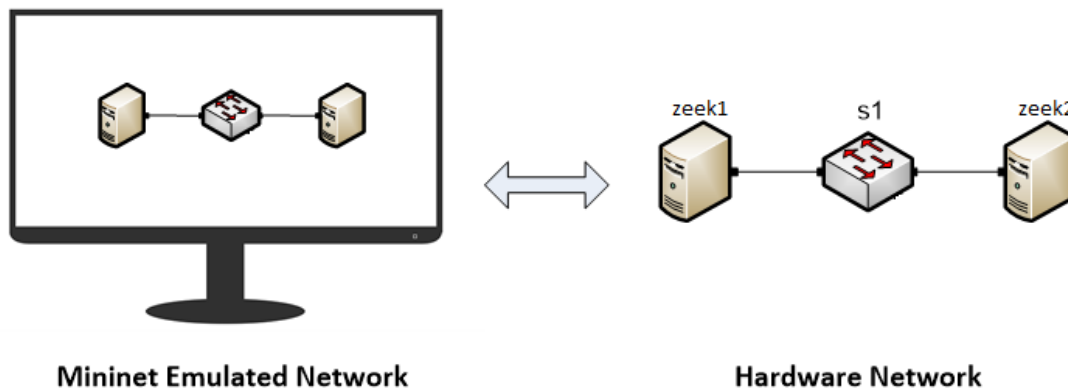


Figure 1. Lab topology.

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: Introduction to Zeek signatures.
2. Section 2: Log file analysis using Zeek signatures.
3. Section 3: Modifying Zeek signatures for advanced pattern matching.

1 Introduction to Zeek signatures

Following the introduction of developing and implementing basic Zeek scripts, we can now begin generating Zeek signatures. Introduced in the beginning of this lab series, the Zeek event-based engine is the primary architecture for running Zeek as an efficient intrusion detection system. The Zeek event-based engine predominantly utilizes the extensive scripting language to develop policies in order to define the steps and notifications necessary to handle anomalies and exceptions.

However, oftentimes it is simpler to create a predetermined string, known as a signature, and parse packet capture files for the specific signature. Because signatures are used for low-level pattern matching, the Zeek signature framework does not provide the same in-depth functionality as the Zeek scripting language for its event-based engine. Zeek signatures are used to quickly aggregate related network packets through signature matching before analysts can perform further, in-depth analysis on such traffic.

It is important to understand and be familiar with signatures due to their widespread usage across many related Intrusion Detection Systems and application-level firewalls. Separate from Zeek, many alternative IDS, such as the popular *Snort*, rely on signature-based pattern matching for anomaly and malicious event detection. Therefore, in operational cybersecurity environments that analyze network traffic to mitigate and prevent malicious events, understanding Zeek's signature framework adds an additional tool for developing a comprehensive IDS.

This lab will begin by introducing Zeek signatures, detailing their unique file type, how to load them into the Zeek event-based engine, and include a number of examples of leveraging signature matching for log file analysis.

1.1 Zeek signature format

The signature below depicts a basic network traffic signature. Depending on their usage, signatures can either include stricter requirements, or be more lax to encompass a larger portion of the processed data.

```

1 signature HTTP-sig {
2     ip-proto == tcp
3     dst-port == 80
4     payload /POST/
5     event "Found HTTP POST!"
6 }
```

1. This line defines a new *signature* object, with the name *HTTP-sig*.
2. Defines the desired match's transport protocol to be TCP.
3. Defines the desired match's destination port to be 80.
4. Defines the desired match's payload to contain the regular expression equivalent to 'POST'.
5. Defines an event if the match is found. Currently, the event will post a "HTTP Packet Found!" message; however, these events can be developed with a more complex functionality if the need arises.

This signature can be loaded into the Zeek signature framework during network traffic analysis, in which Zeek will attempt to match packets with the signature's details. While each individual packet can only be matched one time, multiple signatures can be

Additional signatures and their included variables are outlined and explained in Zeek's official documentation. 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/current/frameworks/signatures.html>

1.2 Creating and using Zeek signatures

Similar to Zeek's policy scripting framework, Zeek signatures are saved in separate files denoted by the `.sig` file extension. There are three ways to initialize Zeek for network traffic analysis while leveraging the Zeek signature framework:

1. When initializing Zeek from the terminal, include the additional `-s` option:

```
zeek -r <pcap file location> -s <signature file location>
```

- `zeek`: command to invoke Zeek.
- `-r`: option signifies to Zeek that it will be reading from an offline file.
- `<pcap file location>`: indicates the pcap file location.
- `-s`: option signifies to Zeek that the next file contains signatures.

- `<script_location>`: indicates the script location.
2. When creating a Zeek policy script, include the `@load-sigs` directive:

```

1  @load-sigs
2
3  module ZeekScript;
4
5  export{
6      /* Append and define new log stream parameters */
7  }
```

3. When creating a Zeek policy script, extend the Zeek global `signature_files` variable by appending the `+=` operator followed by the signature file:

```

1  @load-sigs
2
3  module ZeekScript;
4
5  redef signature_files += "signature_file_path.sig"
```

1.3 Zeek's default signature framework

This section introduces the default Zeek signature file that is compiled and included after Zeek has been installed.

While this default Zeek script includes scan-based detection, it will not correctly identify every unique anomaly that may be encountered. However, it does provide a comprehensive starter code that can be reviewed and customized to understand the Zeek signature framework.

The default Zeek signature file is named `main.zeek`. 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/current/scripts/base/frameworks/signatures/main.zeek.html
```

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

```

1 type Action: enum {
2     ## Ignore this signature completely (even for scan detection).
3     ## Don't write to the signatures logging stream.
4     SIG_IGNORE,
5     ## Process through the various aggregate techniques, but don't
6     ## report individually and don't write to the signatures logging
7     ## stream.
8     SIG_QUIET,
9     ## Generate a notice.
10    SIG_LOG,
11    ## The same as :zeek:enum:`Signatures::SIG_LOG`, but ignore for
12    ## aggregate/scan processing.
13    SIG_FILE_BUT_NO_SCAN,
14    ## Generate a notice and set it to be alarmed upon.
15    SIG_ALARM,
16    ## Alarm once per originator.
17    SIG_ALARM_PER_ORIG,
18    ## Alarm once and then never again.
19    SIG_ALARM_ONCE,
20    ## Count signatures per responder host and alarm with the
21    ## :zeek:enum:`Signatures::Count_Signature` notice if a threshold
22    ## defined by :zeek:id:`Signatures::count_thresholds` is reached.
23    SIG_COUNT_PER_RESP,
24    ## Don't alarm, but generate per-orig summary.
25    SIG_SUMMARY,
26 };

```

The figure above shows the options for signature match events within the *ZeekSignatureFramework.zeek* file. The options are explained as follows. Each number represents the respective line number:

4. `SIG_IGNORE`: if a signature is matched, do not write to the logging stream.
8. `SIG_QUIET`: if a signature is matched, process the included events but do not write to the logging stream.
10. `SIG_LOG`: if a signature is matched, generate a notice.
13. `SIG_FILE_BUT_NO_SCAN`: if a signature is matched and does not meet scan thresholds, write to the logging stream.
15. `SIG_ALARM`: if a signature is matched, generate a notice and set an alarm.
17. `SIG_ALARM_PER_ORIG`: if a signature is matched, generate a notice and set an alarm once per host that triggered the match.
19. `SIG_ALARM_ONCE`: if a signature is matched, generate a notice and set an alarm only one time, no matter the number of matches.
23. `SIG_COUNT_PER_RESP`: if a signature is matched, create a running count per responder host to compare against developed thresholds to identify and exclude scan traffic.
23. `SIG_SUMMARY`: generate a summary of all matched signatures based on the unique hosts that triggered a signature match.

Additional options and signature-specific events can be created using the Zeek scripting framework. Furthermore, Lab 8 of this series will enumerate upon the aforementioned scan thresholds and how Zeek determines if a host is probing a network.

```

1 type Info: record {
2     ## The network time at which a signature matching type of event
3     ## to be logged has occurred.
4     ts:         time           &log;
5     ## A unique identifier of the connection which triggered the
6     ## signature match event.
7     uid:        string         &log &optional;
8     ## The host which triggered the signature match event.
9     src_addr:   addr           &log &optional;
10    ## The host port on which the signature-matching activity
11    ## occurred.
12    src_port:   port           &log &optional;
13    ## The destination host which was sent the payload that
14    ## triggered the signature match.
15    dst_addr:   addr           &log &optional;
16    ## The destination host port which was sent the payload that
17    ## triggered the signature match.
18    dst_port:   port           &log &optional;
19    ## Notice associated with signature event.
20    note:       Notice::Type &log;
21    ## The name of the signature that matched.
22    sig_id:     string         &log &optional;
23    ## A more descriptive message of the signature-matching event.
24    event_msg:  string         &log &optional;
25    ## Extracted payload data or extra message.
26    sub_msg:    string         &log &optional;
27    ## Number of sigs, usually from summary count.
28    sig_count:  count          &log &optional;
29    ## Number of hosts, from a summary count.
30    host_count: count          &log &optional;
31 };

```

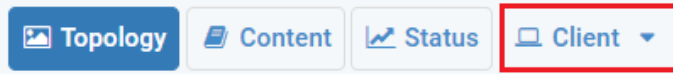
The figure above shows the variables that store signature-specific packet information accessed in the *ZeekSignatureFramework.zeek* file. These variables can be accessed to extract the stored information for notifications and warnings. Furthermore, each variable can be printed to the logging stream, following the Zeek log file format reviewed in previous labs. Each variable is explained by its preceding comments, denoted by the `##` character.

2 Log file analysis using Zeek signatures

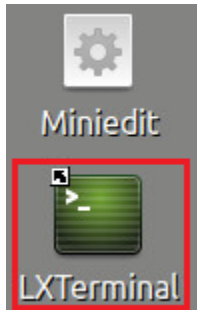
With Zeek's signature framework, we can create specific pattern-based signature filters to be applied during packet capture analysis. This section shows example signatures and their usage for network analysis.

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
```

```
zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
[sudo] password for zeek:
starting zeek ...
zeek@admin:~/usr/local/zeek/bin$
```

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

2.2 Viewing a premade Zeek signature file

Step 1. Navigate to the *Lab-Scripts* directory.

```
cd Zeek-Labs/Lab-Scripts/
```

```
zeek@admin: ~/Zeek-Labs/Lab-Scripts
File Edit Tabs Help
zeek@admin:~$ cd Zeek-Labs/Lab-Scripts/
zeek@admin:~/Zeek-Labs/Lab-Scripts$
```

Step 2: Display the contents of the `lab7_sec2-2.sig` file using `nl`.

```
nl lab7_sec2-2.sig
```



```

zeek@admin: ~/Zeek-Labs/Lab-Scripts
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/Lab-Scripts$ nl lab7_sec2-2.sig
 1 signature HTTP-POST-sig{
 2     ip-proto == tcp
 3     dst-port == 80
 4     payload /POST/
 5     event "Found HTTP Post"
 6 }
 7 signature HTTP-GET-sig{
 8     ip-proto == tcp
 9     dst-port == 80
10     payload /GET/
11     event "Found HTTP Request"
12 }
zeek@admin:~/Zeek-Labs/Lab-Scripts$

```

This signature file contains two signatures to be matched during network traffic analysis and is explained as follows. Each number represents the respective line number:

1. This line defines a new *signature* object, with the name *HTTP-POST-sig*.
2. Defines the desired match's transport protocol to be TCP.
3. Defines the desired match's destination port to be 80.
4. Defines the desired match's payload to contain the regular expression equivalent to 'POST'.
5. Defines an event if the match is found. Currently, the event will post a "Found HTTP Post" message.
7. This line defines a new *signature* object, with the name *HTTP-GET-sig*.
8. Defines the desired match's transport protocol to be TCP.
9. Defines the desired match's destination port to be 80.
10. Defines the desired match's payload to contain the regular expression equivalent to 'GET'.
11. Defines an event if the match is found. Currently, the event will post a "Found HTTP Request" message.

2.3 Executing the premade Zeek signature file

Step 1. Navigate to the *TCP-Traffic* directory.

```

cd ../TCP-Traffic/

```

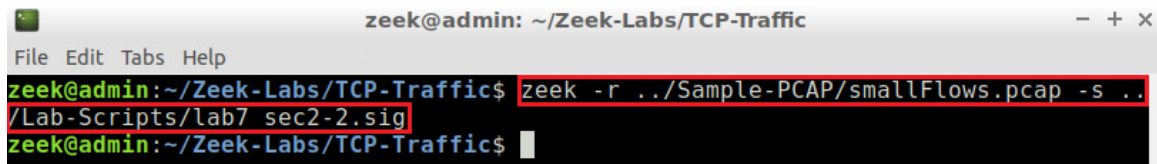
```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/Lab-Scripts$ cd ../TCP-Traffic/
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

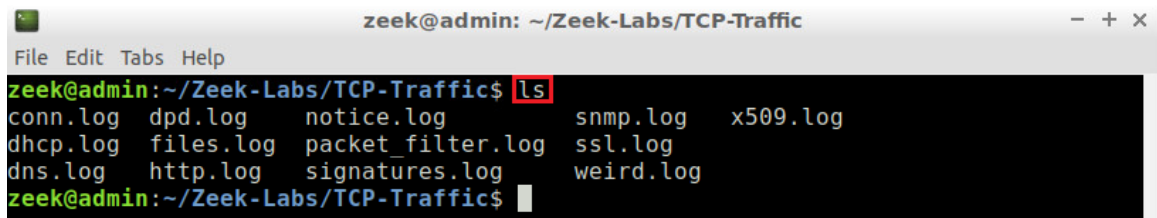
Step 2. Process the *smallFlows.pcap* packet capture file using the signature file *lab7_sec2-2.sig*. It is possible to use the `tab` key to autocomplete the longer paths.

```
zeek -r ../Sample-PCAP/smallFlows.pcap -s ../Lab-Scripts/lab7_sec2-2.sig
```



Step 3: List the generated log files in the current directory.

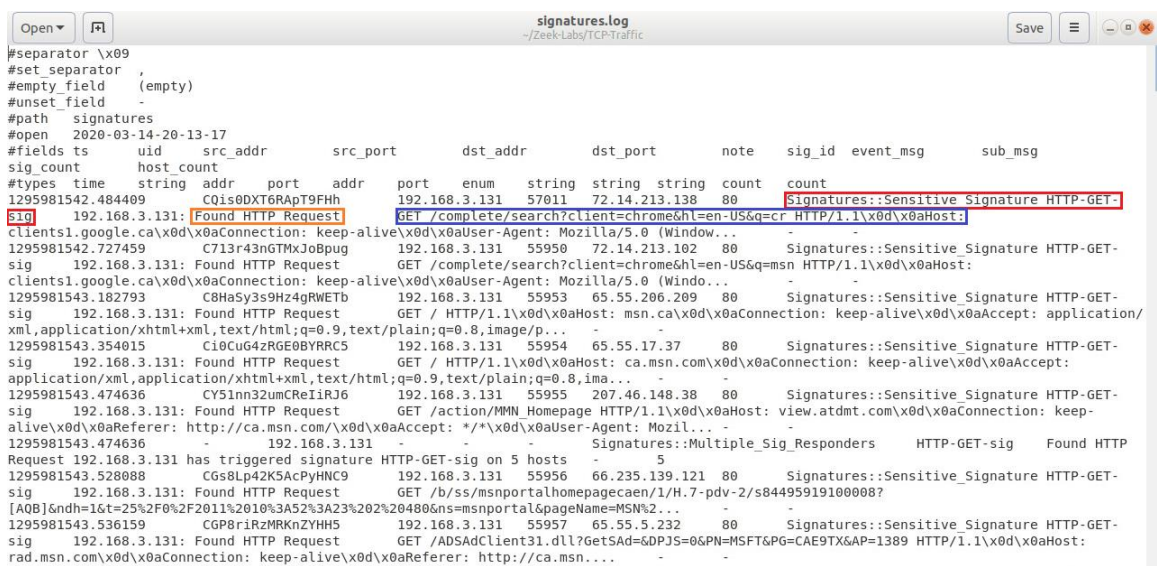
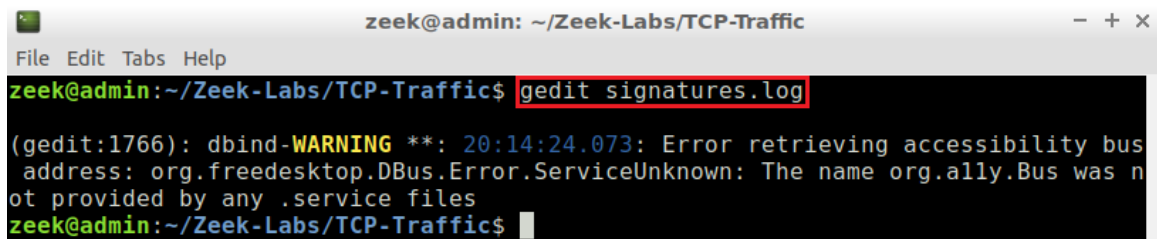
```
ls
```



A new log file that has not been previously introduced is now displayed: *signatures.log*. This log file will contain all signature matches and their corresponding events and notices.

Step 4: View the contents of the *signatures.log* file using the `gedit` text editor.

```
gedit signatures.log
```



The file is explained as follows:

- The red box indicates the name of the signature that was matched.
- The orange box indicates the event or message that was included when defining the signature.
- The blue box indicates the packet payload that was matched against the input signatures.

Step 6: Clear the contents of the *TCP-Traffic* directory.

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

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ../../Lab-Scripts/lab_clean.sh
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

3 Executing Zeek signature matching for network traffic analysis

This section modifies the existing signature file to generate additional signature events and notices. We will be modifying the previous signatures from TCP-based HTTP messages to UDP-based SNMP and DNS messages.

3.1 Modifying the premade Zeek signature file

Step 1: View the contents of the *lab7_sec3-1.sig* file using `nl`.

```
nl ../Lab-Scripts/lab7_sec3-1.sig
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ nl ../Lab-Scripts/lab7_sec3-1.sig
1 signature HTTP-POST-sig{
2     ip-proto == tcp
3     dst-port == 80
4     payload /POST/
5     event "Found HTTP Post"
6 }
7 signature HTTP-GET-sig{
8     ip-proto == tcp
9     dst-port == 80
10    payload /GET/
11    event "Found HTTP Request"
12 }
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

Step 2: Open the *lab7_sec3-1.sig* file with the `gedit` text editor.

```
gedit ../Lab-Scripts/lab7_sec3-1.sig
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ gedit ../Lab-Scripts/lab7_sec3-1.sig
(gedit:1904): dbind-WARNING **: 20:20:33.292: Error retrieving accessibility bus
address: org.freedesktop.DBus.Error.ServiceUnknown: The name org.aally.Bus was n
ot provided by any .service files
(gedit:1904): Gtk-WARNING **: 20:20:33.341: Attempting to read the recently used
resources file at '/home/zeek/.local/share/recently-used.xbel', but the parser
failed: Failed to open file "/home/zeek/.local/share/recently-used.xbel": Perm
ission denied.
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Step 3: Update the `lab7_sec3-1.sig` file to include the following signatures.

```

signature SNMP-REQUEST-sig{
    ip-proto == udp
    dst-port == 161
    event "Found SNMP Request"
}
signature SNMP-RESPONSE-sig{
    ip-proto == udp
    dst-port == 52400
    event "Found SNMP Response"
}
signature DNS-REQUEST-sig{
    ip-proto == udp
    dst-port == 53
    event "Found DNS Request"
}

```

```

Open [ ] *lab7_sec3-1.sig
~/Zeek-Labs/Lab-Scripts
signature SNMP-REQUEST-sig{
    ip-proto == udp
    dst-port == 161
    event "Found SNMP Request"
}
signature SNMP-RESPONSE-sig{
    ip-proto == udp
    dst-port == 52400
    event "Found SNMP Response"
}
signature DNS-REQUEST-sig{
    ip-proto == udp
    dst-port == 53
    event "Found DNS Request"
}

```

3.2 Executing the updated Zeek signature file

Step 1. Process the `smallFlows.pcap` packet capture file using the signature file `lab7_sec3-1.sig`. It is possible to use the `tab` key to autocomplete the longer paths.

```

zeek -r ../Sample-PCAP/smallFlows.pcap -s ../Lab-Scripts/lab7_sec3-1.sig

```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek -r ../Sample-PCAP/smallFlows.pcap -s ../Lab-Scripts/lab7_sec3-1.sig
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

Step 2: List the generated log files in the current directory.

```
ls
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
conn.log  dpd.log  notice.log  snmp.log  x509.log
dhcp.log  files.log  packet_filter.log  ssl.log
dns.log   http.log  signatures.log  weird.log
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

The *signatures.log* file has been recreated and will contain the newly updated signature matches.

Step 3: View the contents of the *signatures.log* file using the `gedit` text editor.

```
gedit signatures.log
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ gedit signatures.log
(gedit:1442): dbind-WARNING **: 14:44:48.194: Error retrieving accessibility bus address: org.freedesktop.DBus.Error.ServiceUnknown: The name org.a11y.Bus was not provided by any .service files
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

```

Open [ ] signatures.log
~/Zeek-Labs/TCP-Traffic Save [ ] [ ] [ ]
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path signatures
#open 2020-03-15-14-42-18
#fields ts uid src_addr src_port dst_addr dst_port note sig_id event_msg sub_msg
sig_count host count
#types time string addr port addr port enum string string string count count
1295981655.843173 Ch4Ifz23dedK6z0QMf 10.0.2.15 49796 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981655.926096 C6htq02iBqdCN3tAE5 10.0.2.15 50559 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981658.781806 CTzqJp1sof9eUhmMpa 10.0.2.15 54657 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981658.854004 CSvx41ltVLQaGd7DI5 10.0.2.15 57524 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981659.567918 CyUNWQ14mIGNAM7j3j 10.0.2.15 54795 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981659.783932 Cqh2xI1SLGXl20wJt4 10.0.2.15 61870 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981660.144937 CEI19BpY4uXc0D1ka 10.0.2.15 64982 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981663.533829 CBY6B02rdUV0XEIbI6 10.0.2.15 57632 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981664.266166 CSovQAF61YGPi6HiC 10.0.2.15 62310 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981664.492158 C9TKqM7eY4clREqyg 10.0.2.15 59794 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981665.894416 Cv38BD3vTNI7rARZue 10.0.2.15 58511 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981668.205003 Cn6chsp1hTmXLqTzc 10.0.2.15 58971 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981685.227252 CXkQnx1TuS8bqIRIma 10.0.2.15 59686 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981696.667788 CONHMjiCbbAgLibDa 10.0.2.15 61133 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981711.656223 CLWqCIApK3BRCSg 10.0.2.15 59365 10.0.2.3 53 Signatures::Sensitive_Signature DNS-REQUEST-
sig 10.0.2.15: Found DNS Request (empty) - - - - -
1295981744.511002 CZx0Ux3gaud2WgVvxg 192.168.3.131 52400 192.168.3.99 161 Signatures::Sensitive_Signature SNMP-
REQUEST-sig 192.168.3.131: Found SNMP Request (empty) - - - - -
1295981744.570907 CZx0Ux3gaud2WgVvxg 192.168.3.99 161 192.168.3.131 52400 Signatures::Sensitive_Signature SNMP-
RESPONSE-sig 192.168.3.99: Found SNMP Response (empty) - - - - -

```

The file is explained as follows:

- The red box indicates the DNS-REQUEST-sig signature match as well as the triggered IP address and event message.
- The orange box indicates the SNMP-REQUEST-sig signature match as well as the triggered IP address and event message.
- The blue box indicates the SNMP-RESPONSE-sig signature match as well as the triggered IP address and event message.

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
```

```

zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin: /usr/local/zeek/bin$

```

Concluding this lab, we have introduced the Zeek signature framework. Leveraging pattern matching, Zeek signatures can be used to quickly discover packets that follow predetermined formats, while employing a low-level framework for generating warnings and notifications.

References

1. “Signature framework”, Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/frameworks/signatures.html>
2. “Logging framework”, Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/frameworks/logging.html#streams>
3. “Monitoring HTTP traffic”, Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/examples/httpmonitor/>
4. “Writing scripts”, Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/examples/scripting/#the-event-queue-and-event-handlers>.



The University of Texas at San Antonio™
The Cyber Center for Security and Analytics

ZEEK INTRUSION 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	3
Objectives.....	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Zeek’s default anomaly detection scripts	4
1.1 Zeek scan-event.....	4
1.2 Zeek bruteforce-event	6
2 Generating customized malicious network traffic.....	7
2.1 Starting a new instance of Zeek	7
2.2 Launching Mininet.....	8
2.3 Setting up the zeek2 virtual machine for live network capture	10
2.4 Using the zeek1 virtual machine for network scanning activities	11
2.4.1 Terminating live network capture	12
3 Applying Zeek scripts to filter network traffic	13
3.1 Applying the ZeekDetectScans filter	13
3.2 Applying the ScanFilter filter	15
3.3 Closing the current instance of Zeek.....	20
References	21

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 workspace topology. The *Client* machine will be used for offline Zeek script development, while the *zeek1* and *zeek2* virtual machines will generate and collect network traffic.

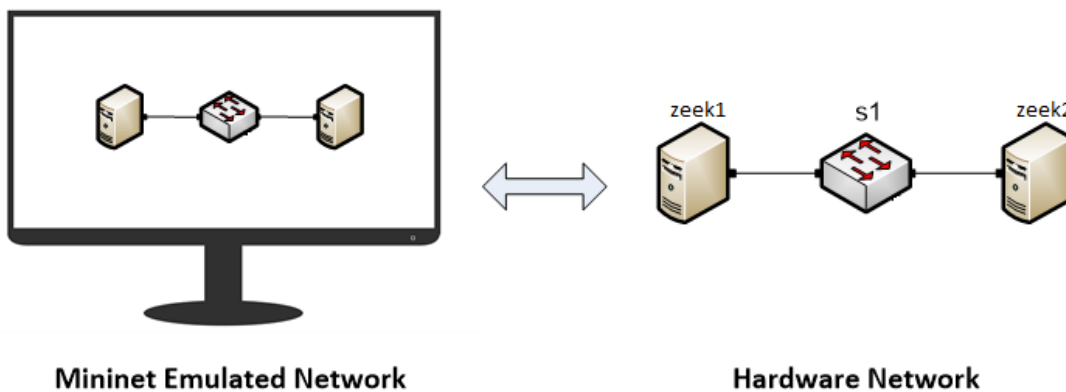


Figure 1. Lab topology.

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.zek* 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.zek* 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.zek.html
```

The file has been copied into the Zeek lab workspace directory and renamed to *ZeekDetectBruteForce.zek* 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:

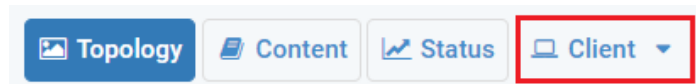
15. `const bruteforce_threshold`: threshold for the number of failed authentications attempts a source IP address can make. The default value is 20 failed attempts within the related time interval threshold.
18. `const bruteforce_measurement_interval`: threshold for the time to check a source IP address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

2 Generating customized malicious network traffic

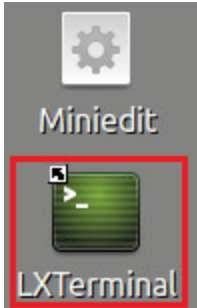
This section introduces creating and using a new Zeek script, tailored to react to more specific events.

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.



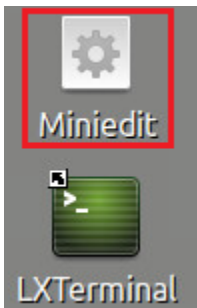
Step 3. Start Zeek by entering the following command on the terminal. This command enters Zeek’s default installation directory and invokes `zeekctl` tool to start a new instance. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key. When prompted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

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

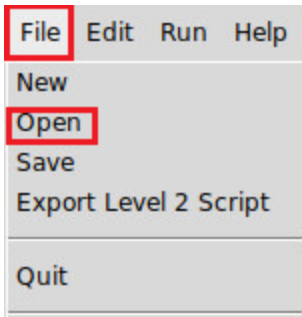
A new instance of Zeek is now active, and we are ready to proceed to the next section of the lab.

2.2 Launching Mininet

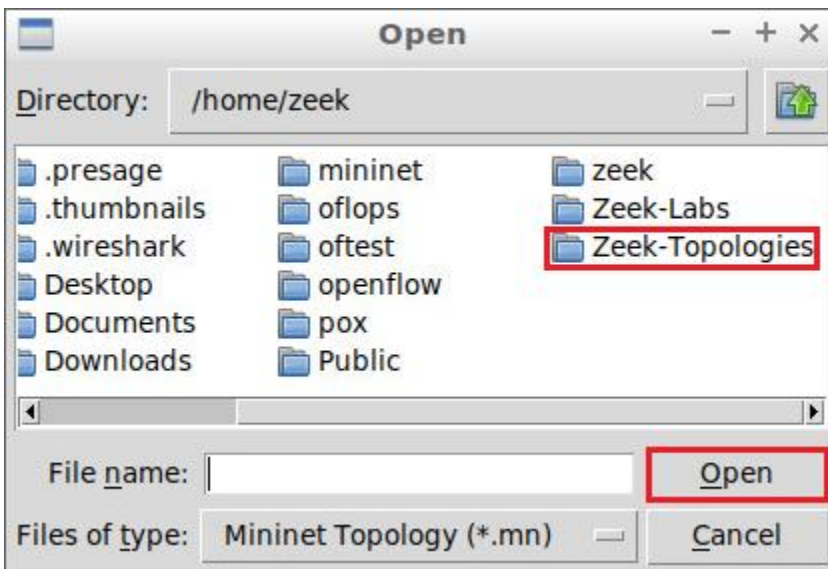
Step 1. From the *Client* machine’s desktop, on the left side of the screen, click on the MiniEdit icon as shown below. When prompted for a password, type `password` and hit `Enter`. The MiniEdit editor will now launch.



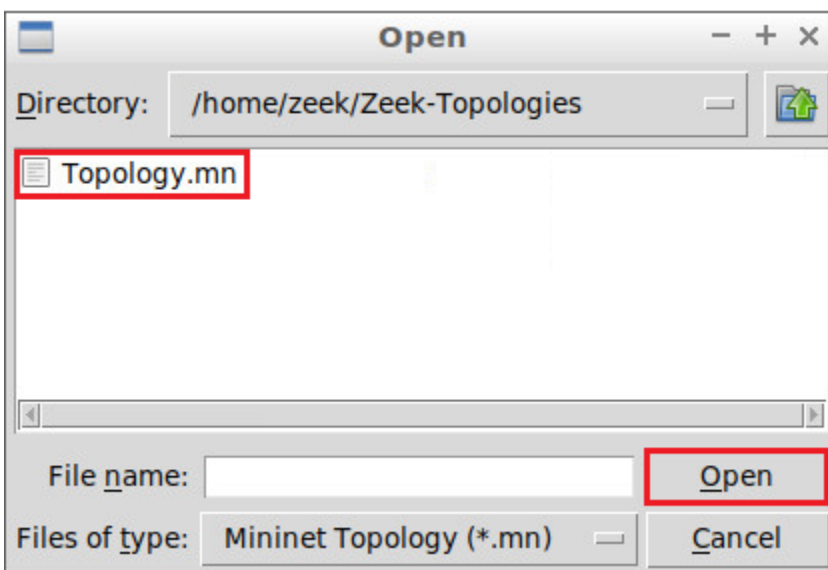
Step 2. The MiniEdit editor will now launch and allow for the creation of new, virtualized lab topologies. Load the correct topology by clicking the `Open` button within the `File` tab on the top left of the MiniEdit editor.



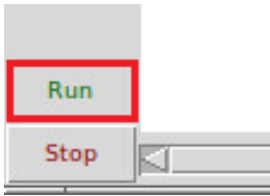
Step 3. Navigate to the Zeek-Topologies directory by scrolling to the right of the active directories and double clicking the Zeek-Topologies icon, or by clicking the `Open` button.



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

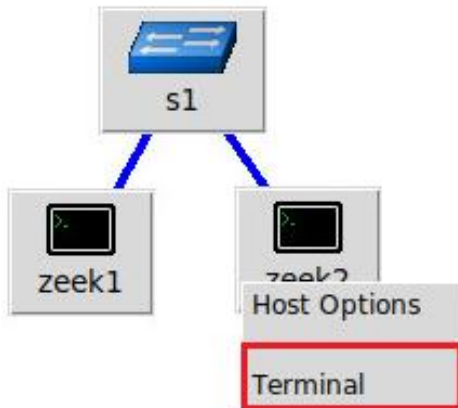


Step 5. To begin running the virtual machines, navigate to the `Run` button, found on the bottom left of the Miniedit editor, and select the `Run` button, as seen in the image below.



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/
```

A screenshot of a terminal window titled '"Host: zeek2"'. The terminal shows the prompt `root@admin:~#` followed by the command `cd Zeek-Labs/TCP-Traffic/` which has been executed. The prompt now shows `root@admin:~/Zeek-Labs/TCP-Traffic#`. The command and its execution are highlighted with a red box.

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

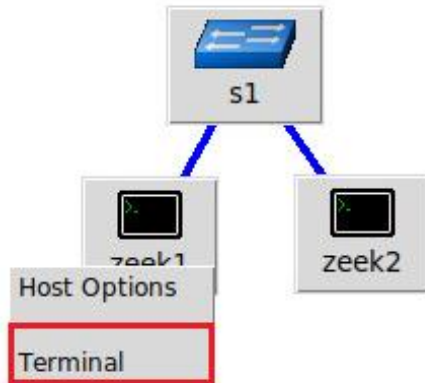
```
tcpdump -i zeek2-eth0 -s 0 -w scantraffic.pcap
```

A screenshot of a terminal window titled '"Host: zeek2"'. The terminal shows the prompt `root@admin:~/Zeek-Labs/TCP-Traffic#` followed by the command `tcpdump -i zeek2-eth0 -s 0 -w scantraffic.pcap` which has been executed. The command and its execution are highlighted with a red box. Below the command, the terminal output shows: `tcpdump: listening on zeek2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes`.

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
```

```

root@admin:~# nmap -sT 10.0.0.2
Starting Nmap 7.60 ( https://nmap.org ) at 2020-01-13 15:03 EST
Nmap scan report for 10.0.0.2
Host is up (0.000086s latency).
All 1000 scanned ports on 10.0.0.2 are closed
MAC Address: 0E:8E:F6:FB:EB:B8 (Unknown)
Nmap done: 1 IP address (1 host up) scanned in 13.33 seconds
root@admin:~#
  
```

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.

```

"Host: zeek1"
root@admin:~# nmap --scanflags SYN,FIN,RST 10.0.0.2
Starting Nmap 7.60 ( https://nmap.org ) at 2020-01-13 15:03 EST
Nmap scan report for 10.0.0.2
Host is up (0.00037s latency).
All 1000 scanned ports on 10.0.0.2 are filtered
MAC Address: 0E:8E:F6:FB:EB:B8 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 34.42 seconds
root@admin:~#
    
```

Step 5. 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
```

```

"Host: zeek1"
root@admin:~# nmap --scanflags SYN,RST,ACK 10.0.0.2
Starting Nmap 7.60 ( https://nmap.org ) at 2020-01-13 15:05 EST
Nmap scan report for 10.0.0.2
Host is up (0.00038s latency).
All 1000 scanned ports on 10.0.0.2 are filtered
MAC Address: 0E:8E:F6:FB:EB:B8 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 34.42 seconds
root@admin:~#
    
```

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.

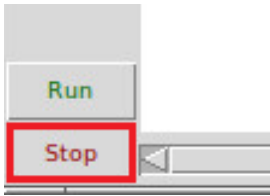


Step 2. Use the `Ctrl+c` key combination to stop live traffic capture. Statistics of the capture session will be displayed. 6014 packets were recorded by the interface, which were then captured and stored in the new *scantraffic.pcap* file.

```

"Host: zeek2"
root@admin:~/Zeek-Labs/TCP-Traffic# tcpdump -i zeek2-eth0 -s 0 -w scantraffic.pcap
tcpdump: listening on zeek2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C6014 packets captured
6014 packets received by filter
0 packets dropped by kernel
root@admin:~/Zeek-Labs/TCP-Traffic#
    
```

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.



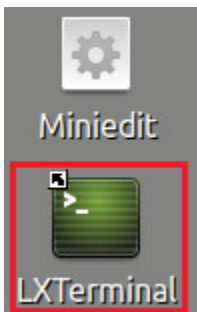
3 Applying Zeek scripts to filter network traffic

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

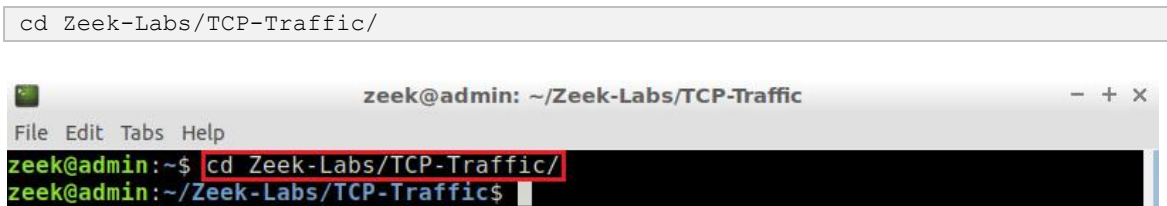
3.1 Applying the ZeekDetectScans filter

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.



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



```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
scantraffic.pcap
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

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
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek -C -r scantraffic.pcap ../Lab-Scripts
/ZeekDetectScans.zeek
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

Step 2: Display the contents of the *notice.log* file using the `cat` command.

```
cat notice.log
```

```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ cat notice.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path notice
#open 2020-01-13-15-11-43
#fields ts uid id.orig_h id.orig_p id.resp_h id.res
p_p fuidd file_mime_type file_desc proto note msg sub s
rc dst p n peer_descr actions suppress_for remote
_location.country_code remote_location.region remote_location.city remote
_location.latitude remote_location.longitude
#types time string addr port addr port string string string
enum enum string string addr addr port count string set[en
um] interval string string string double double
1578946016.741433 - - - - -
- Scan::Port Scan 10.0.0.1 scanned at least 15 unique ports of host 10.0
.0.2 in 0m0s remote 10.0.0.1 10.0.0.2 - - - N
notice::ACTION_LOG 3600.000000 - - - -
#close 2020-01-13-15-11-44
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

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 3: 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
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ head -n 25 conn.log | zeek-cut ts id.orig
h id.orig p id.resp h id.resp p history
1578946016.740376      10.0.0.1      35734      10.0.0.2      135      Sr
1578946016.740425      10.0.0.1      45382      10.0.0.2      8888     Sr
1578946016.740493      10.0.0.1      43620      10.0.0.2      995      Sr
1578946016.740525      10.0.0.1      36164      10.0.0.2      53       Sr
1578946016.740595      10.0.0.1      47598      10.0.0.2      22       Sr
1578946016.740663      10.0.0.1      40448      10.0.0.2      111      Sr
1578946016.740728      10.0.0.1      57284      10.0.0.2      110      Sr
1578946016.740793      10.0.0.1      37798      10.0.0.2      139      Sr
1578946016.740858      10.0.0.1      54886      10.0.0.2      554      Sr
1578946016.740924      10.0.0.1      54362      10.0.0.2      3389     Sr
1578946016.741140      10.0.0.1      58980      10.0.0.2      143      Sr
1578946016.741207      10.0.0.1      60390      10.0.0.2      5900     Sr
1578946016.741271      10.0.0.1      51092      10.0.0.2      199      Sr
1578946016.741336      10.0.0.1      44352      10.0.0.2      443      Sr
1578946016.741423      10.0.0.1      33332      10.0.0.2      256      Sr
1578946016.741487      10.0.0.1      54560      10.0.0.2      21       Sr
1578946016.741550      10.0.0.1      53068      10.0.0.2      445      Sr
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

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/ZeekFilter.zeek
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ nl ../Lab-Scripts/ScanFilter.zeek
 1  module SCAN;
 2
 3  export {
 4      redef enum Log::ID += {CASE1LOG};
 5      redef enum Log::ID += {CASE2LOG};
 6
 7      type outputFormat: record {
 8          ts:          time          &log;
 9          id:          conn_id      &log;
10         orig_h:      addr          &log;
11         orig_p:      port          &log;
12         resp_h:      addr          &log;
13         resp_p:      port          &log;
14         history:     string        &log   &optional;
15     };
  }

```

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.

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help

16 event zeek_init() {
17     Log::create_stream(CASE1LOG, [$columns=outputFormat, $path="Case1"]);
18     Log::create_stream(CASE2LOG, [$columns=outputFormat, $path="Case2"]);
19 }

20 event tcp_packet(c: connection, is_orig: bool, flags: string, seq: count, ack: count, len: count, payload: string) {

21     local rec: SCAN::outputFormat = [$ts=c$start_time, $id=c$id, $orig_h=c$id$orig_h, $orig_p=c$id$orig_p, $resp_h=c$id$resp_h, $resp_p=c$id$resp_p, $history=c$history];

22     if(flags == "SFR") {
23         Log::write(SCAN::CASE1LOG, rec);
24     }
25     if(flags == "SRA") {
26         Log::write(SCAN::CASE2LOG, rec);
27     }
28 }
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

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 `cid<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
```



```

zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ ../../Lab-Scripts/lab_clean.sh
[sudo] password for zeek:
zeek@admin:~/Zeek-Labs/UDP-Traffic$

```

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
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek -C -r scantraffic.pcap ../Lab-Scripts/ScanFilter.zeek
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

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

```
ls
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
Case1.log  conn.log      scantraffic.pcap
Case2.log  packet_filter.log weird.log
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

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
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ head -n 25 Case1.log | zeek-cut ts id.orig
h id.orig p id.resp h id.resp p history
1578946073.989214 10.0.0.1 56046 10.0.0.2 23 I
1578946073.989222 10.0.0.1 56046 10.0.0.2 199 I
1578946073.989223 10.0.0.1 56046 10.0.0.2 993 I
1578946073.989230 10.0.0.1 56046 10.0.0.2 1723 I
1578946073.989245 10.0.0.1 56046 10.0.0.2 3306 I
1578946073.989250 10.0.0.1 56046 10.0.0.2 1025 I
1578946073.989263 10.0.0.1 56046 10.0.0.2 22 I
1578946073.989282 10.0.0.1 56046 10.0.0.2 256 I
1578946073.989293 10.0.0.1 56046 10.0.0.2 8888 I
1578946073.989303 10.0.0.1 56046 10.0.0.2 21 I
1578946075.090251 10.0.0.1 56047 10.0.0.2 23 I
1578946075.090272 10.0.0.1 56047 10.0.0.2 21 I
1578946075.090282 10.0.0.1 56047 10.0.0.2 8888 I
1578946075.090292 10.0.0.1 56047 10.0.0.2 256 I
1578946075.090301 10.0.0.1 56047 10.0.0.2 3306 I
1578946075.090310 10.0.0.1 56047 10.0.0.2 1025 I
1578946075.090319 10.0.0.1 56047 10.0.0.2 22 I
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

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
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ head -n 25 Case2.log | zeek-cut ts id.orig
h id.orig p id.resp h id.resp p history
1578946034.589254 10.0.0.1 53710 10.0.0.2 995 Q
1578946034.589256 10.0.0.1 53710 10.0.0.2 143 Q
1578946034.589260 10.0.0.1 53710 10.0.0.2 587 Q
1578946034.589261 10.0.0.1 53710 10.0.0.2 135 Q
1578946034.589279 10.0.0.1 53710 10.0.0.2 80 Q
1578946034.589281 10.0.0.1 53710 10.0.0.2 53 Q
1578946034.589286 10.0.0.1 53710 10.0.0.2 1723 Q
1578946034.589290 10.0.0.1 53710 10.0.0.2 23 Q
1578946034.589292 10.0.0.1 53710 10.0.0.2 554 Q
1578946034.589306 10.0.0.1 53710 10.0.0.2 111 Q
1578946035.690266 10.0.0.1 53711 10.0.0.2 111 Q
1578946035.690287 10.0.0.1 53711 10.0.0.2 554 Q
1578946035.690297 10.0.0.1 53711 10.0.0.2 23 Q
1578946035.690307 10.0.0.1 53711 10.0.0.2 53 Q
1578946035.690316 10.0.0.1 53711 10.0.0.2 80 Q
1578946035.690325 10.0.0.1 53711 10.0.0.2 1723 Q
1578946035.690337 10.0.0.1 53711 10.0.0.2 995 Q
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

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.

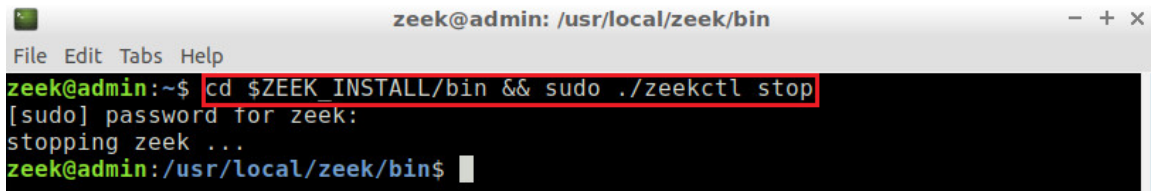
Unlike the default example, we can see the `history` column contains the exact same flag. Our filter was successful in organizing the traffic related to the `Case2` exploit.

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
```

A terminal window titled "zeek@admin: /usr/local/zeek/bin" with a menu bar (File, Edit, Tabs, Help). The terminal shows the command "cd \$ZEEK INSTALL/bin && sudo ./zeekctl stop" highlighted with a red box. The output is "[sudo] password for zeek:", "stopping zeek ...", and the prompt "zeek@admin: /usr/local/zeek/bin\$".

```
zeek@admin:~$ cd $ZEEK INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin: /usr/local/zeek/bin$
```

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 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>.



The University of Texas at San Antonio™
The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 9: Profiling and Performance Metrics of Zeek

Document Version: **03-13-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to Zeek profiling.....	4
2 Generating customized malicious network traffic.....	5
2.1 Starting a new instance of Zeek	5
2.2 Launching Mininet.....	6
2.3 Setting up the zeek2 virtual machine for live network capture	7
2.4 Using the zeek1 virtual machine for network scanning activities	8
2.4.1 Terminating live network capture	10
3 Generating and viewing Zeek profiling log files.....	11
3.1 Applying the profiling filter	11
4 Implementing tools to test Zeek's performance	14
4.1 Using sysstat sar utility.....	14
4.2 Using the top utility.....	16
4.3 Viewing the resource consumption of Zeek	16
4.4 Closing the current instance of Zeek.....	17
References	18

Overview

With Zeek’s event-based framework, anomalies can be detected, processed and analyzed with external software. In this lab, we explain Zeek’s profiling log stream and Zeek’s resource consumption.

Objectives

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

1. Enable Zeek’s profiling log stream for session-based statistics.
2. Generate customized traffic to be captured by Zeek’s profiling.
3. Implement tools necessary for testing Zeek’s resource consumption.

Lab topology

Figure 1 shows the lab workspace topology. The *Client* machine will be used for offline Zeek script development, while the *zeek1* and *zeek2* virtual machines will generate and collect network traffic.

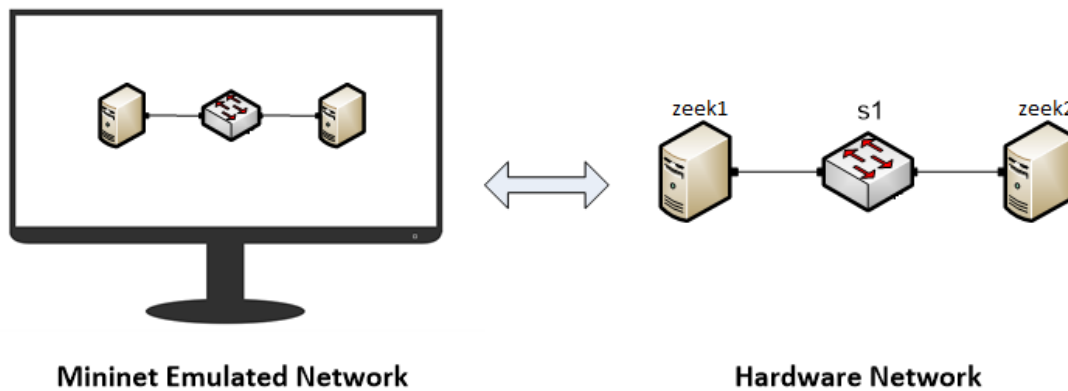


Figure 1. Lab topology.

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: Introduction to Zeek profiling.
2. Section 2: Generating customized malicious network traffic.
3. Section 3: Generating and viewing Zeek profiling log files.
4. Section 4: Implementing tools to test Zeek's performance.

1 Introduction to Zeek profiling

Zeek includes the option of enabling profiling. When profiling is enabled, a new log stream will be created to store session-related statistics. The Profile log file will contain a large variety of information, including but not limited to running time, memory usage, connection information and packet protocol statistics.

To enable profiling while using Zeek for offline packet capture file processing, you will need to implement the following functionality in a Zeek script.

```

1  module Profiling
2
3  redef profiling_file = open (fmt(<filename>, <logstream>));
4  redef profiling_interval = 3 secs;
5  redef expensive_profiling_multiple = 5;
6
7  event zeek_init() {
8      set_buf(profiling_file, F);
9  }
```

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

1. Sets the module workspace as *Profiling*.
3. Specifies the name of the new profiling log file, as well as determines the format based off an input log stream.
4. Specifies the time interval for Zeek to record empirical information. In this example the time interval is 3 seconds.

5. Specifies the number of profiling intervals defined in Line 5. In this example, the profiling interval is 5 instances.
7. Initialization event.
8. Appends the new log stream information.
9. End of initialization event.

Profiling is enabled by calling the Zeek script during packet processing, as reviewed in the previous labs.

```
zeek -r <packet capture file> <Profiling Script>
```

- `<packet capture file>`: denotes the input packet capture file.
- `<Profiling Script>`: denotes the Zeek script to be run during packet processing.

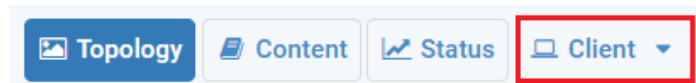
In the following section we generate customized malicious traffic to be viewed within a Zeek profiling log.

2 Generating customized malicious network traffic

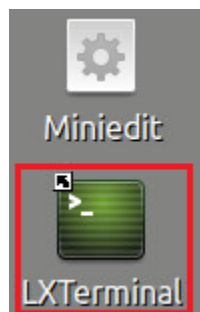
This section introduces creating and using a Zeek profiling script, which will enable session-based statistics for Zeek packet capture file processing.

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.



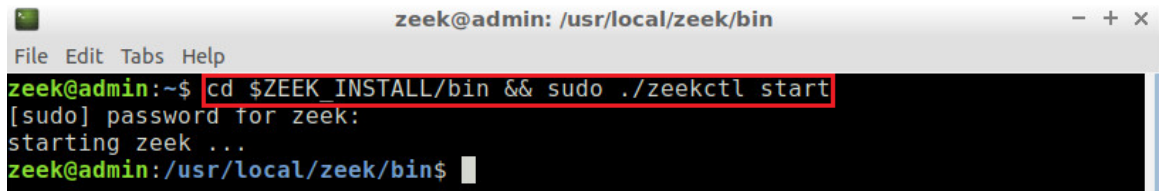
Step 2. The *Client* machine will now open, and the desktop will be displayed. On the left side of the screen, click on the LXTerminal icon as shown below.



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**.

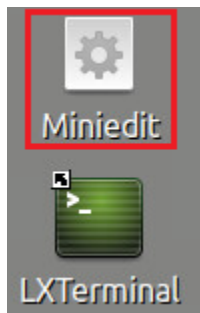
```
cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
```



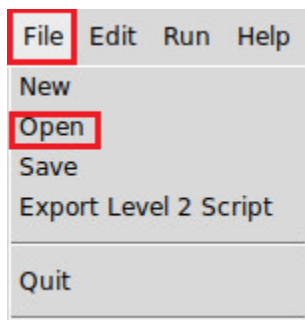
A new instance of Zeek is now active, and we are ready to proceed to the next section of the lab.

2.2 Launching Mininet

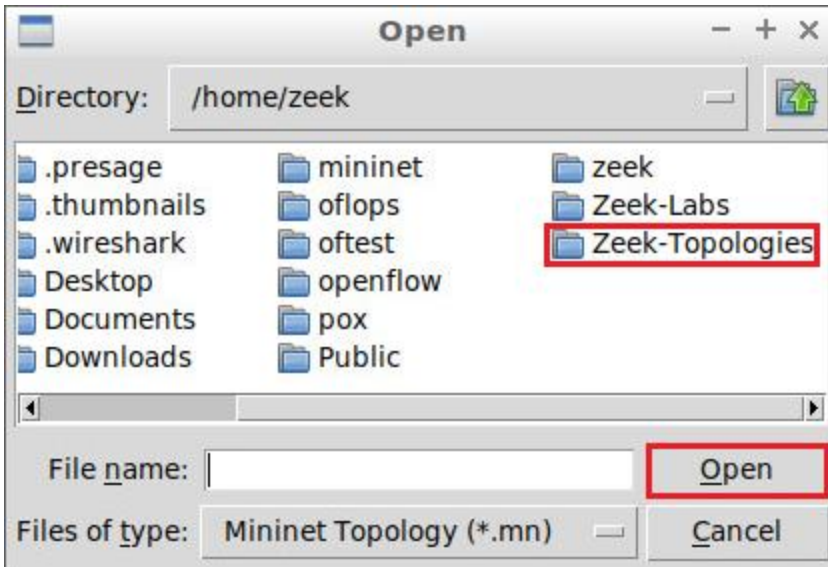
Step 1. From the *Client* machine’s desktop, on the left side of the screen, click on the MiniEdit icon as shown below. When prompted for a password, type **password** and hit **Enter**. The MiniEdit editor will now launch.



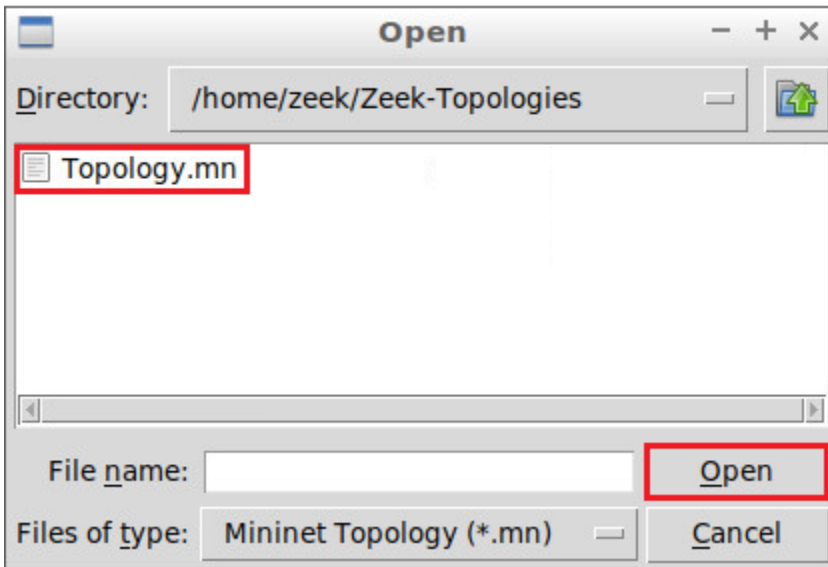
Step 2. The MiniEdit editor will now launch and allow for the creation of new, virtualized lab topologies. Load the correct topology by clicking the **Open** button within the **File** tab on the top left of the MiniEdit editor.



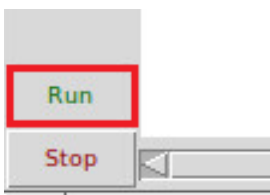
Step 3. Navigate to the Zeek-Topologies directory by scrolling to the right of the active directories and double clicking the Zeek-Topologies icon, or by clicking the **Open** button.



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

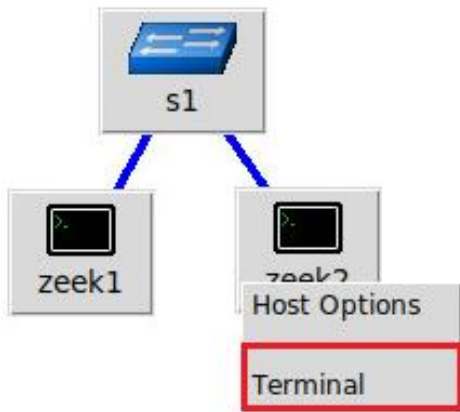


Step 5. To begin running the virtual machines, navigate to the Run button, found on the bottom left of the Miniedit editor, and select the Run button, as seen in the image below.



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. Navigate to the TCP-Traffic directory.

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

A terminal window titled '"Host: zeek2"' with standard window controls. The prompt is 'root@admin:~#'. The command 'cd Zeek-Labs/TCP-Traffic/' is entered and highlighted with a red box. The next line shows the prompt 'root@admin:~/Zeek-Labs/TCP-Traffic#' with a cursor.

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
```

A terminal window titled '"Host: zeek2"' with standard window controls. The prompt is 'root@admin:~/Zeek-Labs/TCP-Traffic#'. The command 'tcpdump -i zeek2-eth0 -s 0 -w scantraffic.pcap' is entered and highlighted with a red box. The output shows 'tcpdump: listening on zeek2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes' followed by a cursor.

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

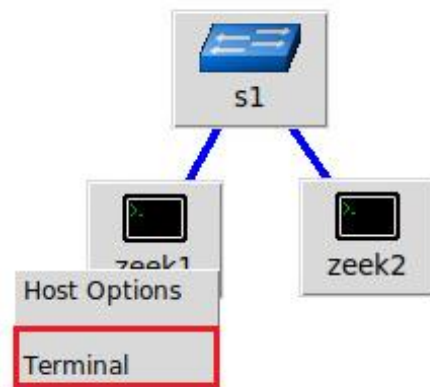
In this section we use the `nmap` software to generate TCP-based scan traffic.

This section introduces two new options for the `nmap` software.

- `-f`: specifies to send packet fragments. By fragmenting packets, a scanner can attempt to bypass firewalls that check for entire packet signatures.

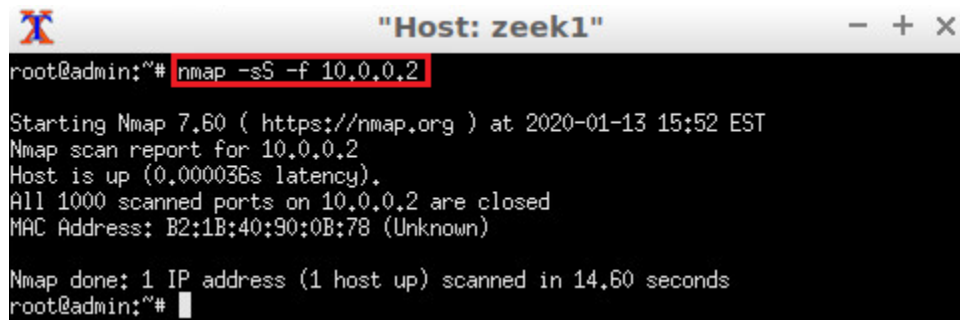
- `-mtu <num>`: specifies the max number of bytes to be sent in a fragmented packet. The number variable must be a multiple of 8.

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.



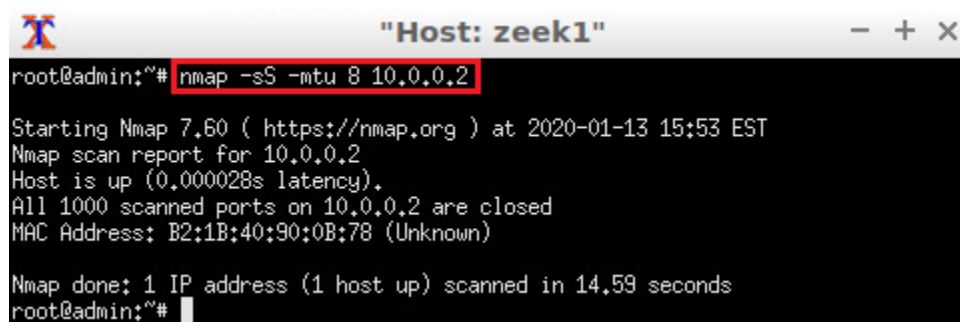
Step 2. Launch a fragmented TCP SYN scan against the *zeek2* machine.

```
nmap -sS -f 10.0.0.2
```



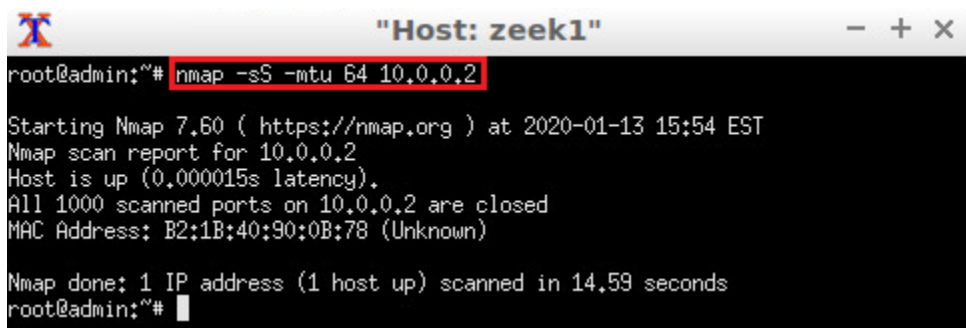
Step 3. Launch a fragmented TCP SYN scan with a packet size of 8 bytes against the *zeek2* machine.

```
nmap -sS -mtu 8 10.0.0.2
```



Step 5. Launch a fragmented TCP SYN scan with a packet size of 64 bytes against the *Bro2* machine.

```
nmap -sS -mtu 64 10.0.0.2
```

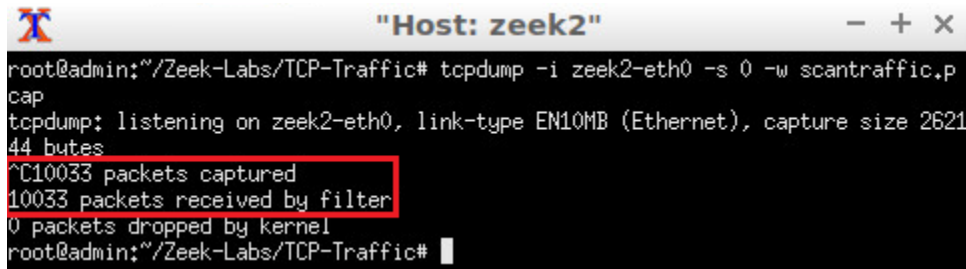


2.4.1 Terminating live network capture

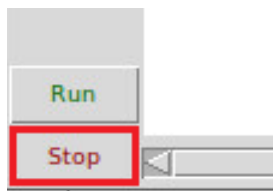
Step 1. Minimize the *zeek1* Terminal and open the *zeek2* Terminal using the navigation bar at the bottom of the screen. If necessary, right click within the Miniedit editor to activate your cursor.



Step 2. Use the `Ctrl+c` key combination to stop live traffic capture. Statistics of the capture session will be displayed. 10,033 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.

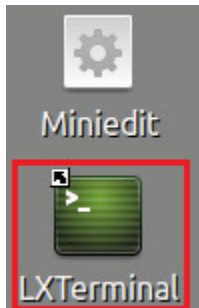


3 Generating and viewing Zeek profiling log files

Now that we have collected fragmented traffic, we can begin processing the packet capture file with Zeek.


3.1 Applying the profiling filter

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



Step 2. Navigate to the *Lab-Scripts* directory.

```
cd Zeek-Labs/Lab-Scripts/
```

A screenshot of a terminal window. The title bar reads "zeek@admin: ~/Zeek-Labs/Lab-Scripts". The terminal shows the command "cd Zeek-Labs/Lab-Scripts/" being entered and executed. The prompt changes from "zeek@admin:~\$" to "zeek@admin:~/Zeek-Labs/Lab-Scripts\$".

```
zeek@admin:~$ cd Zeek-Labs/Lab-Scripts/  
zeek@admin:~/Zeek-Labs/Lab-Scripts$
```

Step 3. View the *EnableProfiling.zeek* Zeek script.

```
nl EnableProfiling.zeek
```

```

zeek@admin: ~/Zeek-Labs/Lab-Scripts
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/Lab-Scripts$ nl EnableProfiling.zeek
 1 module Profiling;
 2 function log_suffix(): string
 3 {
 4     local rprof = getenv("ZEEK_LOG_SUFFIX");
 5     if ( rprof == "" )
 6         return "log";
 7     return rprof;
 8 }
 9 redef profiling_file = open(fmt("prof.%s", Profiling::log_suffix()));
10 redef profiling_interval = 15 secs;
11 redef expensive_profiling_multiple = 20;
12 event zeek_init()
13 {
14     set_buf(profiling_file, F);
15 }
zeek@admin:~/Zeek-Labs/Lab-Scripts$

```

Similar to the example in the introduction, the *EnableProfiling.zeek* Zeek script is used to create a new log file named *Statistics.log* containing Zeek profiling statistics. The script enables the intervals to be 15 seconds apart, with 20 total intervals.

Step 4. Navigate to the TCP-Traffic directory.

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

```

"Host: zeek2"
root@admin:~# cd Zeek-Labs/TCP-Traffic/
root@admin:~/Zeek-Labs/TCP-Traffic#

```

Step 5. Process the *ntraffic.pcap* packet capture file.

```
zeek -C -r ntraffic.pcap EnableProfiling.zeek
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek -C -r scantraffic.pcap ../Lab-Scripts
/EnableProfiling.zeek
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Step 6. Display the contents of the *Statistics.log* file.

```
nano prof.log
```



```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ nano prof.log
zeek@admin:~/Zeek-Labs/TCP-Traffic$
    
```

The *prof.log* file will be displayed.

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
GNU nano 2.9.3 prof.log
0.000000 -----
0.000000 Command line: zeek -C -r scantraffic.pcap ../Lab-Scripts/EnableProfi$
0.000000 -----
0.000000 Memory: total=89364K total_adj=0K malloced: 66980K
0.000000 Run-time: user+sys=0.0 user=0.0 sys=0.0 real=0.0
0.000000 Conns: total=0 current=0/0 ext=0 mem=0K avg=0.0 table=0K connvals=0K
0.000000 Conns: tcp=0/0 udp=0/0 icmp=0/0
0.000000 TCP-States: Inact. Syn. SA Part. Est. Fin. R$
0.000000 TCP-States:Inact. $
0.000000 TCP-States:Syn. $
0.000000 TCP-States:SA $
0.000000 TCP-States:Part. $
0.000000 TCP-States:Est. $
0.000000 TCP-States:Fin. $
0.000000 TCP-States:Rst. $
0.000000 Connections expired due to inactivity: 0
0.000000 Total reassembler data: 0K
0.000000 Timers: current=30 max=30 mem=1K lag=0.00s
    
```

Viewing the *Statistics.log* file, each profiling_interval will be displayed between a line separator made by dashes `---`.

Within the *Statistics.log* file, we can see the total memory used while processing the packet capture file, the Run-time, as well as a number of TCP flags, connections and *Triggers*. Within the first iteration of profiling_interval we see that no TCP packet flags have been recorded.

Step 6. Go to the next iteration of profiling_interval within the *Statistics.log* file.

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
GNU nano 2.9.3 prof.log
1578948752.164306 Memory: total=89364K total_adj=0K malleaged: 66984K
1578948752.164306 Run-time: user+sys=0.0 user=0.0 sys=0.0 real=0.0
1578948752.164306 Conns: total=0 current=0/0 ext=0 mem=0K avg=0.0 table=0K cos
1578948752.164306 Conns: tcp=0/0 udp=0/0 icmp=0/0
1578948752.164306 TCP-States: Inact. Syn. SA Part. Est. $
1578948752.164306 TCP-States:Inact. $
1578948752.164306 TCP-States:Syn. $
1578948752.164306 TCP-States:SA $
1578948752.164306 TCP-States:Part. $
1578948752.164306 TCP-States:Est. $
1578948752.164306 TCP-States:Fin. $
1578948752.164306 TCP-States:Rst. $
1578948752.164306 Connections expired due to inactivity: 0
1578948752.164306 Total reassembler data: 0K
1578948752.164306 Timers: current=27 max=30 mem=1K lag=1578948751.16s
1578948752.164306 DNS_Mgr: requests=0 succesful=0 failed=0 pending=0 cached_h$
1578948752.164306 Triggers: total=0 pending=0
1578948752.164306 ScheduleTimer = 2
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell

```

By scrolling through the *prof.log* file, we can see information found in the next iteration of a *profiling_interval*. We can see the total number of *TCP-States:Syn* has updated multiple parameters, with additional *Triggers* being included. This includes the total memory usage, displayed towards the bottom of the image.

Zeek profiling is a great tool for generating more detailed session-based statistics while processing packet capture files with Zeek.

4 Implementing tools to test Zeek's performance

While Zeek profiling will display the resulting statistics after processing a packet capture file, it is important to monitor Zeek resource consumption during network traffic analysis.

A number of Linux-based software utilities can be used to track system resource consumption in real time.

4.1 Using `sysstat` `sar` utility

The `sar` command can be used to display a number of system resources over specific time intervals. The following steps will highlight the ways to enable `sar` resource tracking.

Step 1. Launch the `sar` utility to track CPU consumption.

```
sar 2 30
```

- `sar`: calls the `sar` utility, belonging to the `sysstat` packages.

- `2`: indicates each iteration of CPU statistics is separated by a 2 second time interval.
- `30`: indicates that a total of 30 iterations of CPU statistics should be displayed.

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ sar 2 30
Linux 4.15.0-70-generic (admin)      01/13/2020      _x86_64_      (8 CPU)
04:08:37 PM   CPU   %user   %nice   %system   %iowait   %steal   %idle
e
04:08:39 PM   all    0.13    0.00    0.06    0.00    0.00    99.8
1
04:08:41 PM   all    0.00    0.00    0.00    0.00    0.00   100.0
0
04:08:43 PM   all    0.06    0.00    0.06    0.00    0.00    99.8
8
04:08:45 PM   all    0.06    0.00    0.00    0.00    0.00    99.9
4
04:08:47 PM   all    0.00    0.00    0.00    0.00    0.00   100.0
0
  
```

Use the `CTRL + C` keyboard combination to terminate the `sar` utility and return to the terminal.

Step 2. Launch the `sar` utility to track memory consumption.

```

sar -r 3 25
  
```

- `sar`: calls the `sar` utility, belonging to the `sysstat` packages.
- `-r`: indicates memory consumption in kilobytes.
- `3`: indicates each iteration of memory statistics is separated by a 3 second time interval.
- `25`: indicates that a total of 25 iterations of memory statistics should be displayed.

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ sar -r 3 25
Linux 4.15.0-70-generic (admin)      01/13/2020      _x86_64_      (8 CPU)
04:09:04 PM kbmemfree  kbavail  kbmemused  %memused  kbbuffers  kbcached  kbcomm
mit   %commit  kbactive   kbinact   kbdirty
04:09:07 PM 7266672  7496160   792116    9.83     92692     332116   1172
688    12.99   432692   121120    48
04:09:10 PM 7266672  7496160   792116    9.83     92692     332116   1172
688    12.99   432692   121120    48
04:09:13 PM 7266672  7496160   792116    9.83     92692     332116   1172
688    12.99   432692   121120    0
  
```

Use the `CTRL + C` keyboard combination to terminate the `sar` utility and return to the terminal.

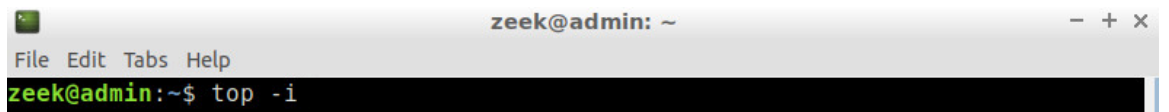
4.2 Using the top utility

Alternative to the `systat` `sar` utility, the `top` utility can be used to display the resource consumption of every active process.

Step 1. Launch the `top` utility to track resource consumption.

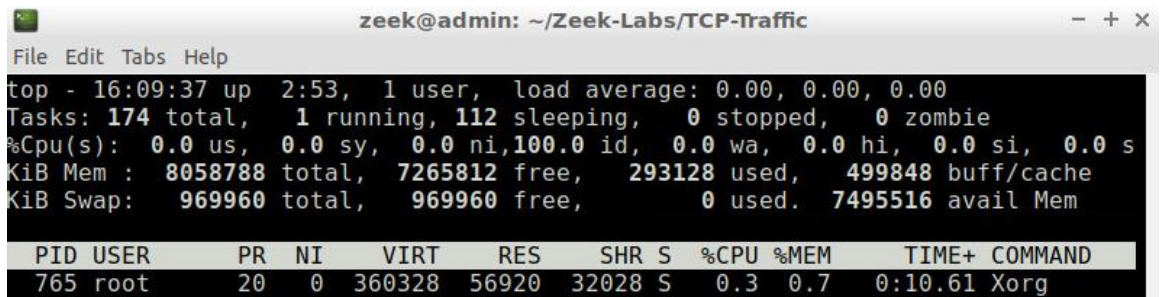
```
top -i
```

- `top`: calls the `top` utility.
- `-i`: toggles idle processes off, so that only active processes will be displayed.



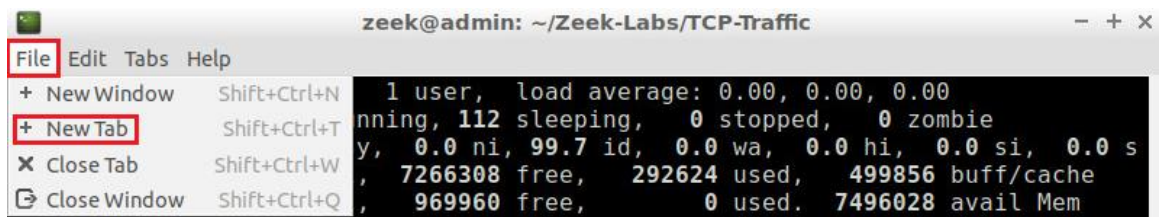
After entering the command, the Terminal will display the resource consumption.

Each row will belong to a unique process and display the related CPU and memory resource usage.



4.3 Viewing the resource consumption of Zeek

Step 1. Using the File drop down options, create a *New Tab* within the Terminal.



Step 2. In the second tab, begin packet capture file processing of the `bigFlows.pcap` file using Zeek.

```
zeek -C -r ../Sample-PCAP/bigFlows.pcap
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
* zeek@ad... X zeek@admi... X
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek -C -r ../Sample-PCAP/bigFlows.pcap
zeek@admin:~/Zeek-Labs/TCP-Traffic$
    
```

Step 3. Return to the first Terminal tab and view the active processes.

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admi... X zeek@admi... X
top - 16:11:25 up 2:54, 1 user, load average: 0.10, 0.03, 0.01
Tasks: 176 total, 2 running, 113 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.5 us, 0.5 sy, 0.0 ni, 97.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 s
KiB Mem : 8058788 total, 6811436 free, 386184 used, 861168 buff/cache
KiB Swap: 969960 total, 969960 free, 0 used. 7401608 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+  COMMAND
 2599 zeek      20   0 1402392 109280 18468  R   20.6   1.4   0:00.62  zeek
   765 root      20   0 360328   56920 32028  S    1.0   0.7   0:11.00  Xorg
 2382 zeek      20   0 547208   34304 26072  S    1.0   0.4   0:03.14  lxtermin+
    
```

Use the **CTRL + C** keyboard combination to terminate the **sar** utility and return to the terminal.

4.4 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
    
```

```

zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin: /usr/local/zeek/bin$
    
```

Concluding this lab, we introduced Zeek’s profiling capabilities and generated fragmented traffic to be processed into a profiling log file. Lastly, we introduced Terminal utilities that can be used to track Zeek’s resource consumption per process. Regular checking of Zeek profiling and resource consumption is necessary to ensure the IDS is working optimally in a real-time environment.

Furthermore, we have concluded introducing Zeek's capabilities as an IDS. The remaining labs within this series will focus on further processing Zeek log files for advanced analysis.

References

1. "Profiling", Zeek user manual, [Online], Available:
<https://docs.zeek.org/en/stable/scripts/policy/misc/profiling.zeek.html>



The University of Texas at San Antonio™
The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 10: Application of the Zeek IDS for Real-Time Network Protection

Document Version: **03-15-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to real-time network traffic analysis using Zeek	4
1.1 Starting a new instance of Zeek	4
1.2 Launching Mininet.....	5
1.3 Setting up the zeek1 virtual machine for live network capture	7
1.4 Using the zeek2 virtual machine for network scanning activities	8
1.4.1 Terminating live network capture	9
1.5 Analyzing the generated Zeek log files	9
2 Introduction to the Zeek NetControl framework	11
2.1 Viewing Zeek NetControl within a script file.....	11
2.2 Executing Zeek NetControl within a script file	15
3 Identifying SSH attacks by leveraging the Zeek NetControl framework	18
3.1 Closing the current instance of Zeek.....	21
References	21

Overview

This lab introduces Zeek’s real-time packet analysis for intrusion prevention. By combining the various Zeek-specific events that were introduced and reviewed in previous labs, we are able to identify and mitigate malicious traffic in real-time.

Objectives

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

1. Run Zeek in live mode to process network traffic *on the wire*.
2. Understand the Zeek NetControl framework.
3. Leverage advanced Zeek scripts for anomaly event detection.

Lab topology

Figure 1 shows the lab workspace topology. This lab primarily uses the *zeek2* virtual machine to generate scan-based traffic, and the *zeek1* virtual machine to perform live network capture.

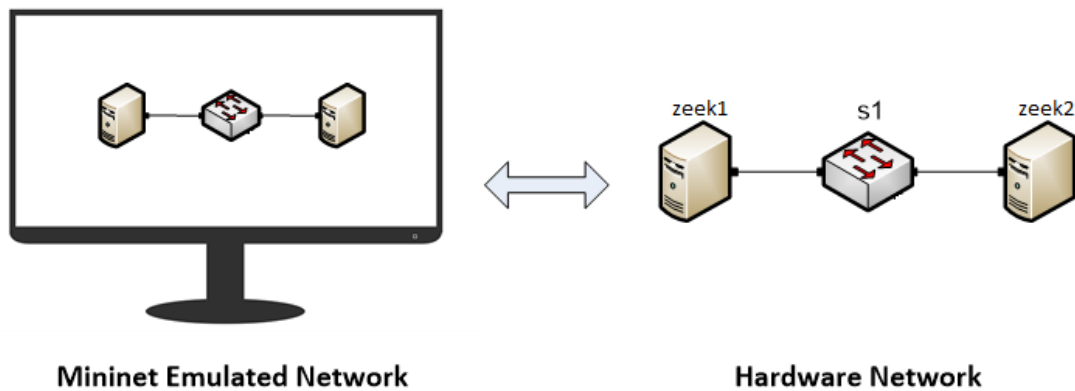


Figure 1. Lab topology.

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: Introduction to real-time network traffic analysis using Zeek.
2. Section 2: Introduction to the Zeek NetControl framework.
3. Section 3: Identifying SSH attacks by leveraging the Zeek NetControl framework.

1 Introduction to real-time network traffic analysis using Zeek

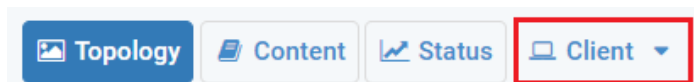
The previous labs within this lab series have leveraged the *tcpdump* terminal utility for capturing network traffic and generating packet capture files. However, Zeek is capable of collecting and analyzing such network traffic in real-time, with the ability to apply signature-matching and event-based Zeek scripts for malicious event detection.

This section will introduce leveraging Zeek for real-time network traffic analysis, without needing to save the packets captured by the receiving interface.

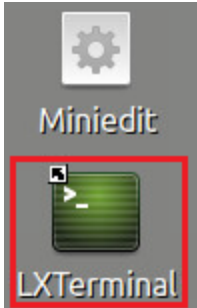
In contrast to the previous `-r` terminal option used for offline packet analysis, this lab will be using the `-i` terminal option to indicate the receiving interface for real-time network traffic analysis.

1.1 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.



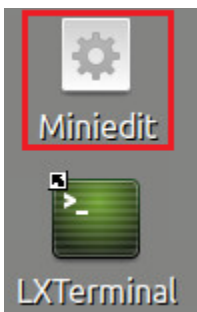
Step 3. Start Zeek by entering the following command on the terminal. This command enters Zeek’s default installation directory and invokes `zeekctl` tool to start a new instance. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key. When prompted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

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

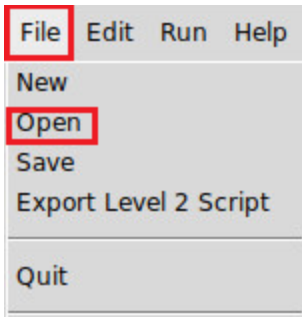
A new instance of Zeek is now active, and we are ready to proceed to the next section of the lab.

1.2 Launching Mininet

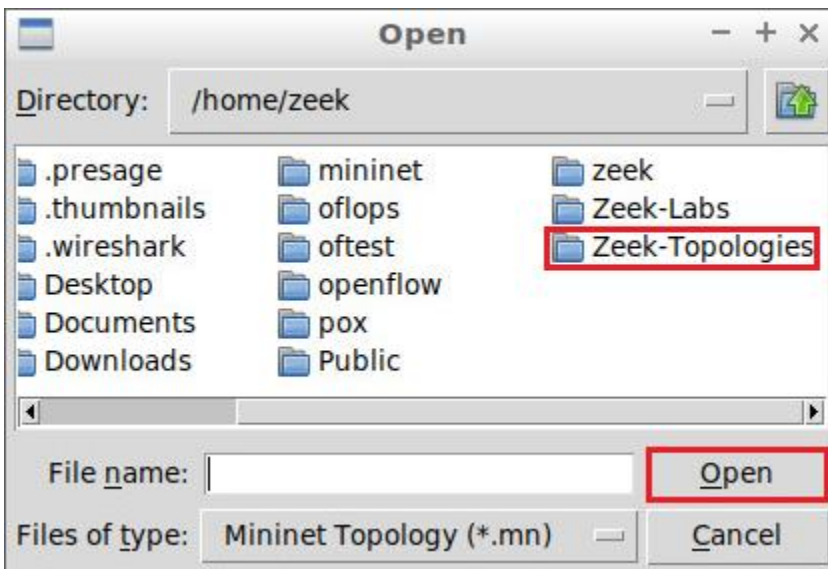
Step 1. From the *Client* machine’s desktop, on the left side of the screen, click on the MiniEdit icon as shown below. When prompted for a password, type `password` and hit `Enter`. The MiniEdit editor will now launch.



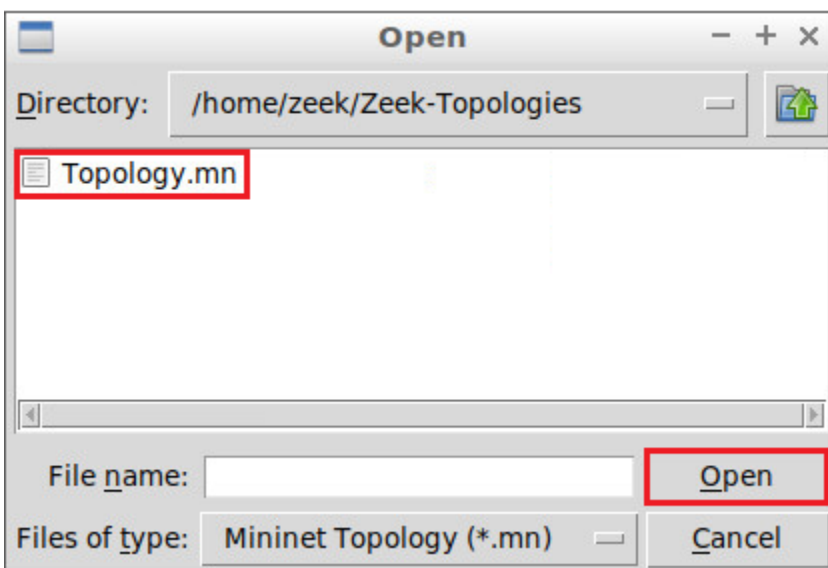
Step 2. The MiniEdit editor will now launch and allow for the creation of new, virtualized lab topologies. Load the correct topology by clicking the `Open` button within the `File` tab on the top left of the MiniEdit editor.



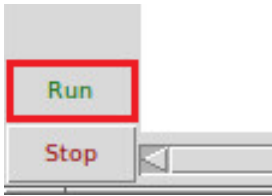
Step 3. Navigate to the Zeek-Topologies directory by scrolling to the right of the active directories and double clicking the Zeek-Topologies icon, or by clicking the **Open** button.



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

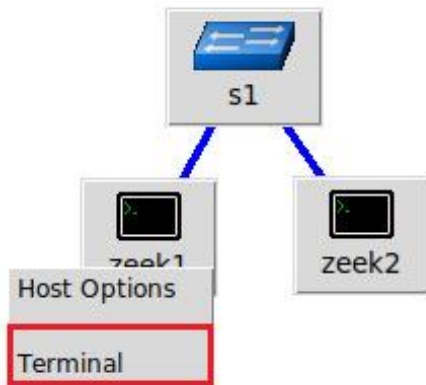


Step 5. To begin running the virtual machines, navigate to the `Run` button, found on the bottom left of the Miniedit editor, and select the `Run` button, as seen in the image below.



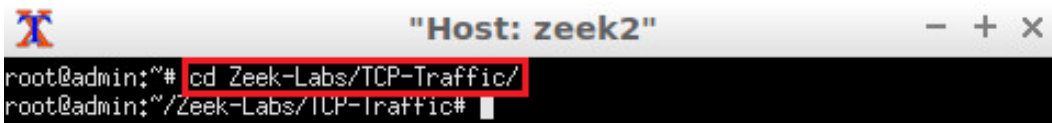
1.3 Setting up the zeek1 virtual machine for live network capture

Step 1. Launch the `zeek1` terminal by holding the right mouse button on the desired machine, and clicking the `Terminal` button.



Step 2. Navigate to the TCP-Traffic directory.

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



Step 3. Start an instance of Zeek live packet capture on interface `zeek1-eth0` while applying the advanced Zeek script `ZeekDetectScans.zeek`. It is possible to use the `tab` key to autocomplete the longer paths.

```
zeek -C -i zeek1-eth0 ../Lab-Scripts/ZeekDetectScans.zeek
```



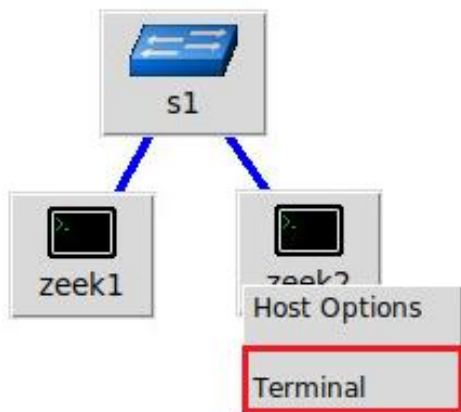
The *ZeekDetectScans.zeek* scripting file was introduced in Lab 8 of this lab series and will be used by the Zeek event-based engine to identify scan-based traffic. During live network traffic analysis, alternative scripts and signature files can be leveraged to identify specific anomalies and malicious attacks.

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

1.4 Using the zeek2 virtual machine for network scanning activities

In this section we use the `nmap` software to generate TCP-based scan traffic in order to trigger Zeek's logging notices.

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



Step 2. Launch a fragmented TCP scan against the *zeek1* machine.

```
nmap -sT 10.0.0.1
```

The screenshot shows a terminal window titled "Host: zeek2". The prompt is root@admin:~#. The command nmap -sT 10.0.0.1 has been entered and is highlighted with a red box. The output of the command is as follows:

```
Starting Nmap 7.60 ( https://nmap.org ) at 2020-03-16 16:08 EDT
Nmap scan report for 10.0.0.1
Host is up (0.00013s latency).
All 1000 scanned ports on 10.0.0.1 are closed
MAC Address: 6A:DB:89:6B:F2:55 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 13.35 seconds
root@admin:~#
```

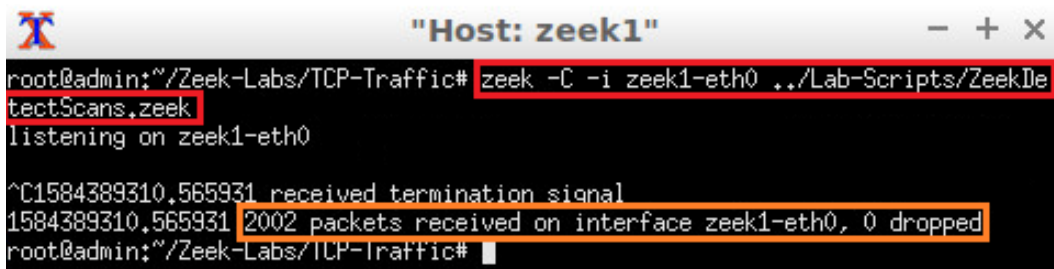
Now that we have generated scan-based traffic, we can verify that Zeek was able to identify such malicious events in real-time, while generating corresponding log files.

1.4.1 Terminating live network capture

Step 1. Minimize the *zeek2* Terminal and open the *zeek1* 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 be displayed. 2002 packets were recorded by the interface, which were continually analyzed by the Zeek event-based engine.



Within the previous image, the red box denotes the live capture command while the orange box indicates the number of packets received on the *zeek1-eth0* interface. 2002 packets were generated by the *zeek2* virtual machine, and no packets were dropped during analysis.

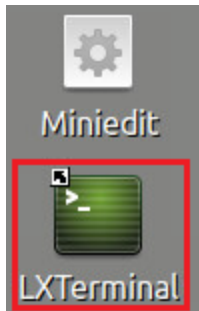
Step 3. Stop the current Mininet session by clicking the `Stop` button on the bottom left of the MiniEdit editor, and close the MiniEdit editor by clicking the `x` on the top right of the editor.



1.5 Analyzing the generated Zeek log files

To verify the success of our real time application of Zeek's event-based engine, we will return to the *Client* machine.

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



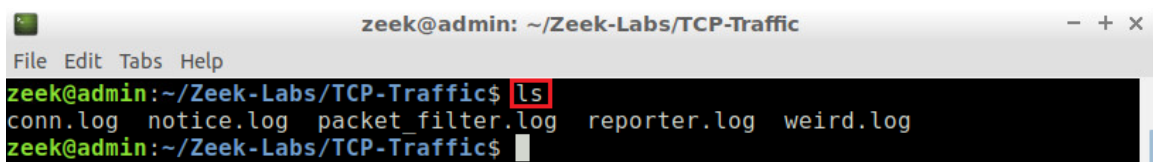
Step 2. Navigate to the *TCP-Traffic* directory to find the *scantraffic.pcap* file.

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



Step 3. View the file contents of the *TCP-Traffic* directory to ensure that Zeek generated log files based on the real-time network traffic analysis.

```
ls
```



A number of log files have been generated, specifically, the *notice.log* file which will contain the event's triggered by the *ZeekDetectScans.zeek* script file.

Step 3. View the file contents of the *notice.log* file to verify scan-based traffic was correctly identified and recorded by the Zeek event-based engine.

```
Head notice.log
```



```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ head notice.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path notice
#open 2020-03-16-16-08-28
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p
#rc dst p n peer_descr actions suppress_for remote_location.country_code remote_location.region remote_location.city remote_location.latitude remote_location.longitude
#types time string addr port addr port string string string
#num enum string string addr addr port count string set[enum]
#interval string string string double double
1584389308.915846 - - - - - -
- Scan::Port Scan 10.0.0.2 scanned at least 15 unique ports of host 10.0.0.1 in 0m0s remote 10.0.0.2 10.0.0.1 - - - N
notice::ACTION_LOG 3600.000000 - - - - -
#close 2020-03-16-16-08-30
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Within the previous image, the red box denotes the terminal command while the orange box indicates the resulting notice generated by Zeek due to the *ZeekDetectScans.zeek* script file. The *zeek2* virtual machine, with an IP address of 10.0.0.2, was recorded to have scanned at least 15 unique ports on the *zeek1* virtual machine.

Concluding this section, we have reviewed the capabilities of Zeek for conducting packet analysis during live network traffic capture. The signature and script files reviewed in previous labs can be leveraged during such real-time analysis, allowing for Zeek to monitor and protect a network in real-time.

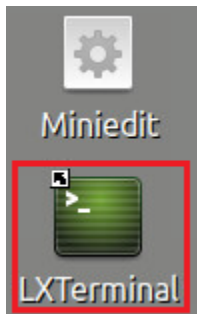
In the following section, we will review Zeek's NetControl framework, which is used to create a *backend* communication channel with application firewalls and related monitoring systems.

2 Introduction to the Zeek NetControl framework

The Zeek NetControl framework is used to create a flexible, unified interface for active mitigation and response against anomalous traffic. The framework allows for connectivity between a large number of devices, removing the heterogeneity of such configurations through creating a task-oriented API. This API is developed using the Zeek scripting language, consisting of a number of high-level calls and lower-level rule syntax. This section will introduce and review basic Zeek NetControl calls and their implementation for network traffic analysis in real-time.

2.1 Viewing Zeek NetControl within a script file

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



Step 2. Navigate to the *Lab-Scripts* directory.

```
cd Zeek-Labs/Lab-Scripts/
```

```
zeek@admin: ~/Zeek-Labs/Lab-Scripts
File Edit Tabs Help
zeek@admin:~$ cd Zeek-Labs/Lab-Scripts/
zeek@admin:~/Zeek-Labs/Lab-Scripts$
```

Step 3. View the contents of the *lab10_sec2-1.zeek* file using `nl`.

```
nl Zeek-Labs/Lab-Scripts/
```

```
zeek@admin: ~/Zeek-Labs/Lab-Scripts
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/Lab-Scripts$ nl lab10_sec2-1.zeek
1  event NetControl::init() {
2      local debug_plugin = NetControl::create_debug(T);
3      NetControl::activate(debug_plugin, 0);
4  }

5  event connection_established(c: connection) {
6
7      if ( |NetControl::find_rules_addr(c$id$orig_h)| > 0 ){
8          print "Error! Rule already exists!";
9          return;
10     }

11     NetControl::drop_connection(c$id, 20 secs);
12     print "Success! Rule created!";
13 }
zeek@admin:~/Zeek-Labs/Lab-Scripts$
```

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

1. Initializes the NetControl API framework.
2. Creates a local variable to contain debug information.
3. Uses the NetControl API to activate debugging and display notifications and/or error messages.
5. Zeek event in which a connection between a source and destination is formed. This can be initialized by the TCP handshake or a series of UDP Request and Reply packets.

7. Checks if a NetControl rule already exists based on the source address requesting a connection.
8. Prints a debug error message if the rule exists.
9. Exits the function and begins checking for the next connection within the packet stream.
11. If a rule has not been created, add a rule to drop any connections made by the current source address that lasts over 20 seconds.
12. Prints a debug message that a new rule was created.

This script is relatively basic and straightforward, yet shows the steps necessary to initialize the NetControl API. Without calling its initialization function, Zeek will be unable to communicate to various hardware devices through its backend.

Step 3. View the contents of the *ZeekDetectSSHAttacks.zeek* file using `nl`.

This script is very similar to the *ZeekDetectScans.zeek* default script reviewed in Lab 8 of this lab series. The following images will briefly review the file contents, while the Zeek documentation and previous lab provide a more in-depth analysis of this Zeek script. 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/master/scripts/policy/protocols/ssh/detect-bruteforcing.zeek.html
```

Command:

```
nl ZeekDetectSSHAttacks.zeek
```

```

zeek@admin: ~/Zeek-Labs/Lab-Scripts$ nl ZeekDetectSSHAttacks.zeek
1  ### Detect hosts which are doing password guessing attacks and/or password
2  ### bruteforcing over SSH.
3  @load base/protocols/ssh
4  @load base/frameworks/sumstats
5  @load base/frameworks/notice
6  @load base/frameworks/intel
7  module SSH;
8  export {
9      redef enum Notice::Type += {
10         ## Indicates that a host has been identified as crossing
the
11         ## :zeek:id:`SSH::password_guesses_limit` threshold with
12         ## failed logins.
13         Password_Guessing,
14         ## Indicates that a host previously identified as a "pas
sword
15         ## guesser" has now had a successful login
16         ## attempt. This is not currently implemented.
17         Login_By_Password_Guesser,

```

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

- 3-6. Zeek pre-directives to load SSH, summary and notice-specific script functionality.
- 7. Sets the module namespace to SSH.
- 8-17. Creates the export block to define the variables used throughout this script, specifically, the *Password_Guessing* variable on line 13 that will store the number of failed SSH password attempts.

```

18     };
19     redef enum Intel::Where += {
20         ## An indicator of the login for the intel framework.
21         SSH::SUCCESSFUL_LOGIN,
22     };
23     ## The number of failed SSH connections before a host is designa
ted as
24     ## guessing passwords.
25     const password_guesses_limit: double = 5 &redef;
26     ## The amount of time to remember presumed non-successful logins
to
27     ## build a model of a password guesser.
28     const guessing_timeout = 10 mins &redef;
29     ## This value can be used to exclude hosts or entire networks fr
om being
30     ## tracked as potential "guessers". The index represents
31     ## client subnets and the yield value represents server subnets.
32     const ignore_guessers: table[subnet] of subnet &redef;
33 }

```

Scroll down on the Terminal to view more of the script. Each number represents the respective line number:

- 25. Variable named *password_guesses_limit* that stores a numerical threshold for total number of failed SSH connections before marking a host as launching a brute-force attack.
- 28. Variable named *guessing_timeout* that stores a time-based threshold before resetting the *password_guesses_limit* variable back to 0.
- 29. Variable named *ignore_guessers* that stores a table of IP addresses identified to be launching SSH brute-force attacks. These addresses can be blocked or partially filtered.

```

zeek@admin: ~/Zeek-Labs/Lab-Scripts
File Edit Tabs Help
71     {
72     local id = c$id;

73     # Add data to the FAILED_LOGIN metric unless this connection sho
uld
74     # be ignored.
75     if ( ! (id$orig_h in ignore_guessers &&
76           id$resp_h in ignore_guessers[id$orig_h]) )
77         SumStats::observe("ssh.login.failure", [$host=id$orig_h]
, [$str=cat(id$resp_h)]);
78     }

79 event NetControl::init(){
80     local debug_plugin = NetControl::create_debug(T);
81     NetControl::activate(debug_plugin, 0);
82 }

83 hook Notice::policy(n: Notice::Info){
84     if ( n$note == SSH::Password_Guessing ){
85         NetControl::drop_address(n$src, 30min);
86     }
87 }

zeek@admin:~/Zeek-Labs/Lab-Scripts$

```

Scroll down on the Terminal to view more of the script. Each number represents the respective line number:

- 79. Initializes the NetControl API framework.
- 80. Creates a local variable to contain debug information.
- 81. Uses the NetControl API to activate debugging and display notifications and/or error messages.
- 83. Zeek hook to the Notice logging stream so that we can append new information with default information.
- 84. Checks the *Password_Guessing* variable to determine if the current source address has been identified to be launching SSH brute-force attacks.
- 85. If the current source address was launching SSH brute-force attacks, create a new rule that will drop all network traffic from this source for the next 30 minutes.

Now that we have reviewed both scripts that will be used within the remainder of the lab, we can see the value of Zeek's NetControl framework. By leveraging Zeek scripts we are able to identify anomalous network traffic events, detect malicious sources and finally leverage NetControl to mitigate their attacks. The remainder of this lab will include examples of executing the aforementioned Zeek scripts.

2.2 Executing Zeek NetControl within a script file

Step 1. Navigate to the *TCP-Traffic* directory.

```
cd ../TCP-Traffic/
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/Lab-Scripts$ cd ../TCP-Traffic/
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Step 2. Process the *smallFlows.pcap* packet capture file using the *lab10_sec2-1.zeek* script. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key. It is possible to use the `tab` key to autocomplete the longer paths.

```

zeek -C -r ../Sample-PCAP/smallFlows.pcap ../Lab-Scripts/lab10_sec2-1.zeek >
terminal.log

```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek -C -r ../Sample-PCAP/smallFlows.pcap ..
../Lab-Scripts/lab10_sec2-1.zeek > terminal.log
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Because we have NetControl debugging enabled, we are going to save all error messages and notifications to the file *terminal.log*. By saving these notifications to a separate file, it is easier to view them in an organized fashion.

Step 3. View the file contents of the *terminal.log* file using `head`.

```

head terminal.log

```

```

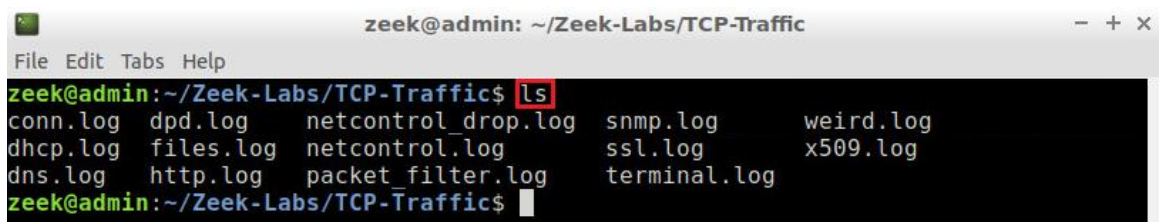
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ head terminal.log
netcontrol debug (Debug-All): init
netcontrol debug (Debug-All): add rule: [ty=NetControl::DROP, target=NetControl:
:FORWARD, entity=[ty=NetControl::CONNECTION, conn=[orig_h=192.168.3.131, orig_p=
55950/tcp, resp_h=72.14.213.102, resp_p=80/tcp], flow=<uninitialized>, ip=<unini
tialized>, mac=<uninitialized>], expire=20.0 secs, priority=0, location=, out_po
rt=<uninitialized>, mod=<uninitialized>, id=2, cid=2, _plugin_ids={\x0a\x0a}, _a
ctive plugin ids={\x0a\x0a}, _no_expire_plugins={\x0a\x0a}, _added=F]
Success! Rule created!
Error! Rule already exists!
Error! Rule already exists!
Error! Rule already exists!
Error! Rule already exists!
Error! Rule already exists!
Error! Rule already exists!
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Reviewing this image, the red box indicates the terminal command used to view the file. The orange box indicates the NetControl debug message that it has been initialized. The blue box indicates that a new rule has been created to drop all packets from the specified IP address for passing the connection-length threshold. The dark blue box indicates the Zeek event message that the rule was created successfully, while the yellow box indicates the Zeek event message that the rule already existed and a duplicate was not created.

Step 4. View the contents of the *TCP-Traffic* directory using `ls`.

```
ls
```

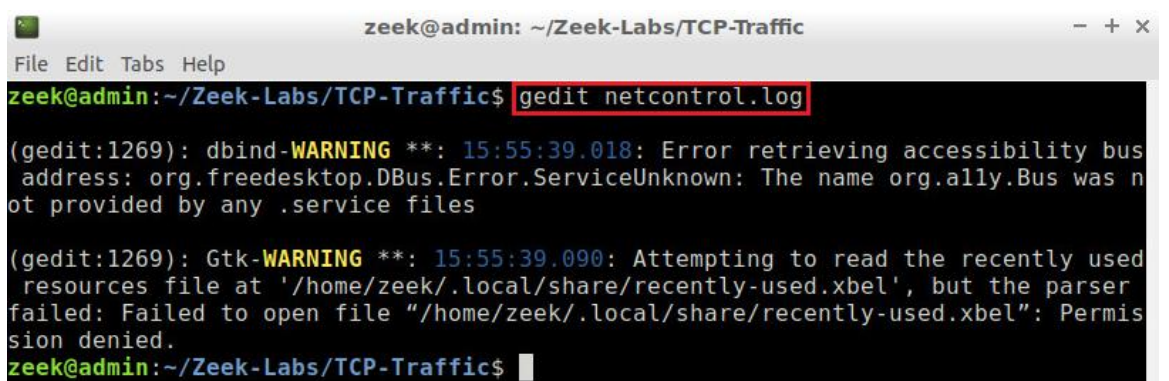


```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ls
conn.log dpd.log netcontrol_drop.log snmp.log weird.log
dhcp.log files.log netcontrol_log ssl.log x509.log
dns.log http.log packet_filter.log terminal.log
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

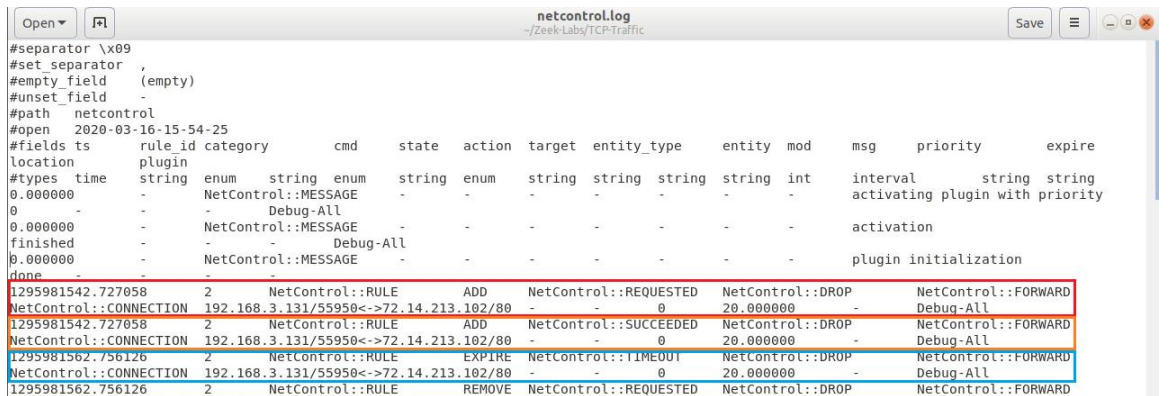
The two log files we are interested in are the *netcontrol.log* and *netcontrol_drop.log* files. The *netcontrol.log* file will contain all information related to adding and removing rules, while the *netcontrol_drop.log* file will contain information regarding to when each rule was triggered and by which source address.

Step 5. View the file contents of the *netcontrol.log* file using `gedit`.

```
gedit netcontrol.log
```



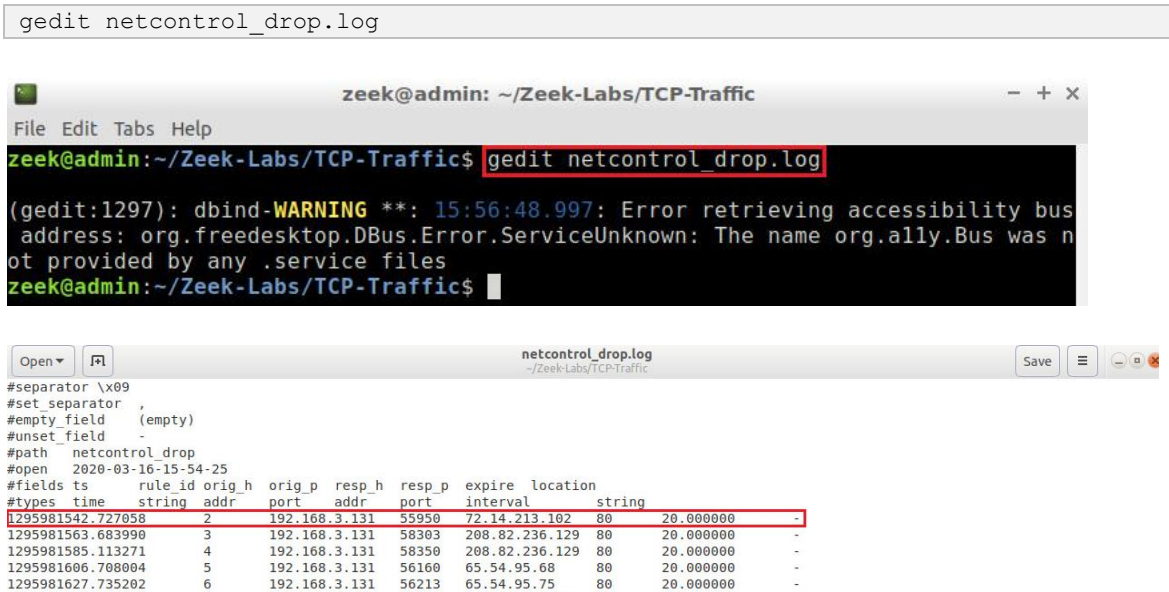
```
zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ gedit netcontrol.log
(gedit:1269): dbind-WARNING **: 15:55:39.018: Error retrieving accessibility bus
address: org.freedesktop.DBus.Error.ServiceUnknown: The name org.ally.Bus was n
ot provided by any .service files
(gedit:1269): Gtk-WARNING **: 15:55:39.090: Attempting to read the recently used
resources file at '/home/zeek/.local/share/recently-used.xbel', but the parser
failed: Failed to open file "/home/zeek/.local/share/recently-used.xbel": Perm
ission denied.
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```



#open	ts	rule_id	category	cmd	state	action	target	entity_type	entity	mod	msg	priority	expire
0.000000	-	-	NetControl::MESSAGE	-	-	-	-	-	-	-	activating plugin with priority	-	-
0.000000	-	-	Debug-All	-	-	-	-	-	-	-	activation	-	-
0.000000	-	-	NetControl::MESSAGE	-	-	-	-	-	-	-	plugin initialization	-	-
0.000000	-	-	Debug-All	-	-	-	-	-	-	-	-	-	-
1295981542.727058	2	NetControl::RULE	ADD	NetControl::REQUESTED	NetControl::DROP	NetControl::FORWARD	192.168.3.131/55950<->72.14.213.102/80	-	0	20.000000	-	Debug-All	-
1295981542.727058	2	NetControl::RULE	ADD	NetControl::SUCCEEDED	NetControl::DROP	NetControl::FORWARD	192.168.3.131/55950<->72.14.213.102/80	-	0	20.000000	-	Debug-All	-
1295981562.756126	2	NetControl::RULE	EXPIRE	NetControl::TIMEOUT	NetControl::DROP	NetControl::FORWARD	192.168.3.131/55950<->72.14.213.102/80	-	0	20.000000	-	Debug-All	-
1295981562.756126	2	NetControl::RULE	REMOVE	NetControl::REQUESTED	NetControl::DROP	NetControl::FORWARD	192.168.3.131/55950<->72.14.213.102/80	-	0	20.000000	-	Debug-All	-

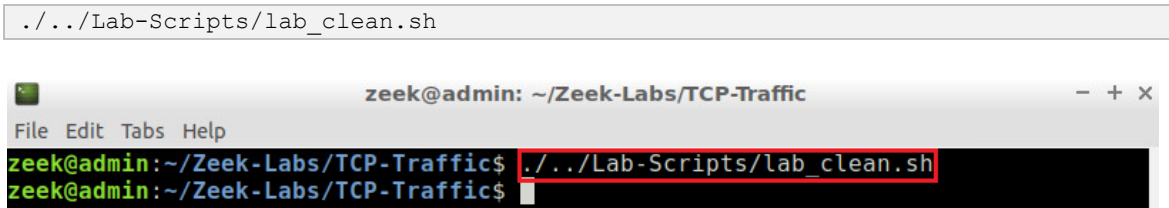
Reviewing this image, the red box indicates that a Connection request was made by the source address *192.168.3.131* to the destination address *72.14.213.102*. The orange box indicates that the connection was established while the blue box indicates that the connection was dropped and forced to time-out because of NetControl filtering packets from this source destination.

Step 6. View the file contents of the `netcontrol_drop.log` file using `gedit`.



Reviewing this image, the red box indicates that a source address attempted to create a connection, breaking the NetControl rule we had previously implemented. Therefore, all packets were dropped from this source host during the time-out interval we declared within the `lab10_sec2-1.zeek` script.

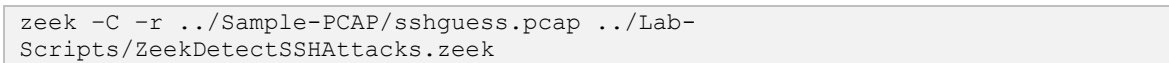
Step 7. Clear the `TCP-Traffic` directory by using the `lab_clean.sh` shell script.



3 Identifying SSH attacks by leveraging the Zeek NetControl framework

Now that we have reviewed a basic implementation of the NetControl framework, creating a connection-based rule and identifying source addresses that broke the rule, we will conduct a more in-depth analysis on SSH brute-force password attacks.

Step 1. Process the `sshguess.pcap` packet capture file using `ZeekDetectSSHAttacks.zeek`. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key. It is possible to use the `tab` key to autocomplete the longer paths.



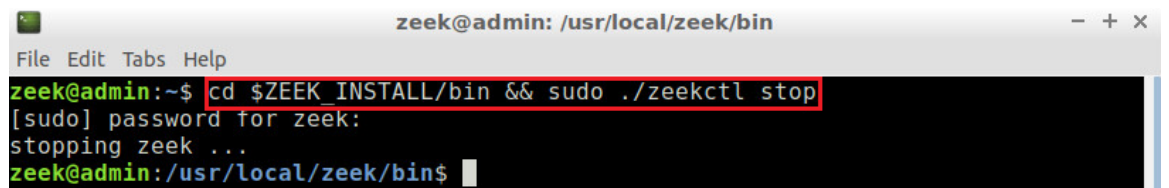
Recall that the *notice.log* file is generated by the *ZeekDetectSSHAttacks.zeek* script. The red box indicates which IP address was logged to have broken the SSH password guessing threshold.

3.1 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
```



```
zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin: /usr/local/zeek/bin$
```

Concluding this lab, we have introduced the Zeek NetControl framework and Zeek’s live processing of real-time network traffic. While the NetControl examples were performed on offline packet capture files, by combining Zeek’s live analysis from Section 1 with the examples from Section 2 and 3, active measures can be taken for identifying malicious network traffic and blocking such sources.

References

1. “NetControl Framework”, Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/frameworks/netcontrol.html>
2. “Logging framework”, Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/frameworks/logging.html>
3. “Writing scripts”, Zeek user manual, [Online], Available: <https://docs.zeek.org/en/stable/examples/scripting/#the-event-queue-and-event-handlers>
4. “Quick start Guide”, Zeek user manual, [Online], Available: <https://docs.zeek.org/en/current/quickstart>



The University of Texas at San Antonio™
The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 11: Preprocessing of Zeek Output Logs for Machine Learning

Document Version: **03-13-2020**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objective	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to machine learning in network security.....	4
1.1 ARFF file format.....	5
2 Aggregating network capture datasets	6
2.1 Starting a new instance of Zeek	6
2.2 Launching Mininet.....	7
2.3 Setting up the zeek2 virtual machine for live network capture	8
2.4 Using the zeek1 virtual machine for network scanning activities	9
2.4.1 Terminating live network capture	10
3 Preprocessing of Zeek log files.....	11
3.1 Preprocessing the malicious dataset	11
3.2 Preprocessing of the benign dataset	15
3.3 Creation of the test and training datasets	16
3.4 Adding the .arff file headers	18
3.5 Closing the current instance of Zeek.....	19
References	20

Overview

This lab introduces the application of machine learning in the network security field. After using Zeek's scripting language to generate anomaly-based output files, it is necessary to format these datasets to be used by machine learning classifiers.

Objective

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

1. Explain the benefits of leveraging machine learning for network analysis.
2. Understand Attribute-Relation File Format (ARFF).
3. Aggregate and preprocess a dataset to be used by a machine learning classifier.

Lab topology

Figure 1 shows the lab workspace topology. This lab primarily uses the *Bro2* machine for offline Zeek log file processing and reformatting.

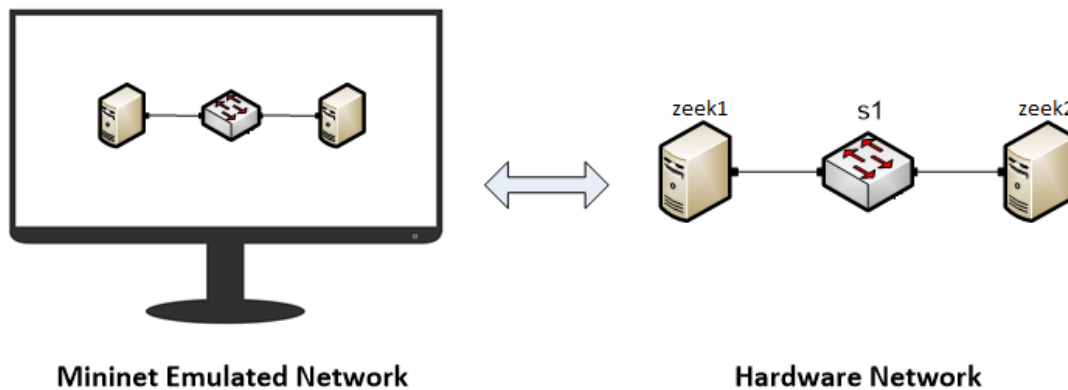


Figure 1. Lab topology.

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: Introduction to machine learning in network security.
2. Section 2: Aggregating network capture datasets.
3. Section 3: Preprocessing of Zeek log files.

1 Introduction to machine learning in network security

Machine learning is programming computers to optimize a performance criterion using example data or past experience¹. Machine learning is particularly useful for computing empirical correlations, and in cases where it is difficult to write a computer program to solve a given problem. In recent years, technological advances in machine learning have propelled its application on various domains and sectors. Cyber-security is a critical area in which machine learning (ML) is increasingly becoming significant.

By using Zeek and text processing languages, it is possible to identify the presence of an anomaly. Once an anomaly is detected, Zeek's scripts can be implemented to extract relevant fields and build a dataset.

In this lab series, we will train machine learning classifiers using these anomaly-based datasets in order to build a model that can be used for future predictions.

This lab focuses on reformatting Zeek log files into Attribute-Relation File Format (ARFF) files, to be used by Weka software. Weka is a workbench for machine learning that is intended to help in the application of machine learning techniques to a variety of real-world problems².

Supervised learning is a common approach used in machine learning. Supervised learning consists of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). When training a machine learning classifier using supervised learning, it is important to include both a training and test dataset:

- **Training dataset**: dataset used by the classifier to "learn" correlations and feature weights. Data should include instances of both variable and control group, while containing a classification label.

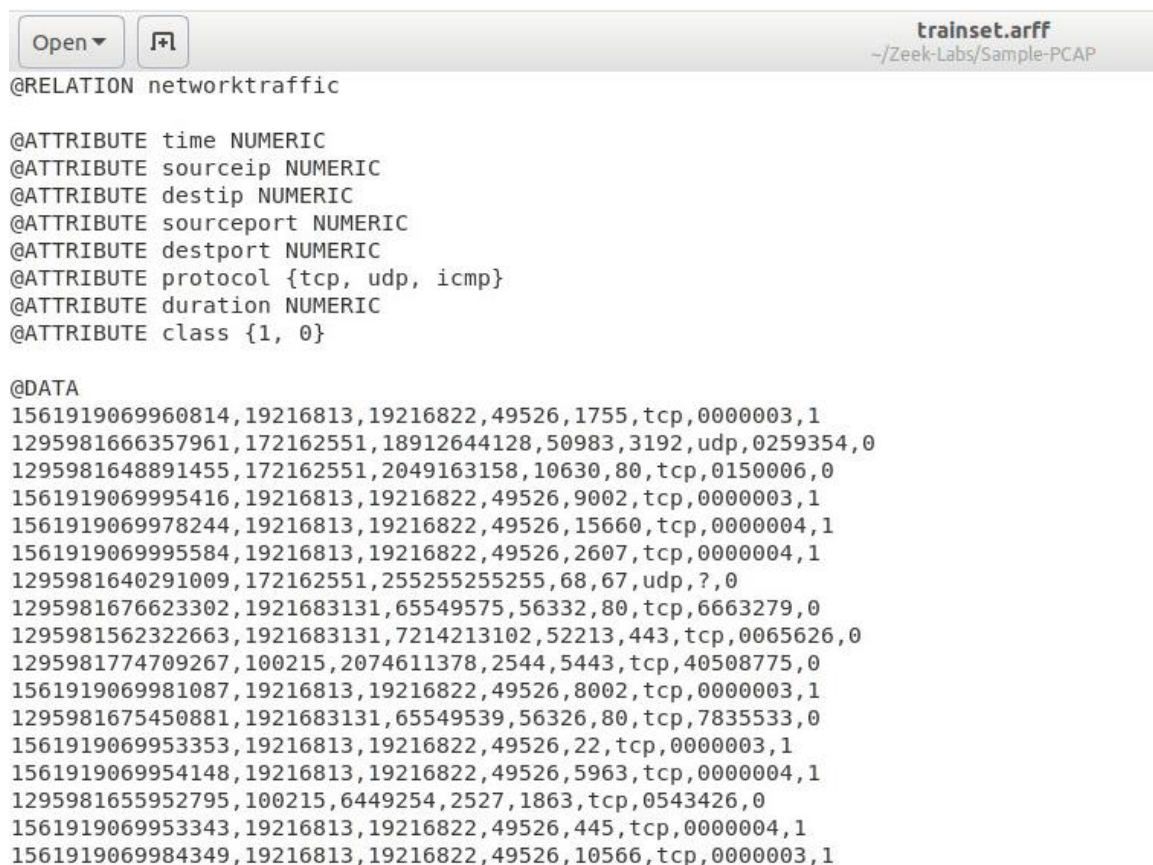
- **Testing dataset**: dataset used by the classifier to test accuracy. If the classifier is able to accurately predict labels for the training dataset but not for the testing dataset, then it is necessary to adjust and retrain the classifier.

1.1 ARFF file format

The Weka software contains a variety of different machine learning algorithms to train a number of classifiers. Each classifier will require different datasets; for instance, decision trees can only handle numeric or nominal values, and strings cannot be used as an input without being listed nominally.

The majority of machine learning classifiers accept numeric data inputs. Therefore, we will need to preprocess our log file datasets to contain only numeric and nominal data. Additionally, Weka requires each input dataset to be formatted in an *.arff* file format.

ARFF files contain comma-separated values and additional headers and labels. Below is a sample of a properly formatted *.arff* file that we will be developing in this lab.



```

trainset.arff
~/Zeek-Labs/Sample-PCAP

@RELATION networktraffic

@ATTRIBUTE time NUMERIC
@ATTRIBUTE sourceip NUMERIC
@ATTRIBUTE destip NUMERIC
@ATTRIBUTE sourceport NUMERIC
@ATTRIBUTE destport NUMERIC
@ATTRIBUTE protocol {tcp, udp, icmp}
@ATTRIBUTE duration NUMERIC
@ATTRIBUTE class {1, 0}

@DATA
1561919069960814,19216813,19216822,49526,1755,tcp,0000003,1
1295981666357961,172162551,18912644128,50983,3192,udp,0259354,0
1295981648891455,172162551,2049163158,10630,80,tcp,0150006,0
1561919069995416,19216813,19216822,49526,9002,tcp,0000003,1
1561919069978244,19216813,19216822,49526,15660,tcp,0000004,1
1561919069995584,19216813,19216822,49526,2607,tcp,0000004,1
1295981640291009,172162551,255255255255,68,67,udp,?,0
1295981676623302,1921683131,65549575,56332,80,tcp,6663279,0
1295981562322663,1921683131,7214213102,52213,443,tcp,0065626,0
1295981774709267,100215,2074611378,2544,5443,tcp,40508775,0
1561919069981087,19216813,19216822,49526,8002,tcp,0000003,1
1295981675450881,1921683131,65549539,56326,80,tcp,7835533,0
1561919069953353,19216813,19216822,49526,22,tcp,0000003,1
1561919069954148,19216813,19216822,49526,5963,tcp,0000004,1
1295981655952795,100215,6449254,2527,1863,tcp,0543426,0
1561919069953343,19216813,19216822,49526,445,tcp,0000004,1
1561919069984349,19216813,19216822,49526,10566,tcp,0000003,1

```

The ARFF file headers can be summarized as follows:

- **@RELATION**: name of the dataset.
- **@ATTRIBUTE**: specifies the label and the data type for each column:

- `NUMERIC`: integer data type.
- `NOMINAL`: values match entries defined within the brackets `{}`.
- `@DATA`: lists the input data.

Now that we have introduced ARFF files and understand what an input dataset should look like, we can start aggregating and preprocessing a dataset using Zeek.

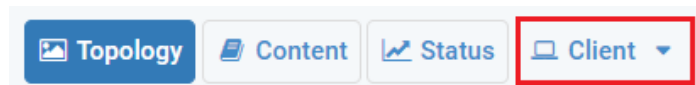
2 Aggregating network capture datasets

To create our dataset, we need to make sure there is a certain level of entropy in the data to guarantee that the machine learning classifier will learn properly. Therefore, we need to combine both benign and malicious datasets.

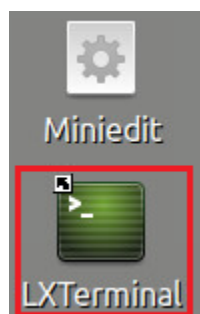
In this lab, we use the *smallFlows.pcap* file as the control group, identified as benign traffic with a class label of 0. We then generate a new *scantraffic.pcap* file to be used as the variable group, identified as malicious traffic with a class label of 1.

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.

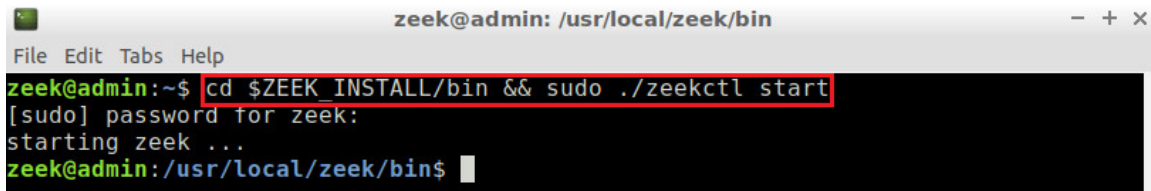


Step 2. The *Client* machine will now open, and the desktop will be displayed. On the left side of the screen, click on the LXTerminal icon as shown below.



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. When prompted for a password, type `password` and hit `Enter`.

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

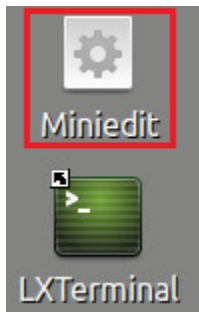


```
zeek@admin: /usr/local/zeek/bin
File Edit Tabs Help
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
[sudo] password for zeek:
starting zeek ...
zeek@admin:~/usr/local/zeek/bin$
```

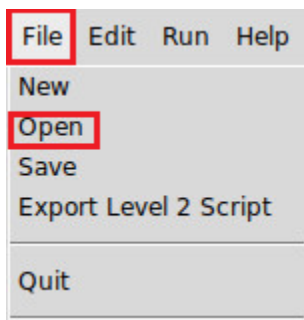
A new instance of Zeek is now active, and we are ready to proceed to the next section of the lab.

2.2 Launching Mininet

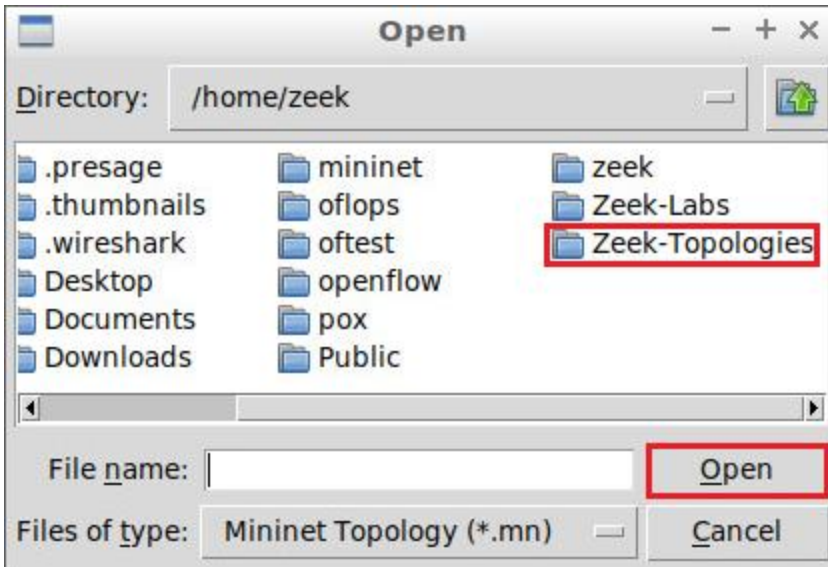
Step 1. From the *Client* machine's desktop, on the left side of the screen, click on the MiniEdit icon as shown below. When prompted for a password, type `password` and hit `Enter`. The MiniEdit editor will now launch.



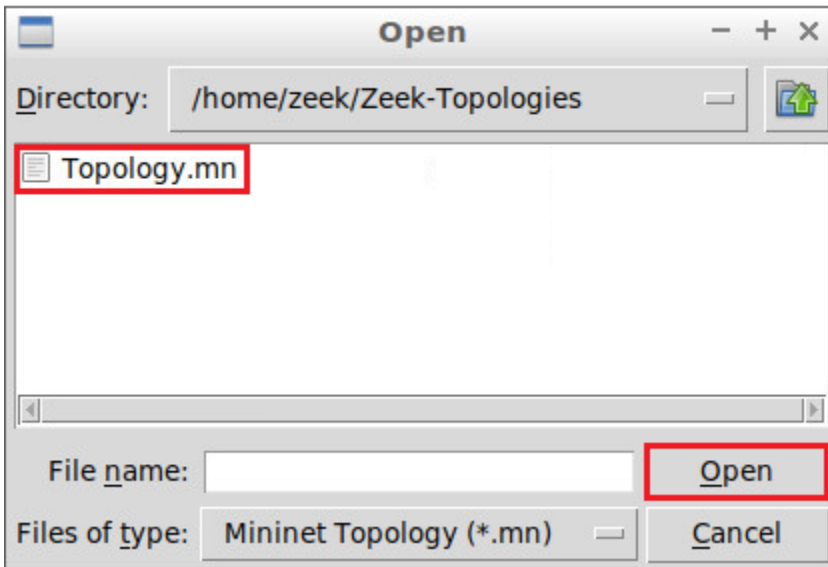
Step 2. The MiniEdit editor will now launch and allow for the creation of new, virtualized lab topologies. Load the correct topology by clicking the `Open` button within the `File` tab on the top left of the MiniEdit editor.



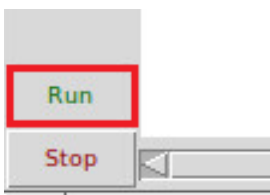
Step 3. Navigate to the Zeek-Topologies directory by scrolling to the right of the active directories and double clicking the Zeek-Topologies icon, or by clicking the `Open` button.



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

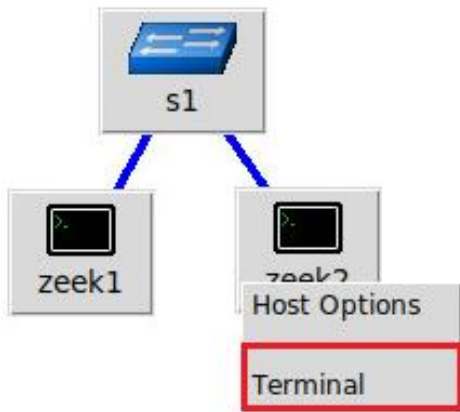


Step 5. To begin running the virtual machines, navigate to the Run button, found on the bottom left of the Miniedit editor, and select the Run button, as seen in the image below.



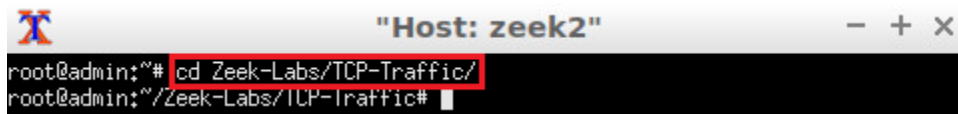
2.3 Setting up the zeek2 virtual machine for live network capture

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



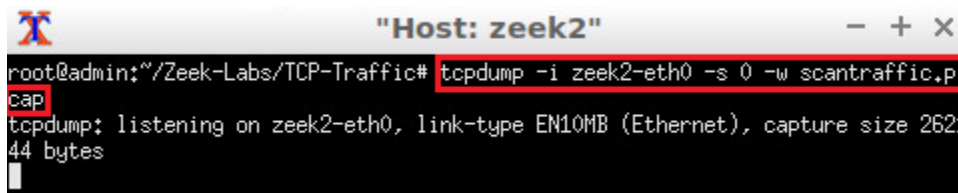
Step 2. Navigate to the TCP-Traffic directory.

```
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
```

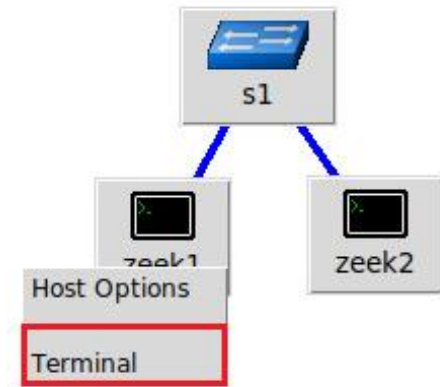


The *zeek2* virtual machine is now ready to begin collecting live network traffic. Next, we will use the *zeek1* machine to generate scan-based network traffic.

2.4 Using the *zeek1* virtual machine for network scanning activities

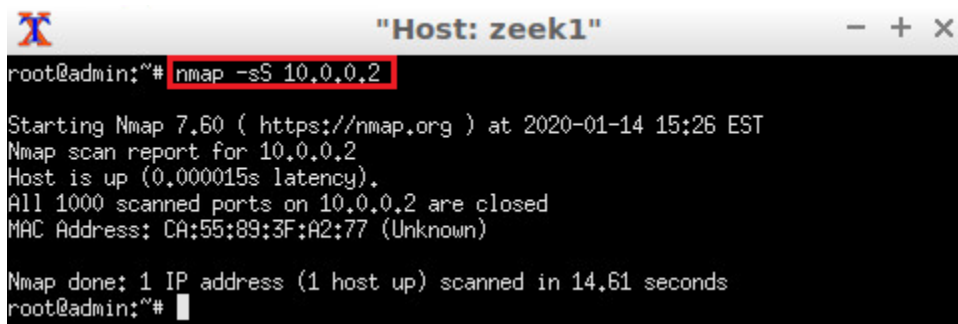
In this section we use the `nmap` software to generate TCP-based scan traffic.

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 SYN scan against the *zeek2* machine.

```
nmap -sS 10.0.0.2
```

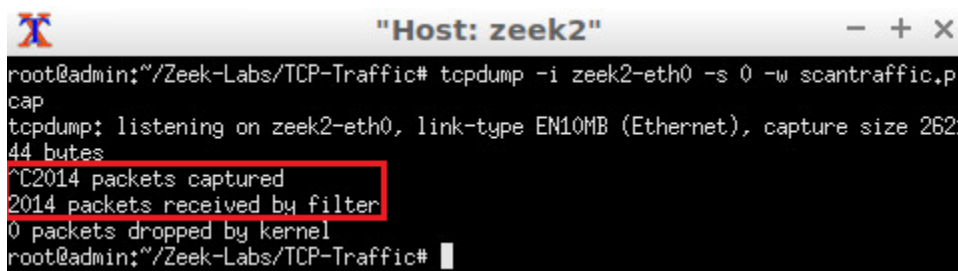


2.4.1 Terminating live network capture

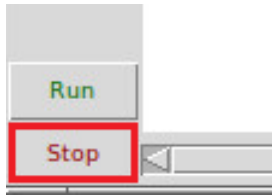
Step 1. Minimize the *zeek1* Terminal and open the *zeek2* Terminal using the navigation bar at the bottom of the screen. If necessary, right click within the Miniedit editor to activate your cursor.



Step 2. Use the `Ctrl+c` key combination to stop live traffic capture. Statistics of the capture session will be displayed. 2,014 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.



We now have our malicious dataset, and because the *smallFlows.pcap* file is already downloaded, we already have our control group, the benign dataset. In the following section we will begin formatting our datasets into ARFF files.

3 Preprocessing of Zeek log files

To generate ARFF files, we first need to process our packet capture files using Zeek's default configuration.

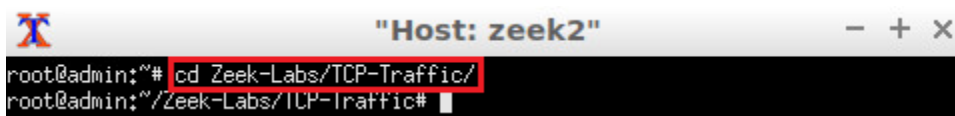
In a real-time environment, at this stage you may include anomaly-specific scripts. Once an anomaly has been processed by Zeek, the resulting log files will need to be reformatted.

Afterwards, we need to select which features we wish to extract from the Zeek log files to be used in our training and testing datasets. It is important to carefully select the relevant features when training a classifier. If features are not strategically selected, classifiers may create unreliable correlations which may lead to poor accuracy in the detection process. In this lab we extract a small number of general packet features.

3.1 Preprocessing the malicious dataset

Step 3. Navigate to the TCP-Traffic directory.

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



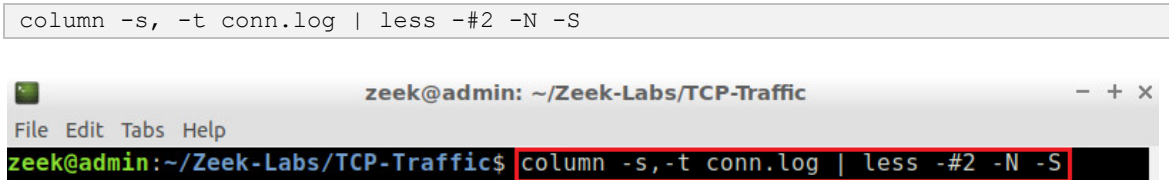
Step 1. Process the *scantraffic.pcap* file.

```
zeek -C -r scantraffic.pcap
```



Step 2. Display the contents of the *conn.log* file.

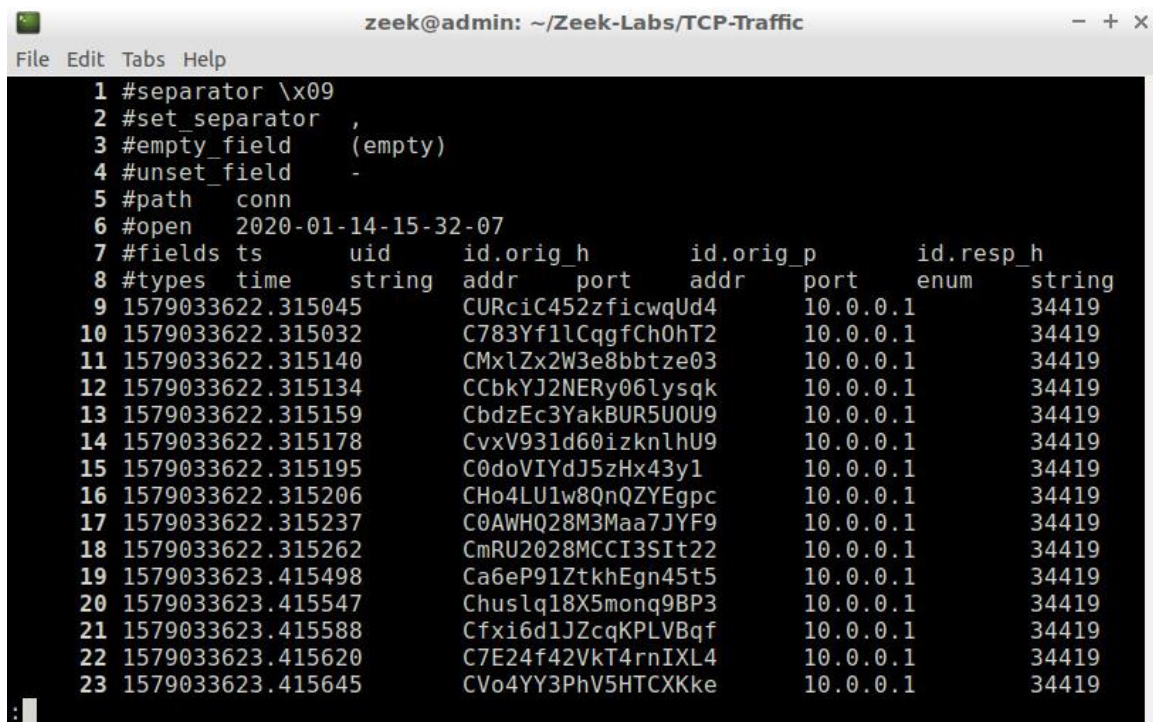
```
column -s, -t conn.log | less -#2 -N -S
```



Examining the previous command:

- `column -s, -t conn.log`: calls the `column` utility to read and columnize the file contents of the *conn.log* file. The `-s` option specifies the separator and the `-t` option enables the output to be created as a table.
- `| less -#2 -N -S`: accepts the output of the `column` utility and calls the `less` utility. The `-#2` specifies the default number of positions to scroll horizontally in the RIGHTARROW and LEFTARROW keys, the `-N` option marks each row with a line number and the `-S` option causes the display to remove any data that would not fit on the current Terminal screen rather than overflowing to a new line.

The previous command results in the following output.



```

1 #separator \x09
2 #set_separator ,
3 #empty_field (empty)
4 #unset_field -
5 #path conn
6 #open 2020-01-14-15-32-07
7 #fields ts uid id.orig_h id.orig_p id.resp_h
8 #types time string addr port addr port enum string
9 1579033622.315045 CURciC452zficwqUd4 10.0.0.1 34419
10 1579033622.315032 C783Yf1lCqgfCh0hT2 10.0.0.1 34419
11 1579033622.315140 CMxlZx2W3e8bbtze03 10.0.0.1 34419
12 1579033622.315134 CCbkYJ2NERy06lysqk 10.0.0.1 34419
13 1579033622.315159 CbdzEc3YakBUR5U0U9 10.0.0.1 34419
14 1579033622.315178 CvXV931d60izknlhU9 10.0.0.1 34419
15 1579033622.315195 C0doVIYdJ5zHx43y1 10.0.0.1 34419
16 1579033622.315206 CHo4LU1w8QnQZYegpc 10.0.0.1 34419
17 1579033622.315237 C0AWHQ28M3Maa7JYF9 10.0.0.1 34419
18 1579033622.315262 CmRU2028MCCI3SIt22 10.0.0.1 34419
19 1579033623.415498 Ca6eP91ZtkhEgn45t5 10.0.0.1 34419
20 1579033623.415547 Chuslq18X5monq9BP3 10.0.0.1 34419
21 1579033623.415588 Cfxi6d1JZcqKPLVBqf 10.0.0.1 34419
22 1579033623.415620 C7E24f42Vkt4rnIXL4 10.0.0.1 34419
23 1579033623.415645 CVo4YY3PhV5HTCXKke 10.0.0.1 34419

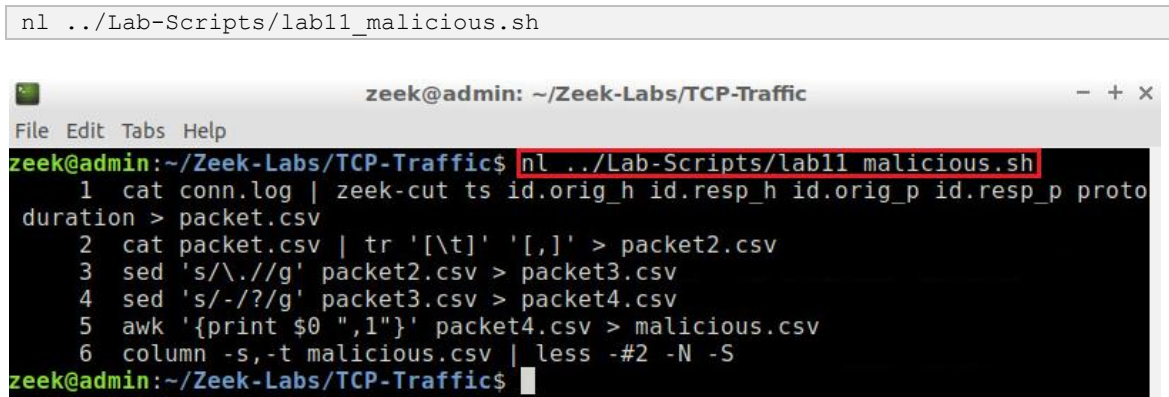
```

We can see in the previous image that the *conn.log* file is nowhere near the *.arff* file format. We will need to remove the Zeek padding, column names, change the tab delimiter and remove excess column features.

Press the `q` key on your keyboard to exit and return to the Terminal.

Step 3: Display the contents of `lab11_malicious.sh` shell script using the `nl` command.

```
nl ../Lab-Scripts/lab11_malicious.sh
```



```
zeek@admin:~/Zeek-Labs/TCP-Traffic$ nl ../Lab-Scripts/lab11_malicious.sh
1 cat conn.log | zeek-cut ts id.orig_h id.resp_h id.orig_p id.resp_p proto
duration > packet.csv
2 cat packet.csv | tr '\t' ',' > packet2.csv
3 sed 's/\./g' packet2.csv > packet3.csv
4 sed 's/-/?g' packet3.csv > packet4.csv
5 awk '{print $0 "1"}' packet4.csv > malicious.csv
6 column -s,-t malicious.csv | less -#2 -N -S
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```


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

- Using the `cat` utility, the contents of the `conn.log` file will be passed into the `zeek-cut` utility to remove the log file header and only include the specified columns. The output of the `zeek-cut` utility will be saved to a new file named `packet.csv`. The feature columns we will be using to train our example machine learning classifier are:
 - `ts`: time the packet was received.
 - `id.orig_h`: source IP address.
 - `id.resp_h`: destination IP address.
 - `id.orig_p`: source port.
 - `id.resp_h`: destination port.
 - `proto`: transport protocol.
 - `duration`: connection or session length.
- Using the `cat` utility, the contents of the `packet.csv` file will be passed into the `tr` utility. The `tr` utility will replace the `packet.csv` file's tab-delimited structure with a comma-delimited structure, and the output will be saved to a new file named `packet2.csv`.
- Using the `sed` utility, all instances of a period `.` will be removed. This will allow for the IP addresses to be input as a numeric data type rather than a string, and the output will be saved to a new file named `packet3.csv`.
- Using the `sed` utility, all instances of a dash `-` will be replaced by `?`. Currently, when a column is empty, Zeek writes a dash `-`. However, Weka reads question marks `?` as an empty column. The output will be saved to a new file named `packet4.csv`.
- Using the `awk` utility, every row will have an additional `1` appended to the end of the row. This will represent the class label; we used 1 to denote the malicious traffic. The output will be saved to a new file named `malicious.csv`.

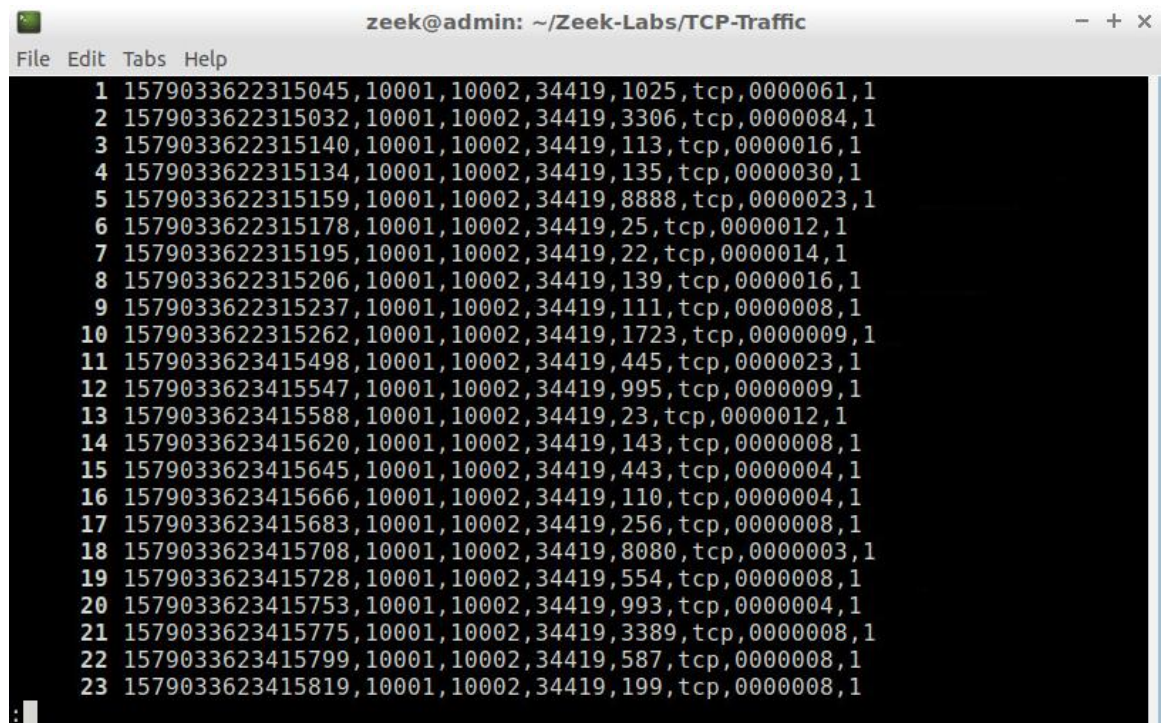
- The file contents of *malicious.csv* will be displayed. This command is introduced in the Step 1 of this subsection.

Step 4: Execute the *lab11_malicious.sh* shell script. If prompted for a password, type `password` and hit *Enter*.

```
./../Lab-Scripts/lab11_malicious.sh
```



After executing all commands in the script, the *malicious.csv* file contents will be displayed on the Terminal as shown in the figure below.



```
1 1579033622315045,10001,10002,34419,1025,tcp,0000061,1
2 1579033622315032,10001,10002,34419,3306,tcp,0000084,1
3 1579033622315140,10001,10002,34419,113,tcp,0000016,1
4 1579033622315134,10001,10002,34419,135,tcp,0000030,1
5 1579033622315159,10001,10002,34419,8888,tcp,0000023,1
6 1579033622315178,10001,10002,34419,25,tcp,0000012,1
7 1579033622315195,10001,10002,34419,22,tcp,0000014,1
8 1579033622315206,10001,10002,34419,139,tcp,0000016,1
9 1579033622315237,10001,10002,34419,111,tcp,0000008,1
10 1579033622315262,10001,10002,34419,1723,tcp,0000009,1
11 1579033623415498,10001,10002,34419,445,tcp,0000023,1
12 1579033623415547,10001,10002,34419,995,tcp,0000009,1
13 1579033623415588,10001,10002,34419,23,tcp,0000012,1
14 1579033623415620,10001,10002,34419,143,tcp,0000008,1
15 1579033623415645,10001,10002,34419,443,tcp,0000004,1
16 1579033623415666,10001,10002,34419,110,tcp,0000004,1
17 1579033623415683,10001,10002,34419,256,tcp,0000008,1
18 1579033623415708,10001,10002,34419,8080,tcp,0000003,1
19 1579033623415728,10001,10002,34419,554,tcp,0000008,1
20 1579033623415753,10001,10002,34419,993,tcp,0000004,1
21 1579033623415775,10001,10002,34419,3389,tcp,0000008,1
22 1579033623415799,10001,10002,34419,587,tcp,0000008,1
23 1579033623415819,10001,10002,34419,199,tcp,0000008,1
```

We can see from the above image that the *malicious.csv* file is now properly formatted to fit in the `@DATA` section of an ARFF file. Each row contains an equal number of comma-delimited columns with only numeric characters.

Press the `q` key on your keyboard to exit and return to the Terminal.

Now that we have our malicious dataset created, we can begin formatting our benign dataset.

Step 5: Execute the *lab_clean.sh* shell script to clear the directory. If required, type `password` as the password.

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

```

zeek@admin: ~/Zeek-Labs/UDP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/UDP-Traffic$ ./../Lab-Scripts/lab_clean.sh
[sudo] password for zeek:
zeek@admin:~/Zeek-Labs/UDP-Traffic$

```

3.2 Preprocessing of the benign dataset

Step 1: Process the *smallFlows.pcap* file using the `zeek -r` command.

```
zeek -C -r ../Sample-PCAP/smallFlows.pcap
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ zeek -C -r ../Sample-PCAP/smallFlows.pcap
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

Step 2: Display the contents *lab11_benign.sh* shell script using the `nl` command.

```
nl ../Lab-Scripts/lab11_benign.sh
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ nl ../Lab-Scripts/lab11_benign.sh
 1 cat conn.log | zeek-cut ts id.orig_h id.resp_h id.orig_p id.resp_p proto
duration > packet.csv
 2 cat packet.csv | tr '\t' ',' > packet2.csv
 3 sed 's/\.//g' packet2.csv > packet3.csv
 4 sed 's/-/?/g' packet3.csv > packet4.csv
 5 awk '{print $0 ",0"}' packet4.csv > benign.csv
 6 column -s,-t benign.csv | less -#2 -N -S
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

With the exception of Line 5, the script is exactly the same as the one explained in Step 3 of the previous section.

Line 5 has been modified to append `,0` to the end of each row. This value represents the benign class label. The output will be saved to a new file named *benign.csv*.

Step 3: Execute the *lab11_benign.sh* shell script.

```
./../Lab-Scripts/lab1_benign.sh
```

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ./../Lab-Scripts/lab11_benign.sh

```

After executing all commands in the script, the *benign.csv* file contents will be displayed on the Terminal as shown in the figure below.

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
1 1295981542708292,1921683131,7214213102,55950,80,tcp,0058485,0
2 1295981543461968,1921683131,2074614838,55955,80,tcp,0028620,0
3 1295981543337241,1921683131,65551737,55954,80,tcp,1776718,0
4 1295981546609581,1921683131,20882236129,58264,80,tcp,0125448,0
5 1295981546736952,1921683131,20882236129,58265,80,tcp,0169843,0
6 1295981549760088,1921683131,7214213105,57721,443,tcp,0001363,0
7 1295981549832444,1921683131,20882236129,58272,80,tcp,0166319,0
8 1295981543841133,1921683131,655495140,55963,80,tcp,9427326,0
9 1295981545127681,1921683131,655495142,55973,80,tcp,8140921,0
10 1295981543652521,1921683131,206108207139,55960,80,tcp,17762163,0
11 1295981561406421,1921683131,742175010,57038,80,tcp,0081750,0
12 1295981562220872,1921683131,7214213102,52201,443,tcp,0067879,0
13 1295981562221263,1921683131,7214213102,52203,443,tcp,0068419,0
14 1295981562221386,1921683131,7214213102,52204,443,tcp,0070702,0
15 1295981562223069,1921683131,7214213102,52205,443,tcp,0070857,0
16 1295981562223270,1921683131,7214213102,52206,443,tcp,0071444,0
17 1295981562269795,1921683131,7214213102,52207,443,tcp,0068342,0
18 1295981562271216,1921683131,7214213102,52209,443,tcp,0073342,0
19 1295981562270019,1921683131,7214213102,52208,443,tcp,0074541,0
20 1295981562271658,1921683131,7214213102,52211,443,tcp,0077819,0
21 1295981562271438,1921683131,7214213102,52210,443,tcp,0078040,0
22 1295981562317554,1921683131,7214213102,52212,443,tcp,0066224,0
23 1295981562322663,1921683131,7214213102,52213,443,tcp,0065626,0

```

We can see from the above image that the *benign.csv* file is now properly formatted to fit in the `@DATA` section of an ARFF file. Each row contains an equal number of comma-delimited columns with only numeric characters.

Press the `q` key on your keyboard to exit and return to the Terminal.

Now that we have our both of our datasets created, we are ready to combine them into the training and test input datasets.

3.3 Creation of the test and training datasets

Step 1: Combine the *malicious.csv* and *benign.csv* files into the *dataset.csv* file.

```

zeek@admin: ~/Zeek-Labs/TCP-Traffic
File Edit Tabs Help
zeek@admin:~/Zeek-Labs/TCP-Traffic$ cat malicious.csv benign.csv > dataset.csv
zeek@admin:~/Zeek-Labs/TCP-Traffic$

```

The *dataset.csv* file will now contain the *benign.csv* rows appended to the end of the *malicious.csv* rows. We now need to randomize the file contents and apply further formatting by executing the *lab11_create_sets.sh* shell script.

Step 2: Display the contents of *lab11_create_sets.sh* shell script using the `nl` command.

```
nl ../Lab-Scripts/lab11_create_sets.sh
```

```
zeek@admin:~/Zeek-Labs/TCP-Traffic$ nl ../Lab-Scripts/lab11_create_sets.sh
1 shuf dataset.csv > randomized.csv
2 head -n 300 randomized.csv > test.csv
3 sed -e '1,300d' randomized.csv > trainset.arff
4 sed 's/.$/?/' test.csv > testset.arff
5 wc -l testset.arff
6 wc -l trainset.arff
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

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

1. Using the `shuf` utility, the contents of the `dataset.csv` file will be shuffled, and the output will be saved to a new file named `randomized.csv`.
2. Using the `head` utility, the top 300 rows from the `randomized.csv` file were saved to a new file named `test.csv`.
3. Using the `sed` utility, rows 1-300 are removed from the `randomized.csv` file and the output is saved to the new `trainset.arff` file.
4. Using the `sed` utility, the last column of the `test.csv` file is removed. We are removing the label of each instance of the test dataset so that we can have the classifier attempt to predict these labels. The output is saved to the new `testset.arff` file.
5. Using the `wc` utility, the number of rows within the `testset.arff` file are displayed. We can compare this value against the value found in Line 8 to make sure no packet data was lost.
6. Using the `wc` utility, the number of rows within the `trainset.arff` file are displayed. We can compare this value against the value found in Line 7 to make sure no packet data was lost.

Step 3: Execute the `lab11_create_sets.sh` shell script.

```
../Lab-Scripts/lab11_create_sets.sh
```

```
zeek@admin:~/Zeek-Labs/TCP-Traffic$ ../Lab-Scripts/lab11_create_sets.sh
300 testset.arff
1400 trainset.arff
zeek@admin:~/Zeek-Labs/TCP-Traffic$
```

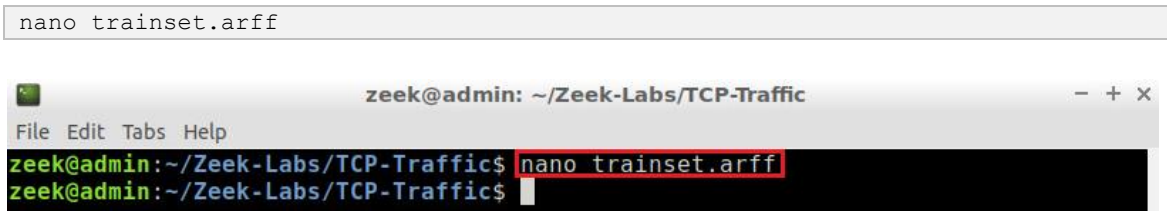
The figure above shows the line count of the `testset.arff` and `trainset.arff` files. The `testset.arff` file contains 300 rows while the `trainset.arff` file contains 1400 rows. The `trainset.arff` file size may be variable due to the number of packets generated during the original TCP SYN scans; however, the `testset.arff` file should always be equal to 300 rows due to the executed script.

Now that we have generated our testing and training *.arff* files, the final step for preprocessing the Zeek datasets is to add the *.arff* file headers to each file.

3.4 Adding the *.arff* file headers

Step 1: Using the `nano` text editor, open the *trainset.arff* file for editing.

```
nano trainset.arff
```

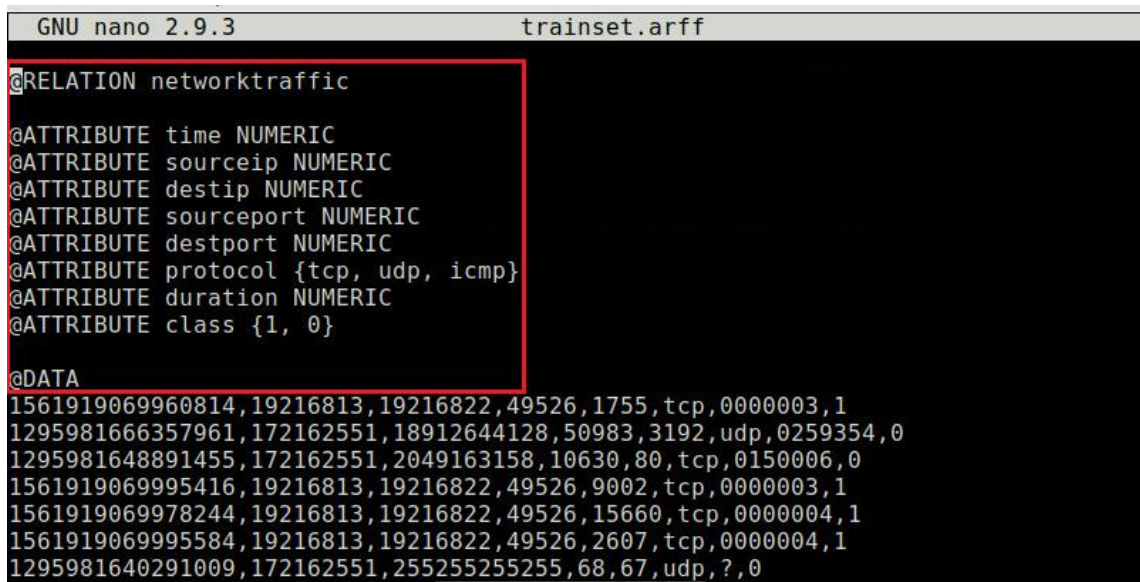


Step 2: Prepend the following headers to the *trainset.arff* file. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key.

```
@RELATION networktraffic

@ATTRIBUTE time NUMERIC
@ATTRIBUTE sourceip NUMERIC
@ATTRIBUTE destip NUMERIC
@ATTRIBUTE sourceport NUMERIC
@ATTRIBUTE destport NUMERIC
@ATTRIBUTE protocol {tcp, udp, icmp}
@ATTRIBUTE duration NUMERIC
@ATTRIBUTE class {1,0}

@DATA
```



```
GNU nano 2.9.3 trainset.arff
@RELATION networktraffic
@ATTRIBUTE time NUMERIC
@ATTRIBUTE sourceip NUMERIC
@ATTRIBUTE destip NUMERIC
@ATTRIBUTE sourceport NUMERIC
@ATTRIBUTE destport NUMERIC
@ATTRIBUTE protocol {tcp, udp, icmp}
@ATTRIBUTE duration NUMERIC
@ATTRIBUTE class {1, 0}
@DATA
1561919069960814,19216813,19216822,49526,1755,tcp,0000003,1
1295981666357961,172162551,18912644128,50983,3192,udp,0259354,0
1295981648891455,172162551,2049163158,10630,80,tcp,0150006,0
1561919069995416,19216813,19216822,49526,9002,tcp,0000003,1
1561919069978244,19216813,19216822,49526,15660,tcp,0000004,1
1561919069995584,19216813,19216822,49526,2607,tcp,0000004,1
1295981640291009,172162551,255255255255,68,67,udp,?,0
```

The input training dataset is now a properly formatted *.arff* file and can be input into a machine learning algorithm to train a classifier.

Step 3: Using the `nano` text editor, open the *testset.arff* file for editing.

```
nano testset.arff
```

Step 2: Prepend the following headers to the *testset.arff* file. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key.

The headers are the same as those added to the *trainset.arff* file, so they can be copied and pasted directly into the *testset.arff* file.

```
@RELATION networktraffic

@ATTRIBUTE time NUMERIC
@ATTRIBUTE sourceip NUMERIC
@ATTRIBUTE destip NUMERIC
@ATTRIBUTE sourceport NUMERIC
@ATTRIBUTE destport NUMERIC
@ATTRIBUTE protocol {tcp, udp, icmp}
@ATTRIBUTE duration NUMERIC
@ATTRIBUTE class {1,0}

@DATA
```

```
GNU nano 2.9.3 testset.arff
@RELATION networktraffic
@ATTRIBUTE time NUMERIC
@ATTRIBUTE sourceip NUMERIC
@ATTRIBUTE destip NUMERIC
@ATTRIBUTE sourceport NUMERIC
@ATTRIBUTE destport NUMERIC
@ATTRIBUTE protocol {tcp, udp, icmp}
@ATTRIBUTE duration NUMERIC
@ATTRIBUTE class {1, 0}
@DATA
1295981622205977,1921683131,2049163166,58443,80,tcp,47357377,?
1561919069964921,19216813,19216822,49526,19780,tcp,0000003,?
1561919069986518,19216813,19216822,49526,8090,tcp,0000004,?
1295981609684965,1921683131,65549568,56174,80,tcp,13589864,?
1295981658219915,172162551,66235143184,10648,443,tcp,95860479,?
1561919069964373,19216813,19216822,49526,1105,tcp,0000004,?
1561919069975717,19216813,19216822,49526,58080,tcp,0000003,?
```

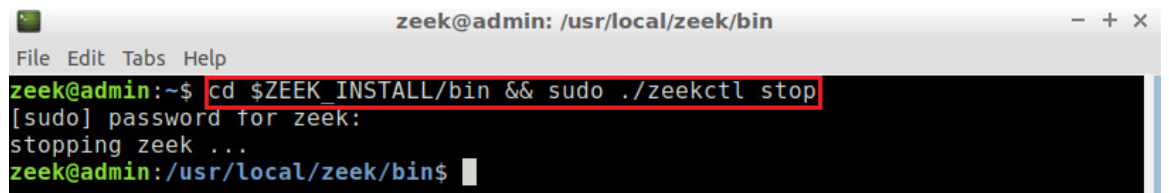
The input test dataset is now a properly formatted *.arff* file and can be input into a machine learning classifier to test the classifier’s accuracy.

3.5 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
```

A terminal window titled "zeek@admin: /usr/local/zeek/bin" with a menu bar "File Edit Tabs Help". The terminal shows the command "cd \$ZEEK_INSTALL/bin && sudo ./zeekctl stop" being entered and highlighted with a red box. The output is "[sudo] password for zeek:", "stopping zeek ...", and the prompt "zeek@admin: /usr/local/zeek/bin\$".

```
zeek@admin:~$ cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
[sudo] password for zeek:
stopping zeek ...
zeek@admin: /usr/local/zeek/bin$
```

References

1. Alpaydin, E., "Introduction to machine learning," MIT press (2009).
2. Holmes, G., Donkin, A., & Witten, I. H. (1994). Weka: A machine learning workbench.
3. "Attribute-relation file format", The university of waikato, [Online], Available: <https://www.cs.waikato.ac.nz/~ml/weka/arff.html>



The University of Texas at San Antonio™

The Cyber Center for Security and Analytics

ZEEK INTRUSION DETECTION SERIES

Lab 12: Developing Machine Learning Classifiers for Anomaly Inference and Classification

Document Version: 03-13-2020



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab topology.....	3
Lab settings	3
Lab roadmap	4
1 Introduction to Weka.....	4
1.1 Starting Weka.....	4
2 Importing a dataset into Weka	6
2.1 Loading the training dataset	7
2.2 Filtering the training dataset.....	9
2.3 Training a decision table classifier	13
2.4 Training a decision tree classifier	15
2.4.1 Updating the decision tree classifier	17
3 Reviewing the classifier's predictions on a test dataset.....	22
3.1 Saving the decision table.....	22
3.2 Using the classifier to predict labels for the test dataset	24
3.3 Viewing the predicted labels for the testdataset	26
References	29

Overview

This lab introduces the application of machine learning in the network security field. The lab explains how to generate a decision table and decision tree to infer scan-related network traffic. The lab is designed to train and test a machine learning classifier using network traffic dataset.

Objectives

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

1. Train a decision table to classify scan-related network traffic.
2. Train a decision tree to classify scan-related network traffic.
3. Test and modify the trained classifiers and review their output classifications on a test dataset.

Lab topology

Figure 1 shows the lab workspace topology. This lab primarily uses the *Client* machine for offline Zeek log file processing and reformatting.

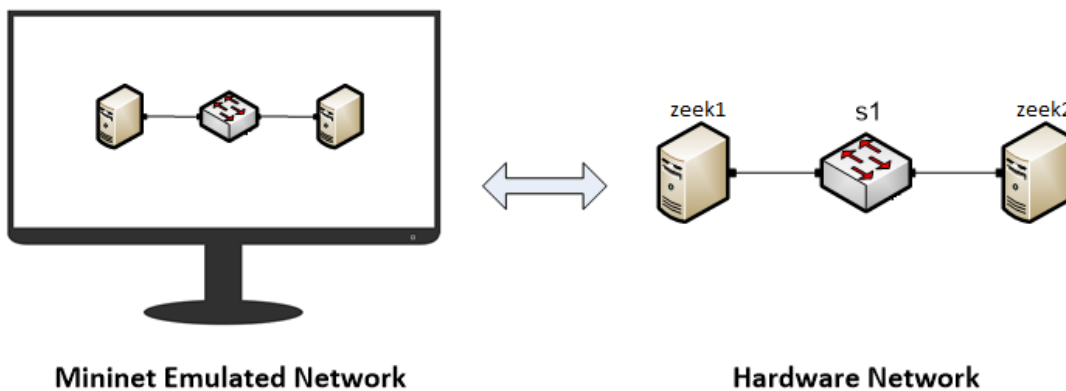


Figure 1. Lab topology.

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: Introduction to Weka.
2. Section 2: Building a decision classifier with Weka.
3. Section 3: Reviewing the classifier's predictions on a test dataset.

1 Introduction to Weka

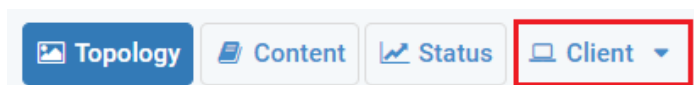
After formatting Zeek output logs into the ARFF files, Weka is now able to process them. Weka contains the algorithms necessary to develop a number of machine learning classifiers. More information on the Weka software can be found on their documentation pages. To access the following link, users must have access to an external computer connected to the Internet, because the Zeek Lab topology does not have an active Internet connection.

```
https://www.cs.waikato.ac.nz/ml/weka/documentation.html
```

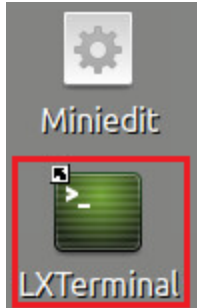
In the following sections, we train a *DecisionTable* and a *J48 Decision Tree* classifier.

1.1 Starting Weka

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.



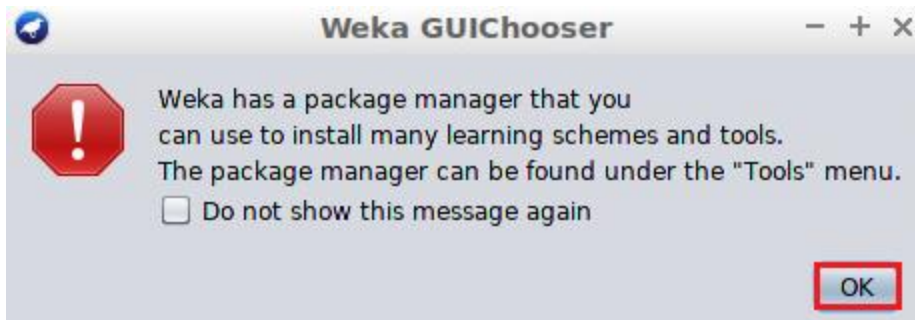
Step 3. Navigate to the Weka workspace directory.



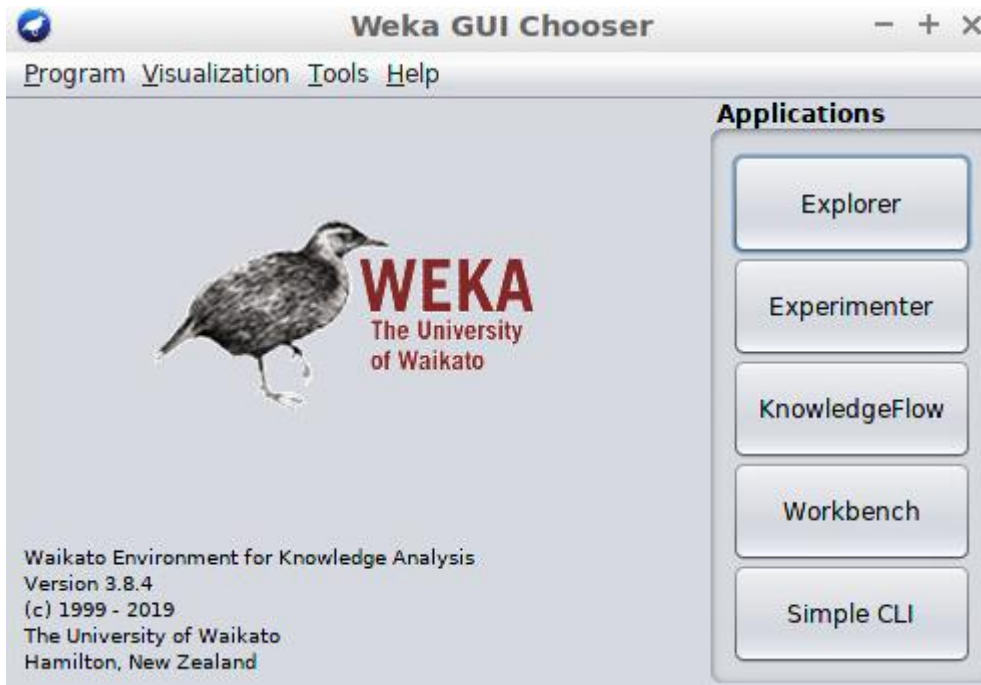
Step 4. Using Java, launch the Weka software.



Step 5. Once Weka has been loaded, a notification containing Weka related information will be displayed. Select the *OK* button to continue to the *Weka GUI Chooser* panel.



The *Weka GUI Chooser* panel will look similar to the following image.



Step 6. For this lab, we will be using the *Explorer* application. Click the *Explorer* button to launch the application.



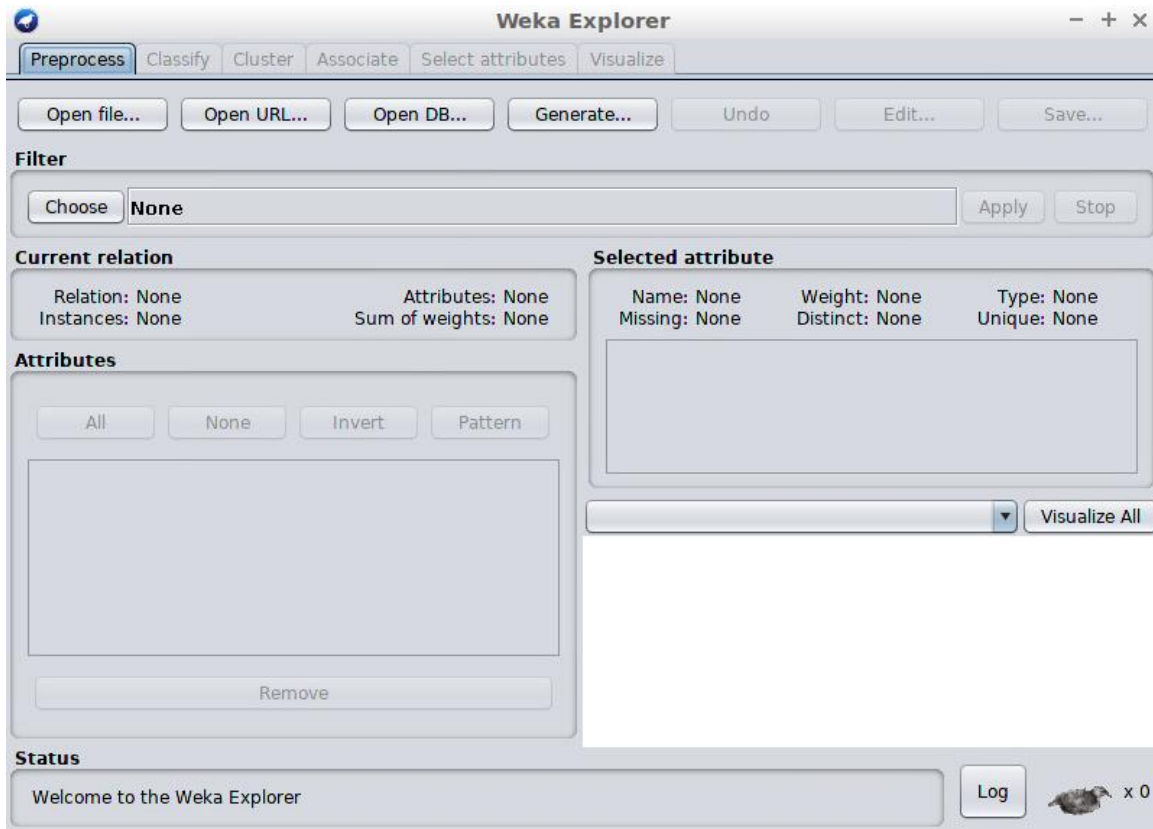
Weka has been successfully launched and we can proceed to the next section.

2 Importing a dataset into Weka

Once the *Explorer* application opens, a new GUI window will be displayed. Initially, this window has all options greyed out, indicating that we have not yet opened or loaded a dataset.

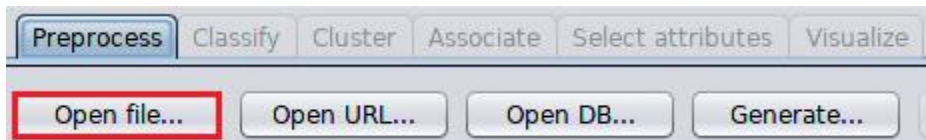
The *Explorer* panel contains a *Menu Bar* located at the top of the GUI window. There is a total of 6 additional panels, which contain related information necessary to train, test and visualize classifiers developed while using Weka. By default, the *Preprocess* panel will be selected.

The *Preprocess* panel is used to import a training dataset to be used for training a machine learning classifier. Features can be removed, randomized or appended within this panel.



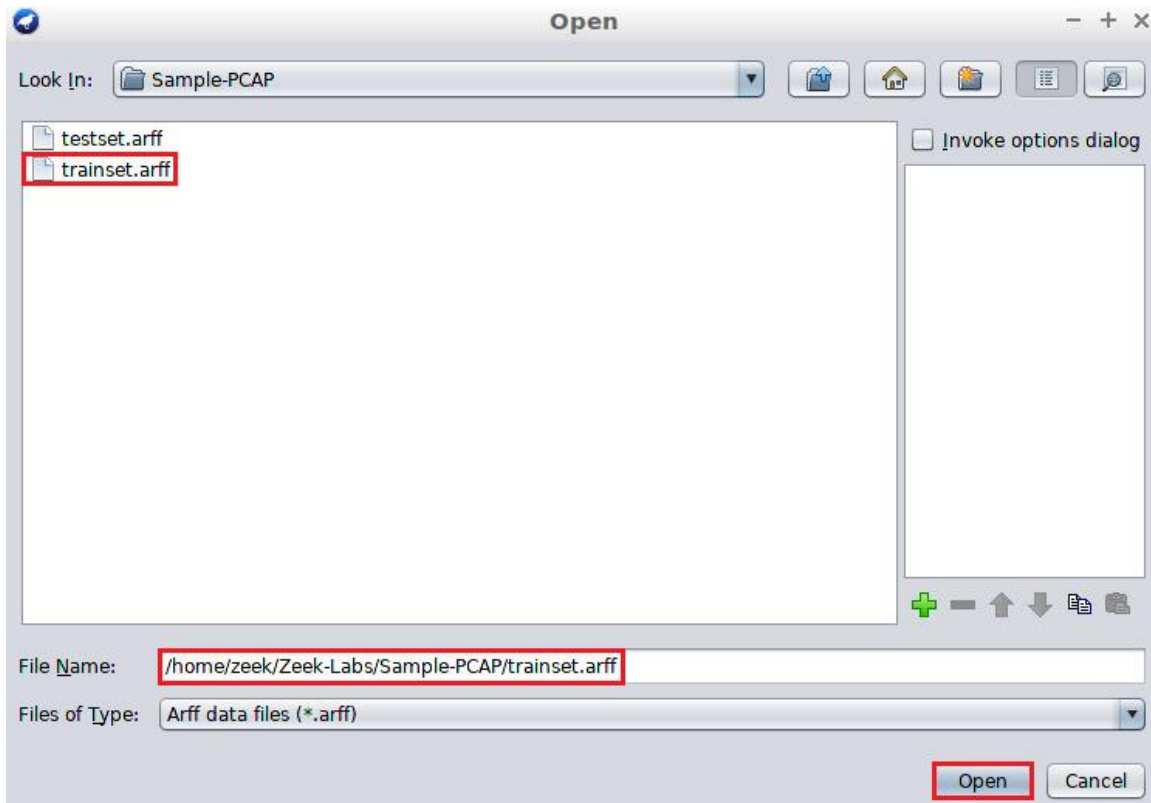
2.1 Loading the training dataset

Step 1. On the top left of the *Preprocess* window the *Open file* button can be found. Click the *Open file* button to load the training dataset.



Step 2. Enter the path to the *trainset.arff* file. Alternatively, use the GUI to navigate to the lab workspace directory to select the file. Use the *Open* button to load the *trainset.arff* file into Weka.

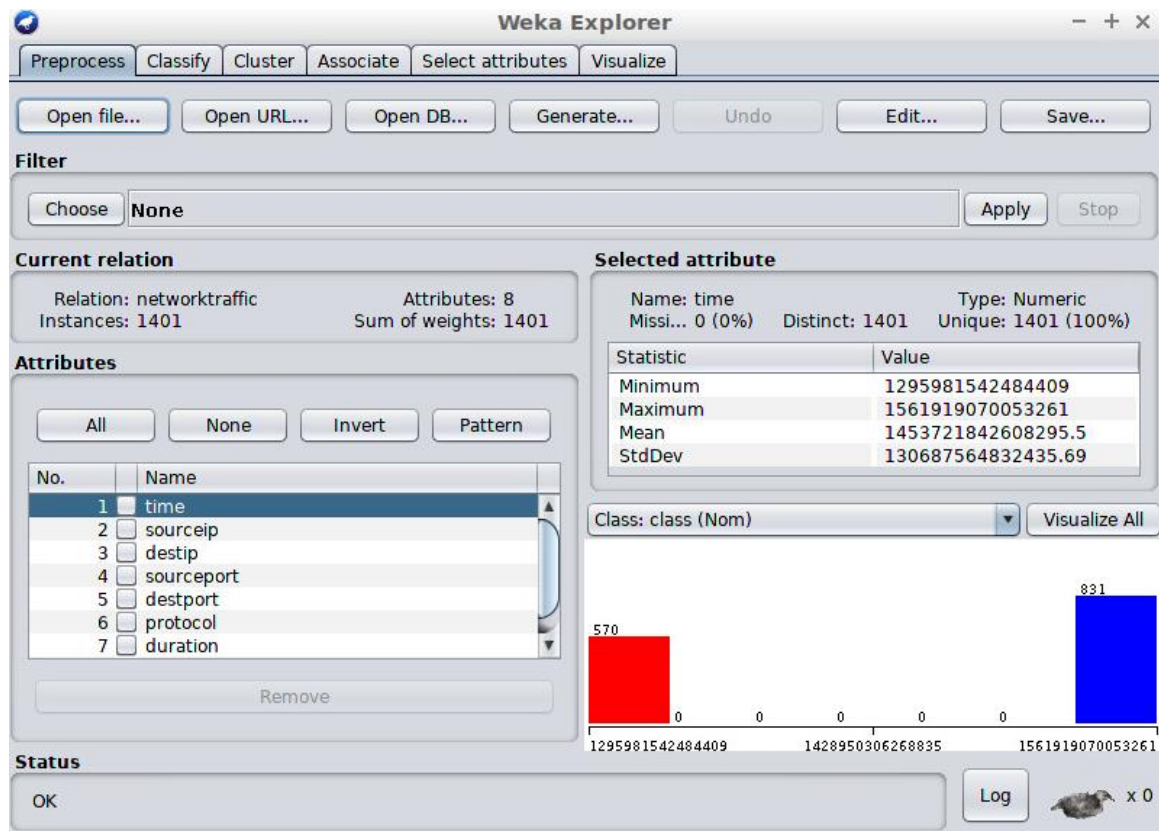
```
/home/zeek/Zeek-Labs/Sample-PCAP/trainset.arff
```



After click the *Open* button, the *Preprocess* panel will be updated to contain the *trainset.arff* file statistics.

Each section header has been highlighted with a red box. We can see that the *Current relation*, *Attributes* and *Selected attribute* sections have been updated to contain *trainset.arff* file data.

Step 3. Within the *Attributes* section, click the *class* feature to change the active attribute.



By selecting the *class* feature within the *Attributes* section, the *Explorer* panel will be updated to display the active feature.

Within the *Current relation* section, our dataset's name, *networktraffic*, is displayed. Additionally, it is shown that the dataset contains 1401 unique data objects (instances).

Within the *Selected attribute* section, the *class* labels added to the dataset in the previous lab are counted. The *trainset.arff* dataset contains 831 data objects labeled with a 1, belonging to the malicious class, while 570 data objects are labeled with a 0, belonging to the benign class.

At this point, *trainset.arff* dataset has been successfully loaded into Weka and we can begin filtering the data before training a machine learning classifier.

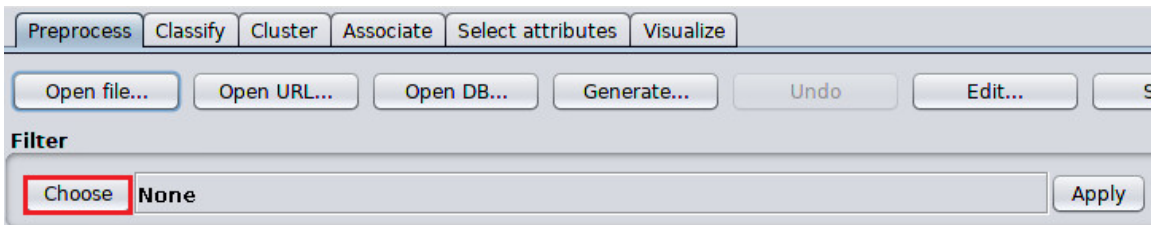
2.2 Filtering the training dataset

The majority of machine learning classifiers are unable to handle string attributes. For network analysts, source and destination IP addresses are valuable features that are often necessary for traffic analysis. However, these IP addresses are unable to be stored as string values when training a classifier.

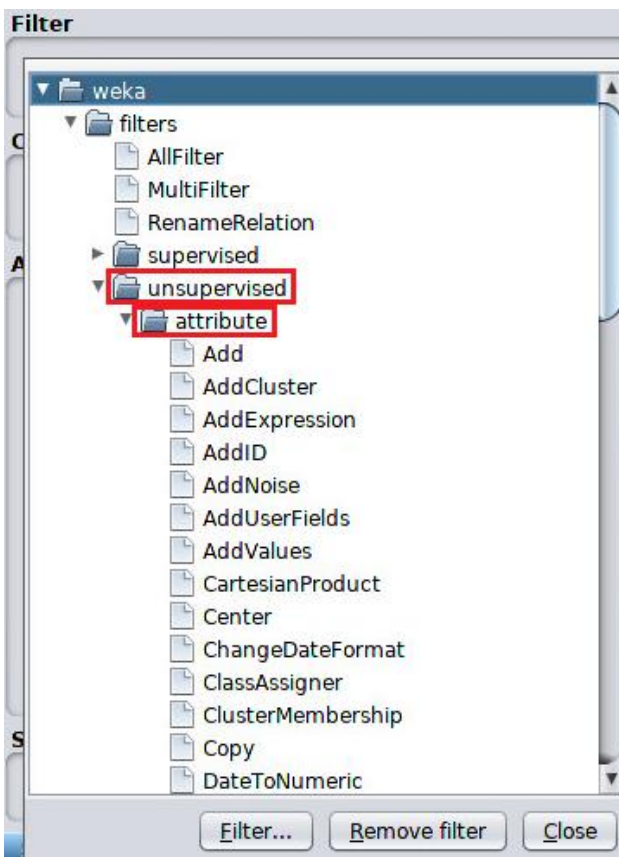
There is a number of ways to address this issue. If a network analyst were to know all of the unique IP addresses, when generating their ARFF dataset, they can create the nominal values similar to how we created the nominal protocol values.

Because Internet-scale traffic contains a very large number of unique IP addresses, the aforementioned process may not be feasible. Therefore, in the previous lab, we converted our source and destination IP addresses into numerical values. In this section, we will be using all unique iterations of the numerical values to generate a nominal list. By reformatting the IP addresses into numeric values using Terminal utilities, the Weka software will be able to select all unique IP addresses and convert them into a nominal feature set.

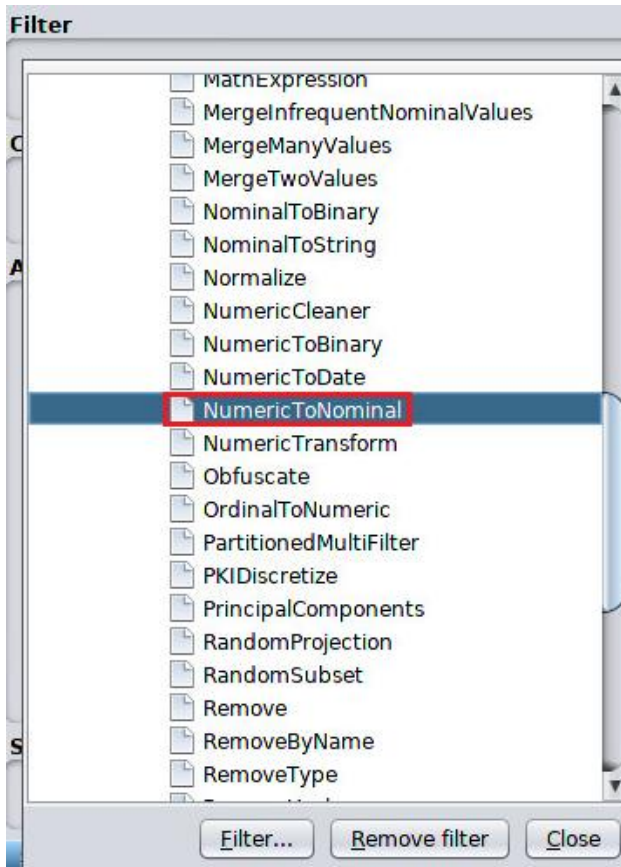
Step 1. Within the *Preprocess* tab, under the *Filter* section, click the *Choose* button.



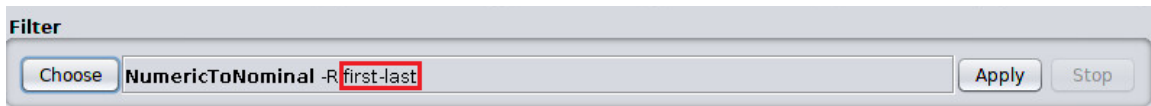
Step 2. Under the *unsupervised* option, select the *attribute* option to display a list of attribute-based filters.



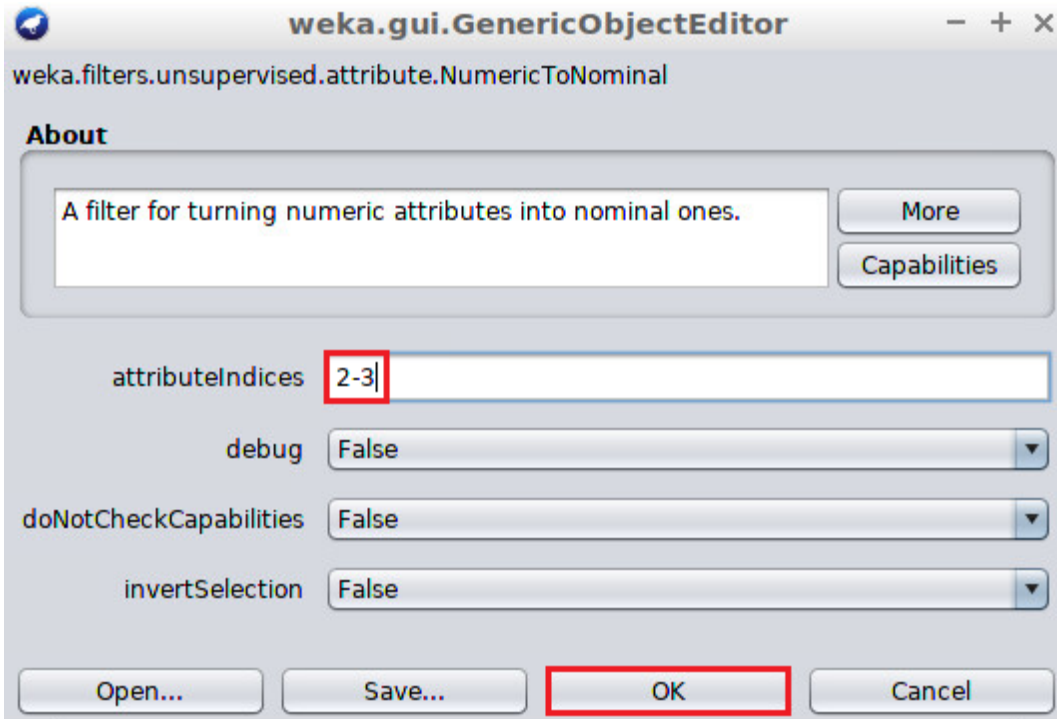
Step 3. Scroll to the *NumericToNominal* filter and double click to select it.



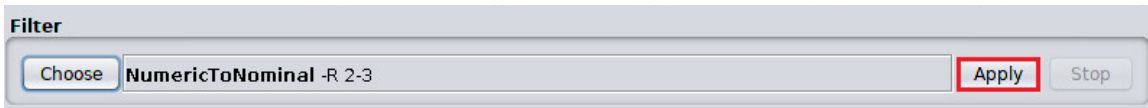
Step 4. Within the *Filter* section, click the *first-last* description to modify the filter.



Step 5. Update the *Indexes of the Attributes* to be filtered. Click the *Apply* button to edit the indexes.



Step 6. On the right side of the *Filter* section, click the *Apply* button to apply the modified filter.

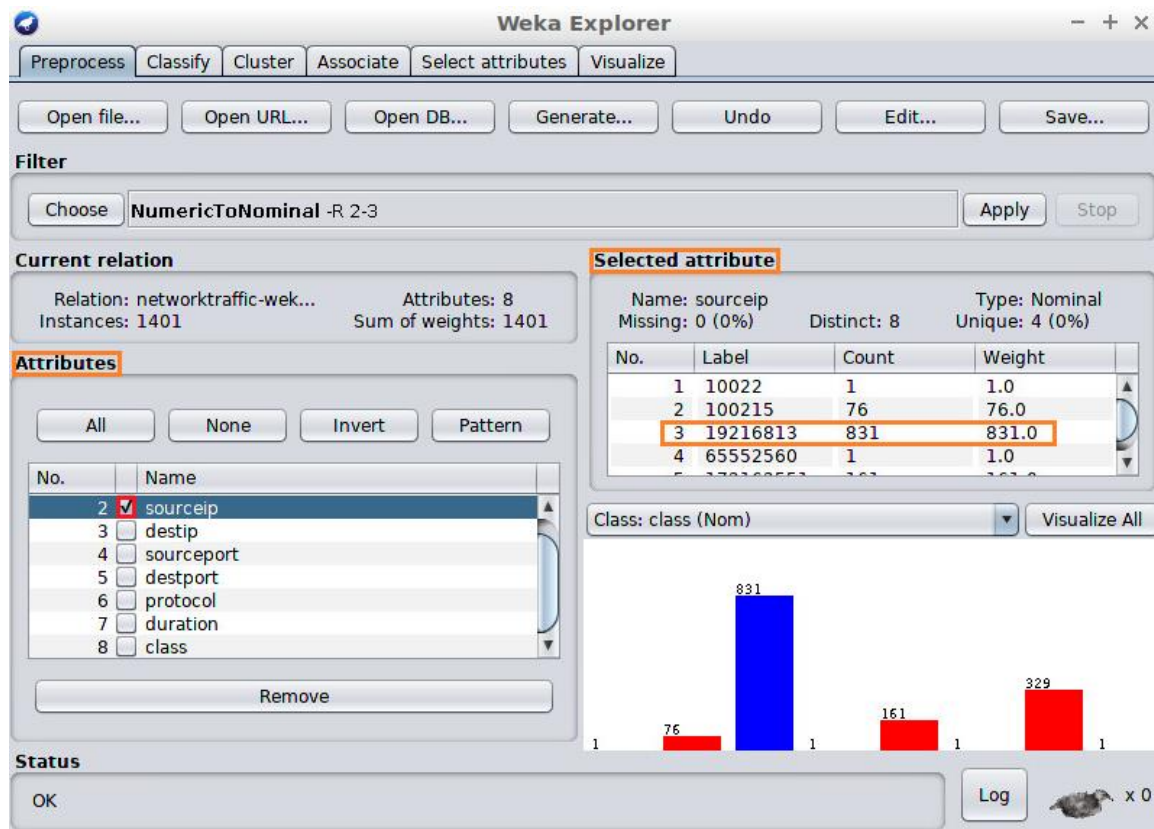


The source and destination IP addresses will now be converted to the Nominal feature type.

Step 7. Within the *Attributes* section, click the *sourceip* feature to change the active attribute.

By selecting the *sourceip* feature within the *Attributes* section, the *Explorer* panel will be updated to display the active feature.

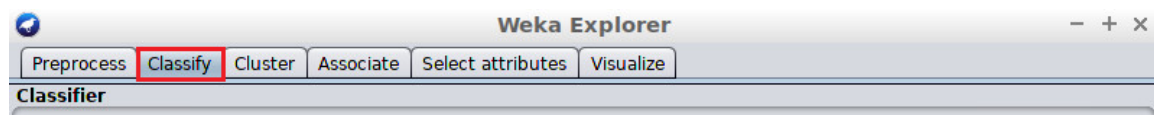
Within the *Selected attribute* section, the *sourceip* feature will now display the Nominal data objects. In the following image, the highlighted *sourceip* related to the scanning machine's IP address (192.168.1.3), displays 831 unique instances being recorded.



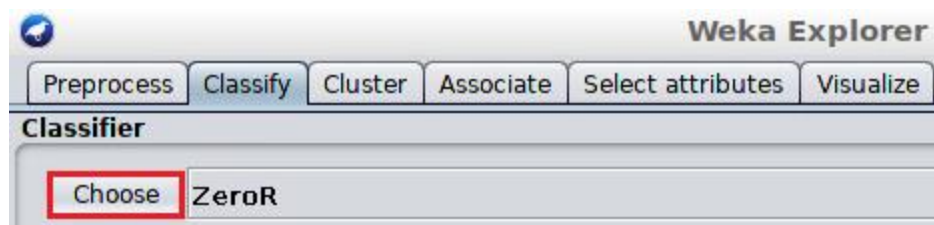
Additionally, the *Selected attribute* section will be updated to show new statistics for each feature. The updated *Selected attribute* section is displayed in the previous image.

2.3 Training a decision table classifier

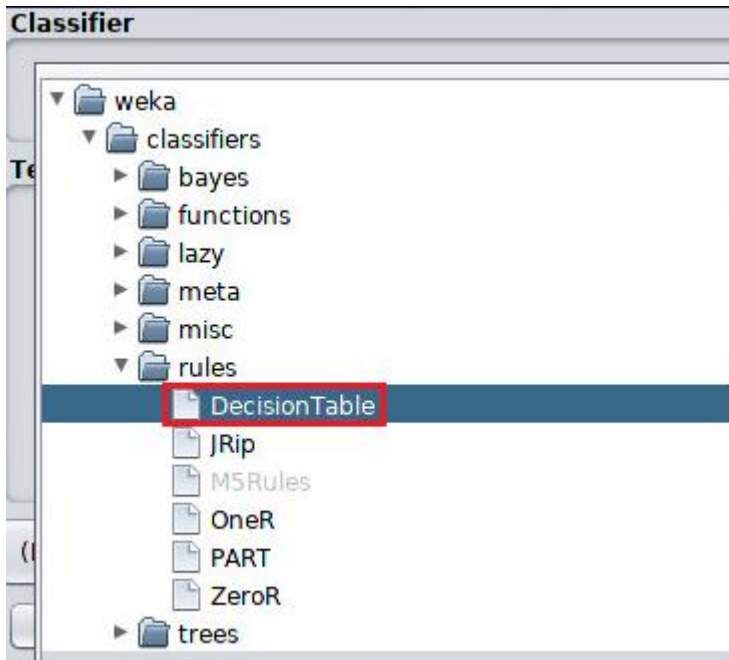
Step 1. Within the *Explorer* panel, click the *Classify* tab located at the top of the *Explorer* panel to switch to the *Classify* panel.



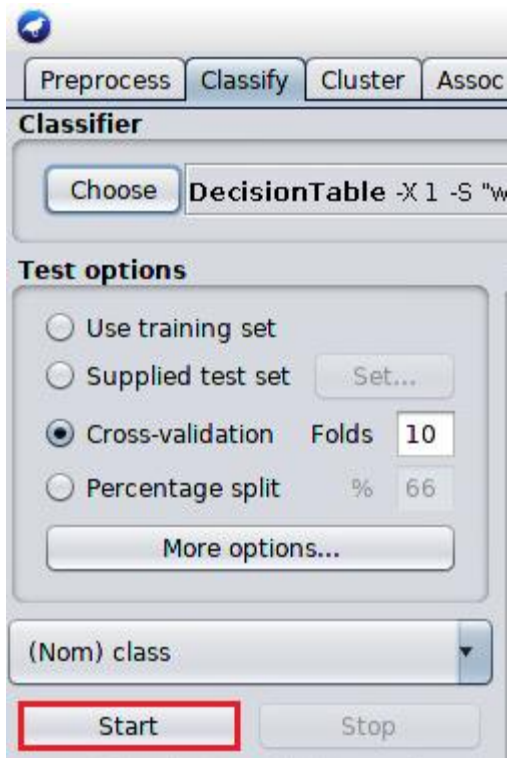
Step 2. Once the *Classify* panel has loaded, click the *Choose* button within the *Classifier* section to select which machine learning classifier we are developing.



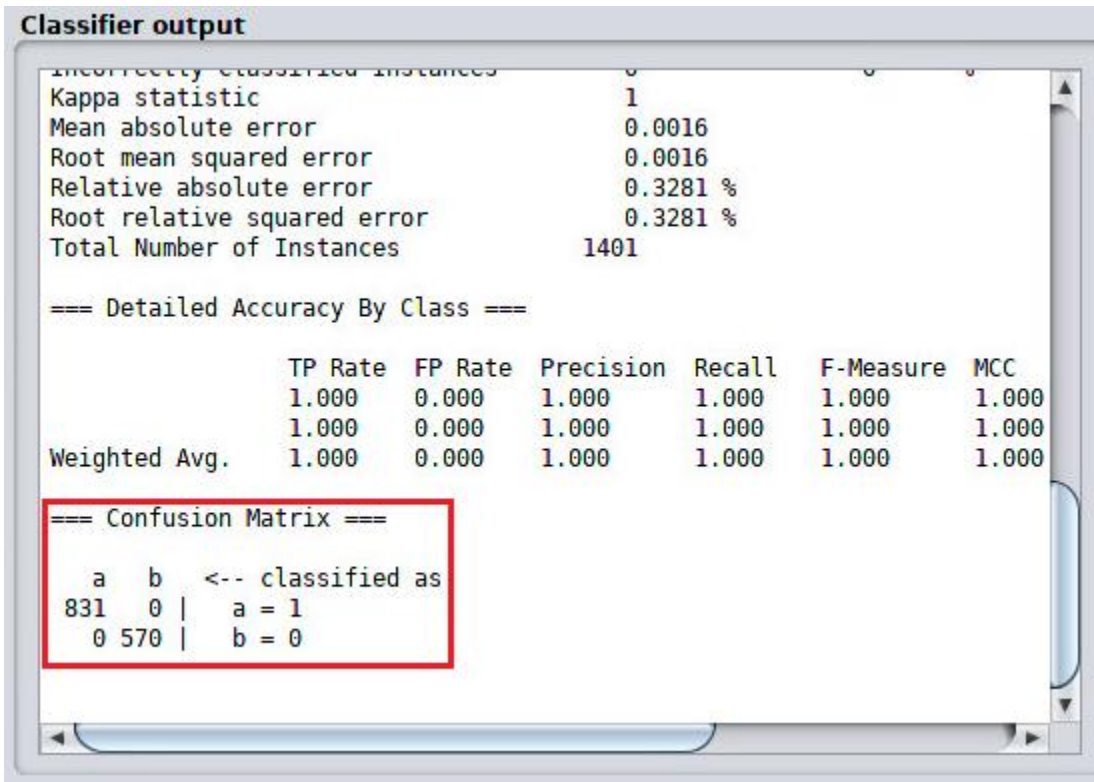
Step 3. Under the *rules* collection, double-click with your mouse to select the *DecisionTable* classifier.



Step 4. Under the *Test options* section, click the *Start* button to begin training the classifier. Notice the *Classifier* section has been updated to display the *DecisionTable* classifier.



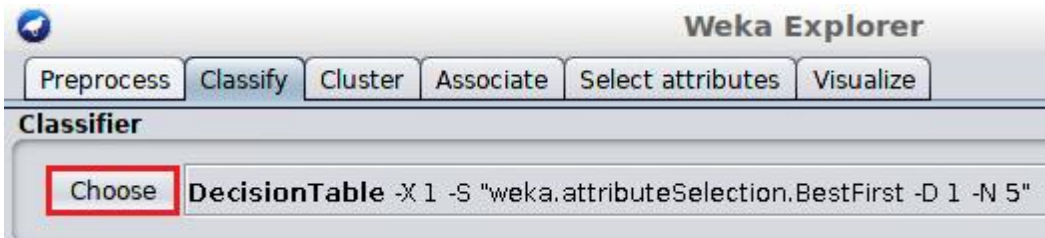
Step 4. See the *Decision Table* classifier's results.



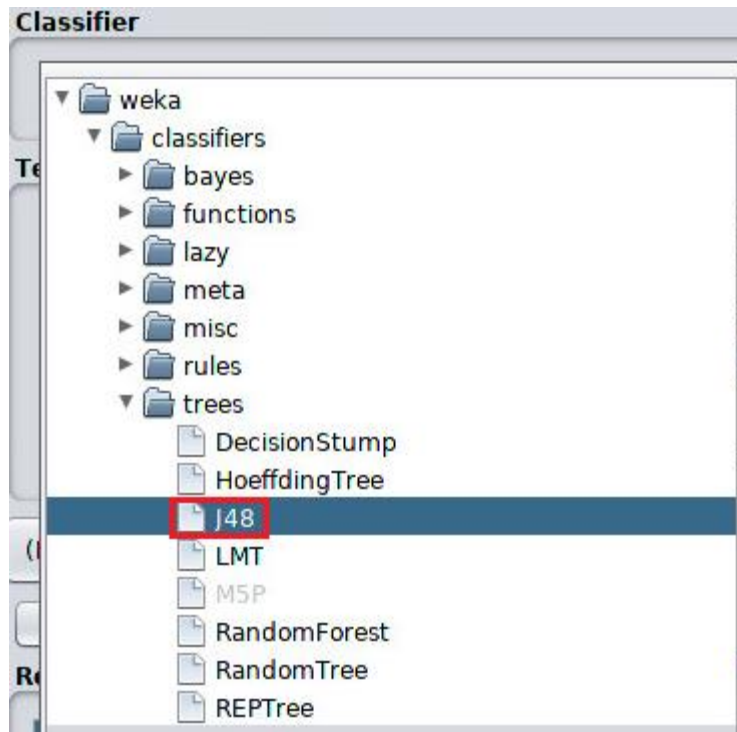
Within the *Result list* section we can see our new *Decision Table* that has been trained with the *transet.arff* dataset. Within the *Classifier output* section, we can see the prediction results for the *Decision Table* classifier. The *Confusion Matrix* depicts that the classifier had a 100% accuracy when predicting labels after being trained.

2.4 Training a decision tree classifier

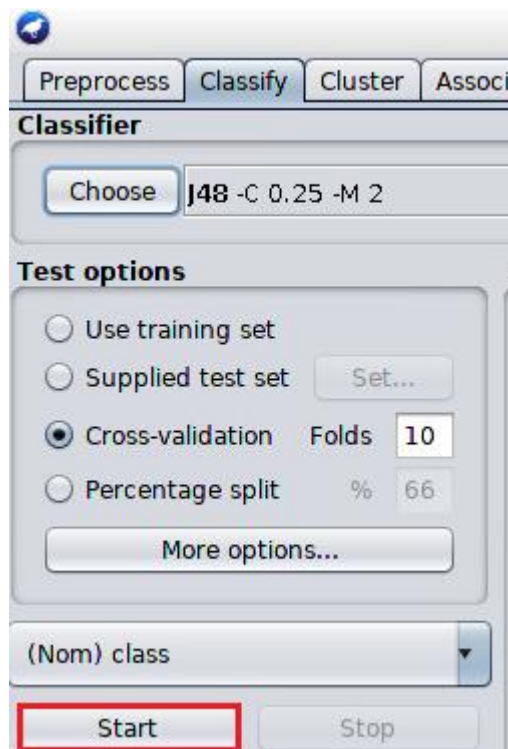
Step 1. Click the *Choose* button within the *Classifier* section to select which machine learning classifier we are developing.



Step 2. Under the *trees* collection, double-click with your mouse to select the *J48* decision tree classifier.



Step 3. Under the *Test options* section, click the *Start* button to begin training the classifier. Notice the *Classifier* section has been updated to display the *J48* classifier.



Step 4. See the *J48 Decision Tree* classifier's results.

```

Classifier output
Incorrectly classified instances      1
Kappa statistic                     0.9985
Mean absolute error                  0.0007
Root mean squared error              0.0267
Relative absolute error              0.1479 %
Root relative squared error          5.4385 %
Total Number of Instances           1401

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
                1.000   0.002   0.999     1.000   0.999     0.999
                0.998   0.000   1.000     0.998   0.999     0.999
Weighted Avg.   0.999   0.001   0.999     0.999   0.999     0.999

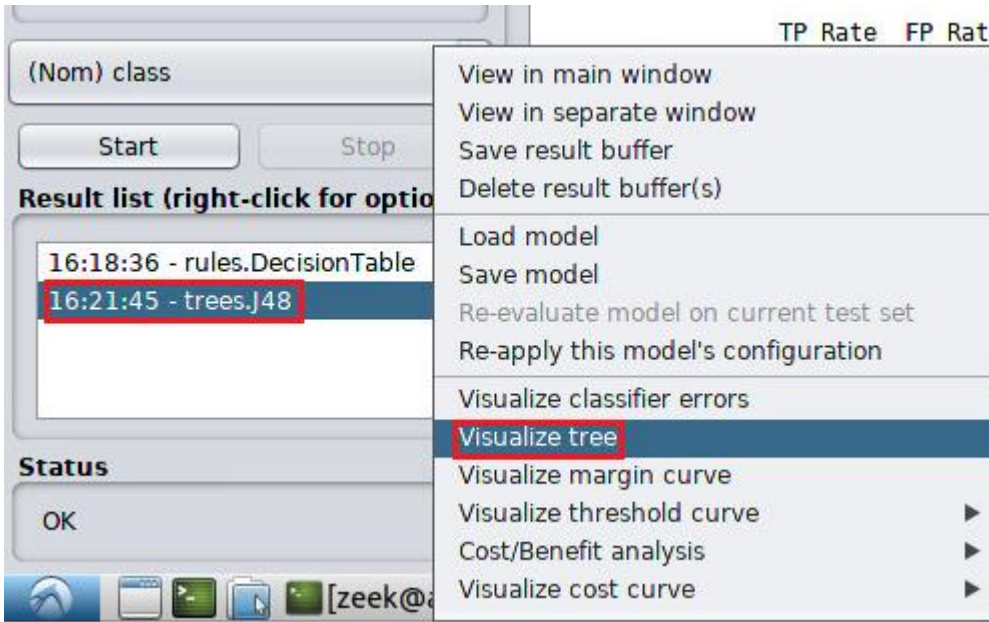
=== Confusion Matrix ===
  a  b  <-- classified as
831  0 |  a = 1
 1 569 |  b = 0
    
```

Within the *Result list* section we can see our new *J48 Decision Tree* that has been trained with the *transet.arff* dataset. Within the *Classifier output* section, we can view the prediction results for the *Decision Tree* classifier. The *Confusion Matrix* depicts that the classifier did not have a 100% accuracy when predicting labels after being trained and misclassified a single malicious data packet as benign.

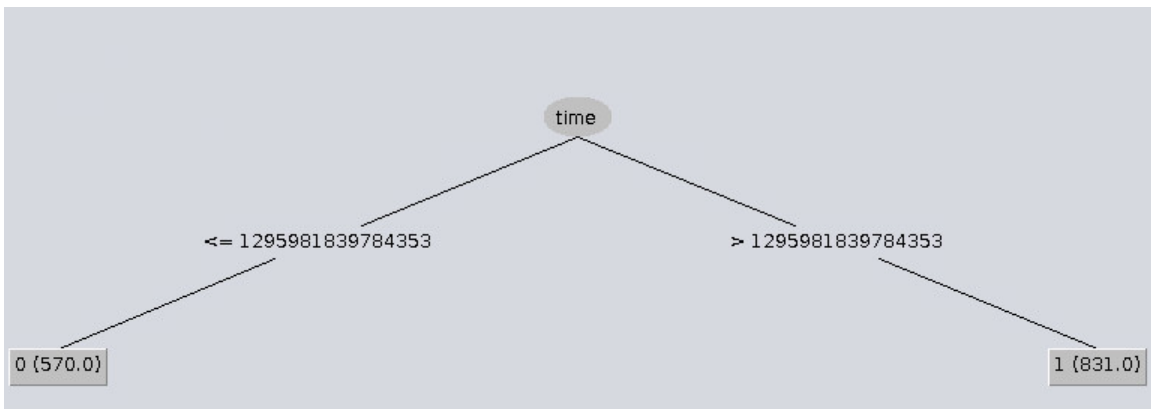
2.4.1 Updating the decision tree classifier

Because our *J48 Decision Tree* has made an error in predicted a label, we can attempt to remove or add additional features to increase the classifier’s accuracy.

Step 1. Right click the *J48 Decision Tree* under the *Result list* section to display more options. Click to *Visualize* the *J48 Decision Tree*.



Step 2. View the *Visualized J48 Decision Tree*.

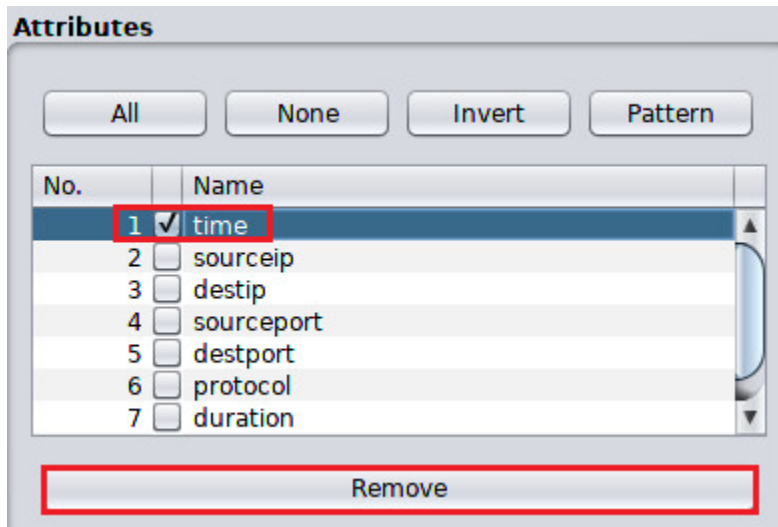


We can see the *time* feature column was the only decision node within the tree. For the purposes of this lab, the datasets were collected at varying times; therefore, the decision tree had an over reliance on the *time* feature to determine when the malicious and benign events took place.

Step 3. Within the *Explorer* panel, click the *Preprocess* tab located at the top of the *Explorer* panel to switch to the *Preprocess* panel.

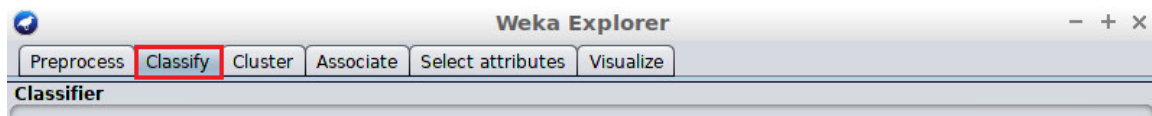


Step 4. Once the *Preprocess* tab has loaded, click the *time* feature within the *Attributes* section and select the *Remove* button.

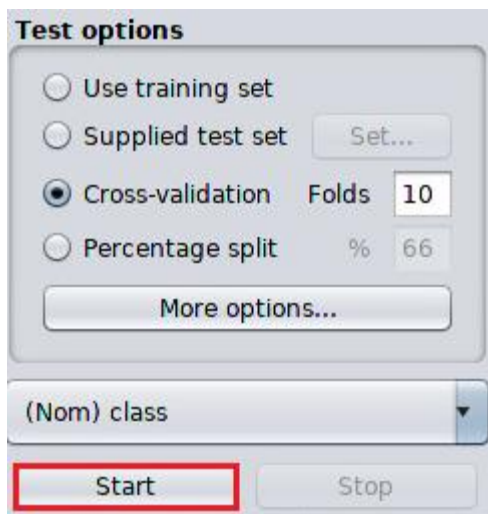


With the *time* feature removed, we can retrain our decision tree to view the new accuracy.

Step 5. Within the *Explorer* panel, click the *Classify* tab located at the top of the *Explorer* panel to switch to the *Classify* panel.



Step 6. The *J48 Decision Tree* should still be selected. Under the *Test options* section, click the *Start* button to begin training the classifier. Notice the *Classifier* section has been updated to display the new *J48* classifier.



Step 7. See the *J48 Decision Tree* classifier's results.

```

Classifier output
Incorrectly classified instances      0      0.0000 %
Kappa statistic                      0.9926
Mean absolute error                  0.003
Root mean squared error              0.0415
Relative absolute error               0.6131 %
Root relative squared error           8.4411 %
Total Number of Instances            1401

=== Detailed Accuracy By Class ===

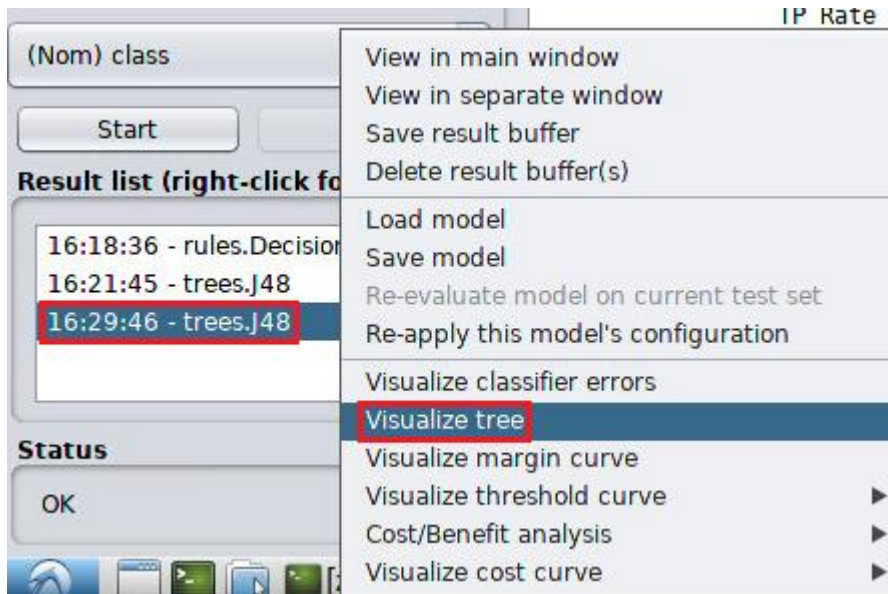
                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
                0.999   0.007   0.995     0.999   0.997     0.993
                0.993   0.001   0.998     0.993   0.996     0.993
Weighted Avg.   0.996   0.005   0.996     0.996   0.996     0.993

=== Confusion Matrix ===
  a  b  <-- classified as
830  1 |  a = 1
 4 566 |  b = 0
    
```

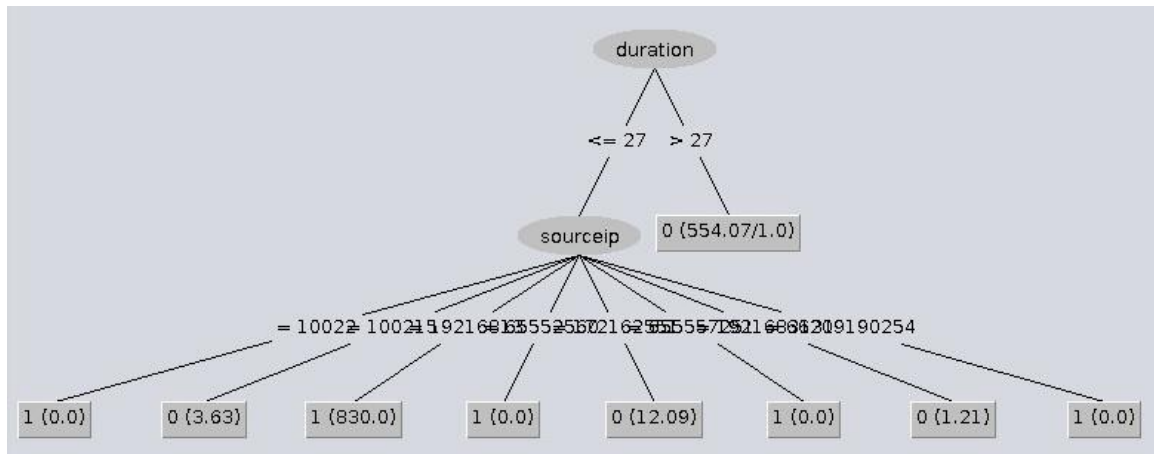
Within the *Result list* section we can see our new *J48 Decision Tree* that has been trained with the *transet.arff* dataset. Within the *Classifier output* section, we can view the prediction results for the *Decision Tree* classifier. The *Confusion Matrix* depicts that the classifier actually had a worse accuracy than the previously trained *J48 Decision Tree*.

In this example, we highlight the importance of choosing the best fit features when training a classifier. In a real-time network environment, it may take multiple tests before discovering which features are necessary for classifying a specific anomaly.

Step 6. Right click the newest *J48 Decision Tree* under the *Result list* section to display more options. Click to *Visualize the J48 Decision Tree*.

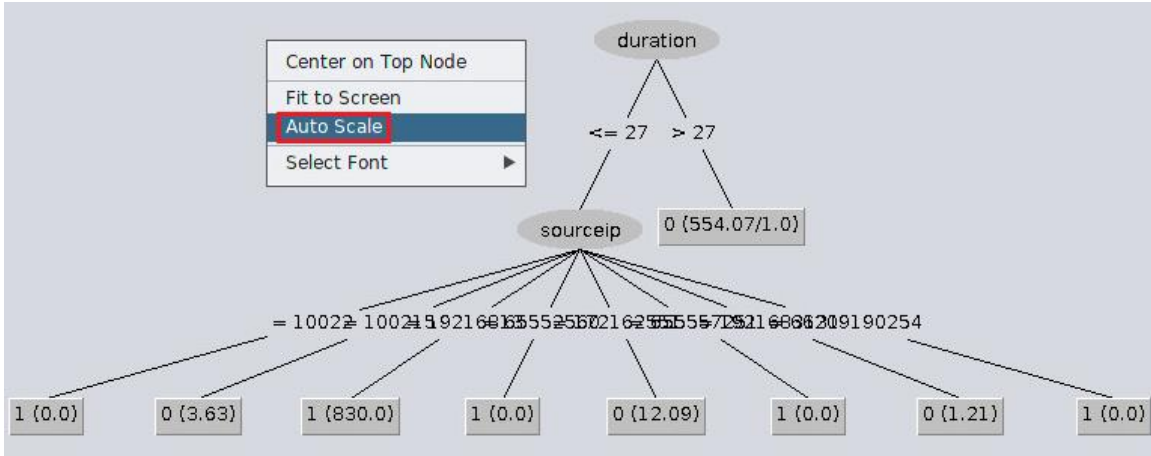


Step 7. View the *Visualized J48 Decision Tree*.

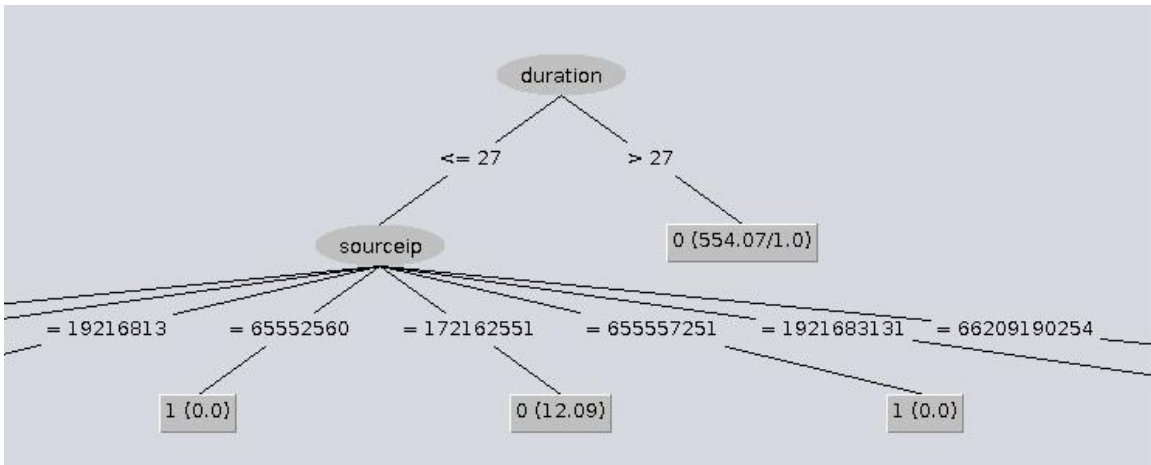


Because the *Decision Tree* has a larger number of nodes, we are unable to see some of the decision thresholds. The following steps will explain how to scale the *Visualized tree*.

Step 8. Right click on the *Visualized J48 Decision Tree* and select the *Auto Scale* option.



Step 8. View the *Visualized J48 Decision Tree*.



Here we can see the new *J48 Decision Tree* has multiple layers and decision nodes. The *duration* feature has replaced the *time* feature as the root node, and the *sourceip* feature is used to further classify the dataset. However, because this tree has reduced accuracy, we will be continuing the lab by using the *Decision Table* created initially.

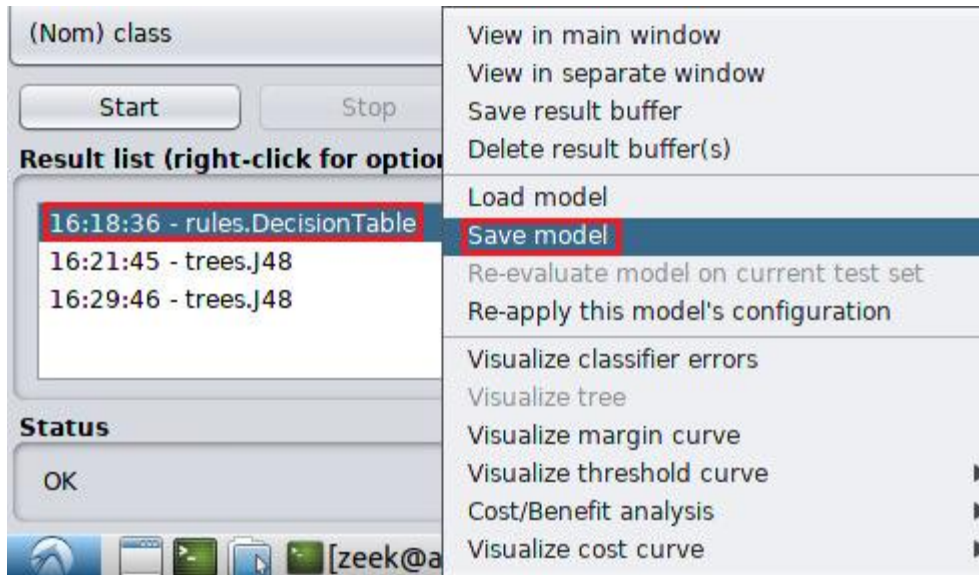
3 Reviewing the classifier’s predictions on a test dataset

Now that we have determined that the *Decision Table* was a more accurate classifier, we can begin testing the classifier’s accuracy using the test dataset.

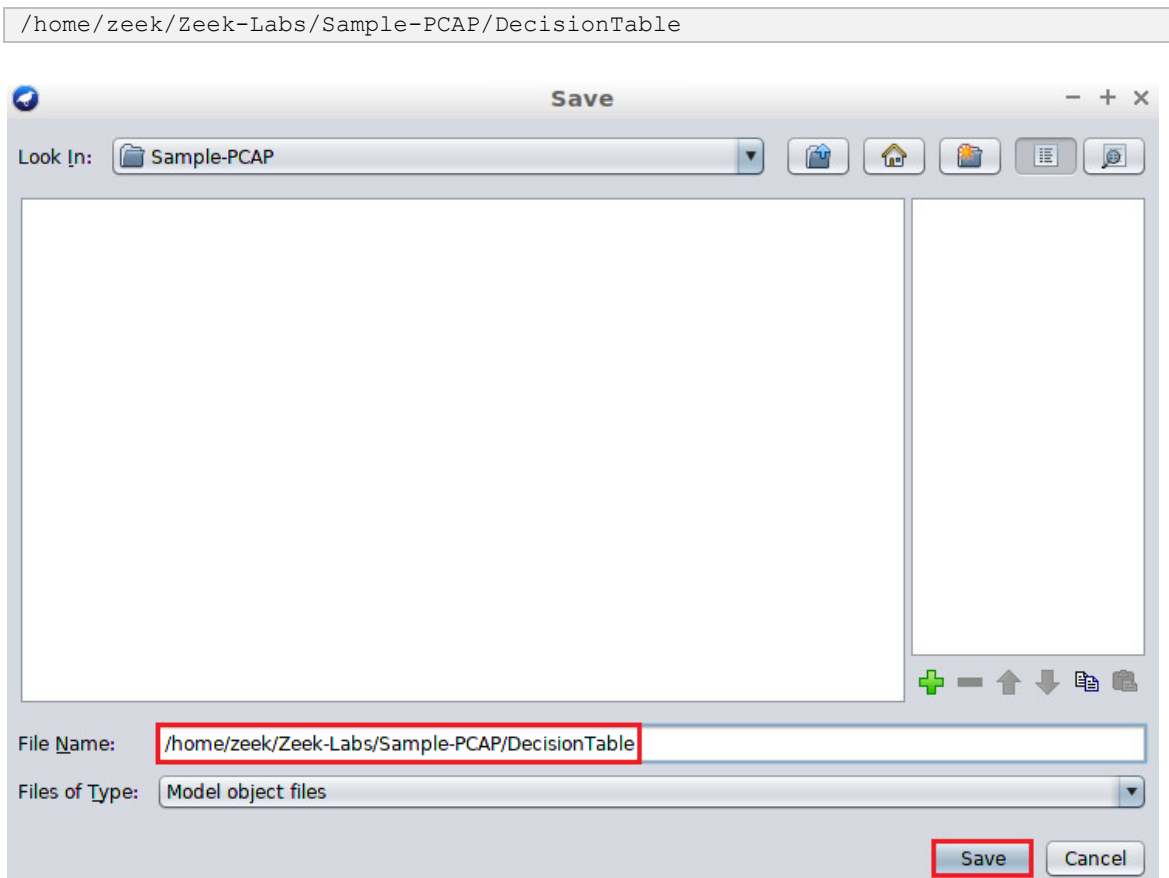
3.1 Saving the decision table

It is possible to save a trained classifier to be reused in future instances of testing and classification. This section will introduce how to save a trained classifier.

Step 1. Under the *Result list* section, right click on the *Decision Table* and select *Save model*.



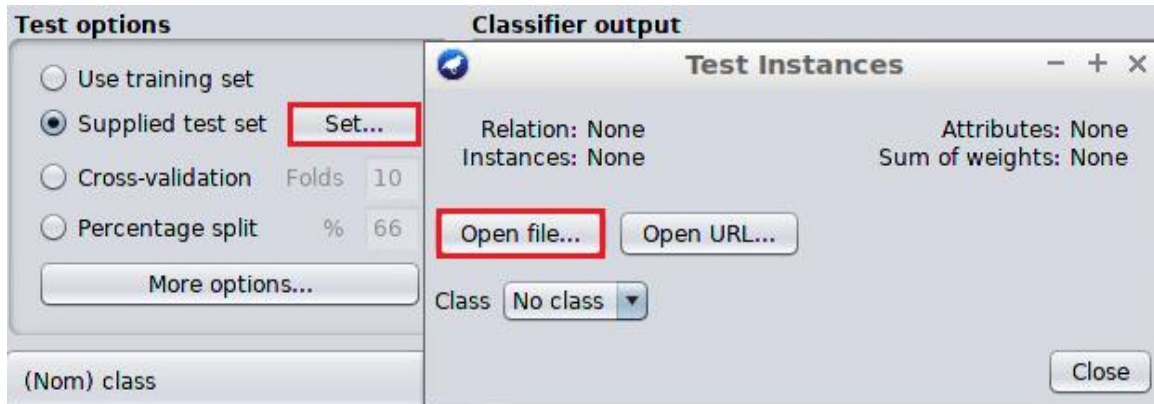
Step 2. Navigate to the Lab workspace directory and save the *Decision Table*. Alternatively, use the GUI to navigate to the lab workspace directory to select the file. Use the *Save* button to save the new *DecisionTable* file into Weka.



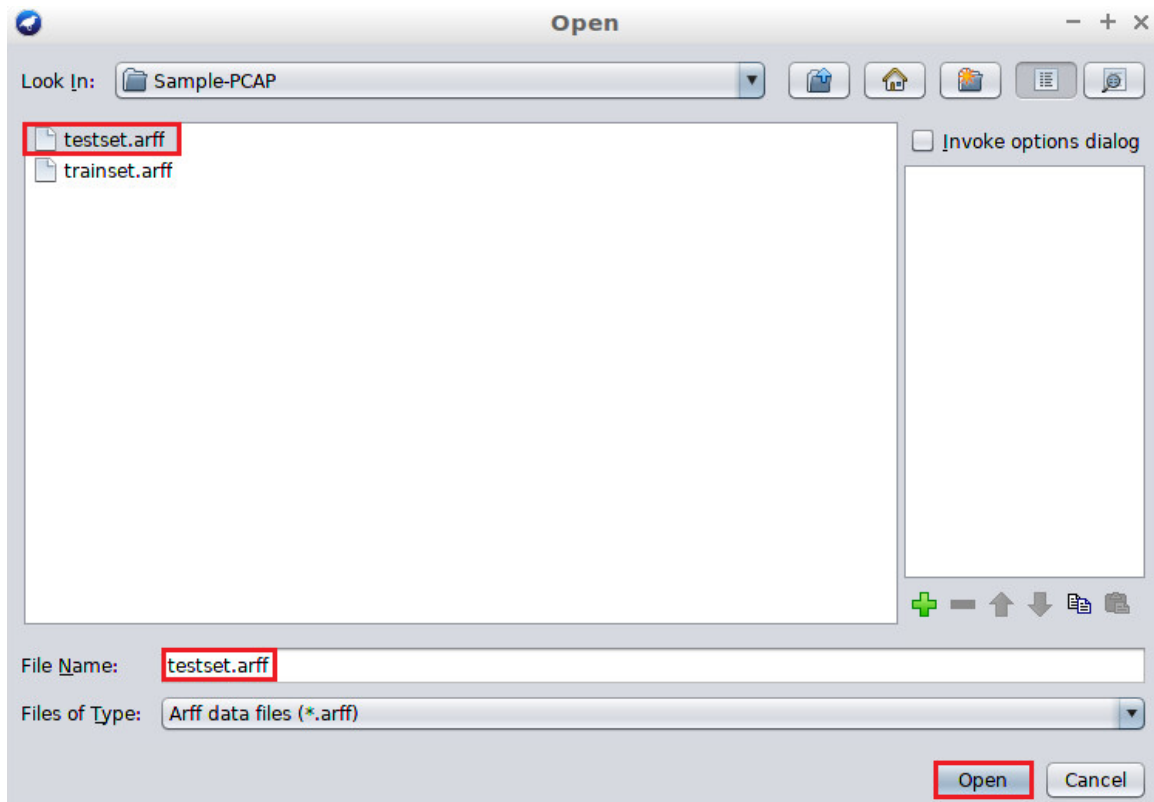
Once saved, we can proceed to testing the classifier's accuracy on predicting labels for the test dataset.

3.2 Using the classifier to predict labels for the test dataset

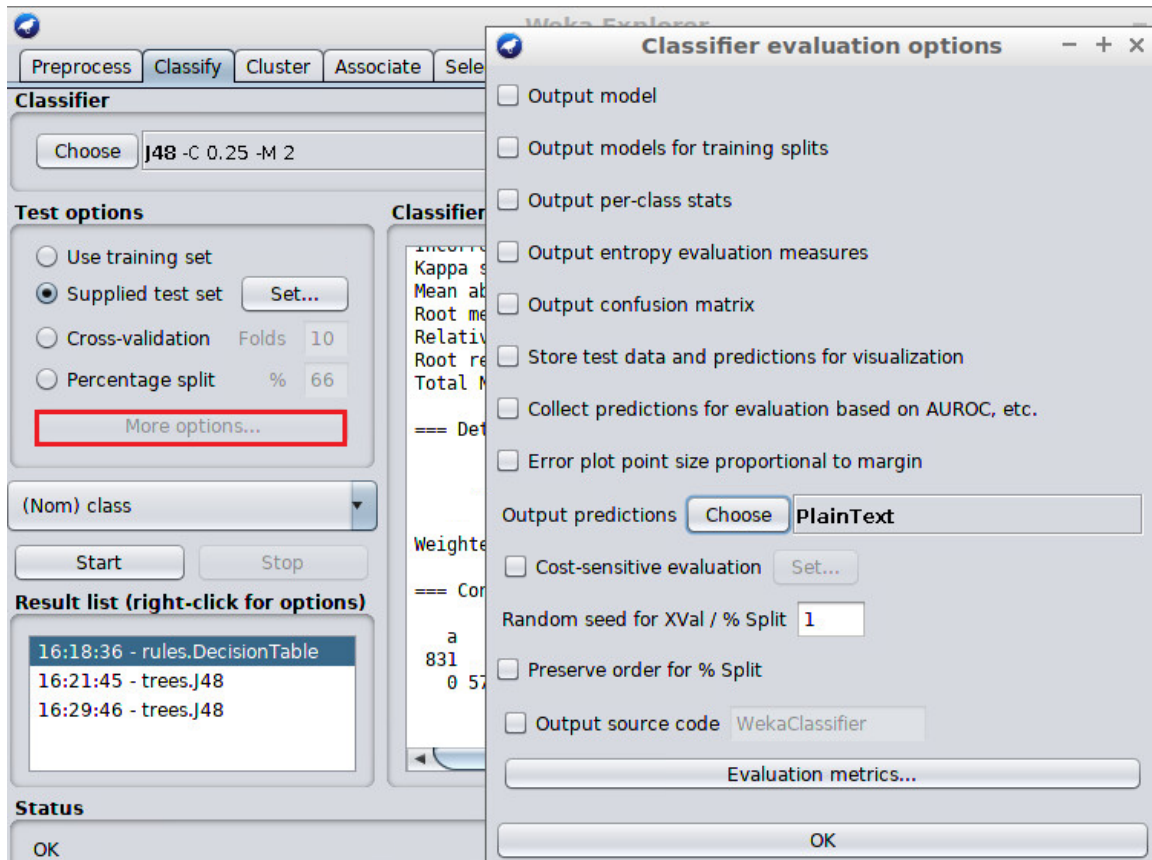
Step 1. Under the *Test options* section, select the *Set* button to load the test dataset. Within the *Test Instances* window, click the *Open file* button.



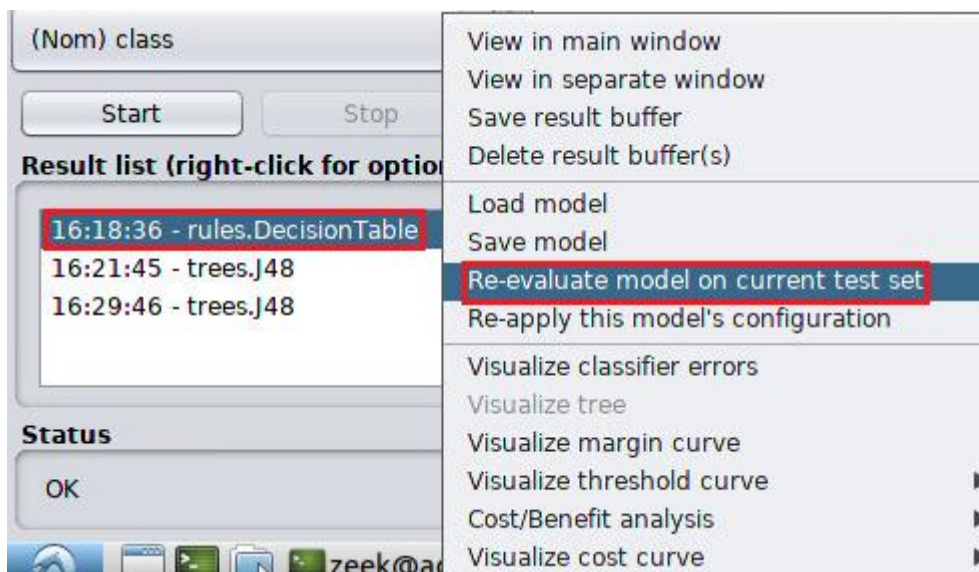
Step 2. Select the *testset.arff* file and click the *Open* button to load the test dataset.



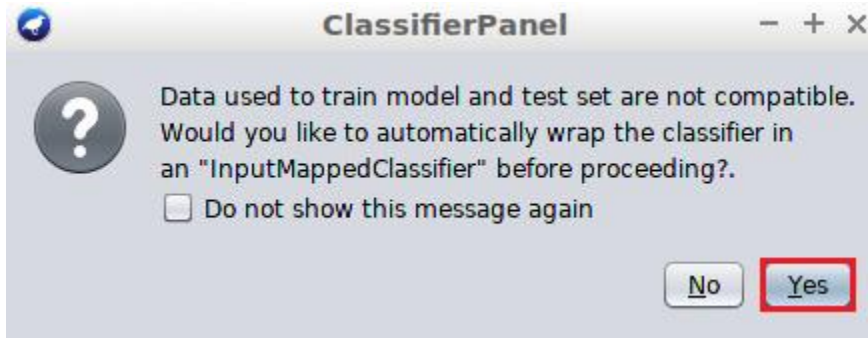
Step 3. Under the *Test options* section, select the *More options* button to configure the classifier to match the following image then, click on *OK*.



Step 4. Under the *Result list* section, right click on the *Decision Table* and select *Re-evaluate model on current test set*.



Step 5. After filtering the *sourceip* and *destip* features into Nominal attributes, the *testset.arff* file will not be properly formatted. Weka will need to update the *testset.arff* dataset to be used by the classifier. Select the *Yes* button on the *ClassifierPanel* pop-up panel.

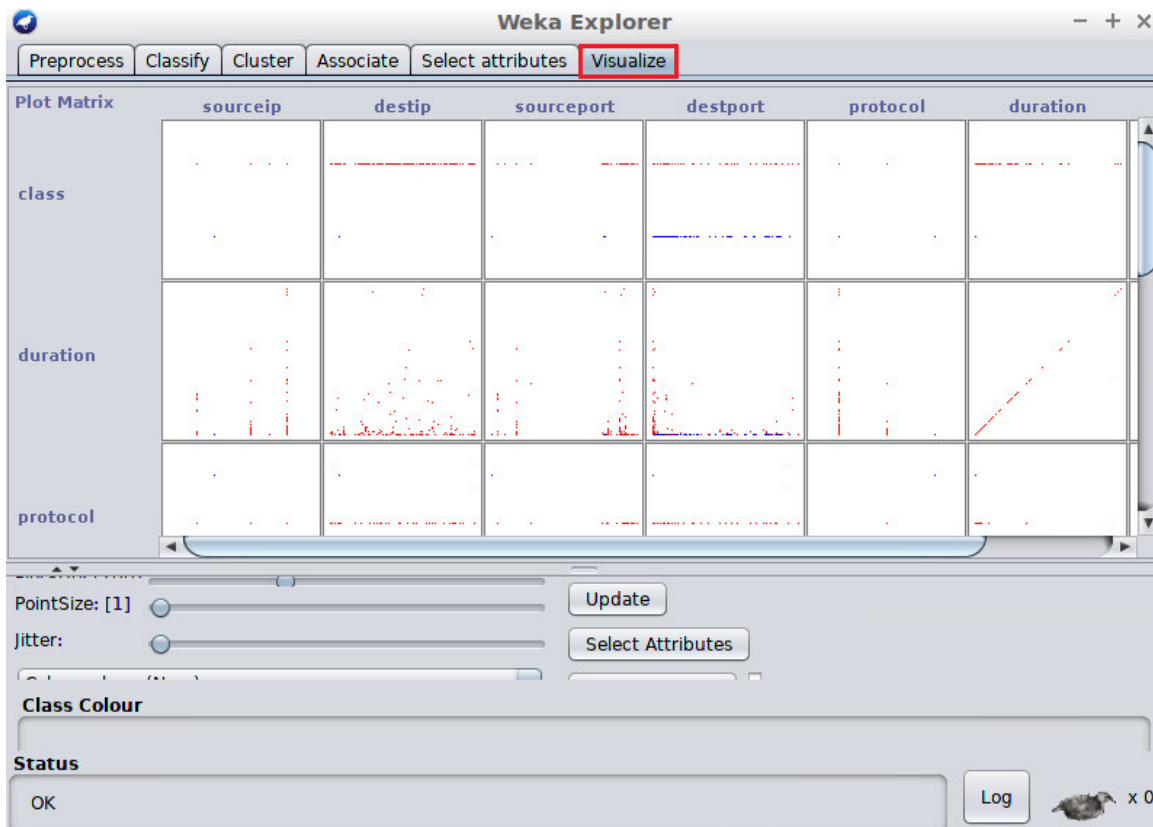


The classifier will generate new predictions, which can be viewed by saving the resulting *.arff* file.

3.3 Viewing the predicted labels for the testdataset

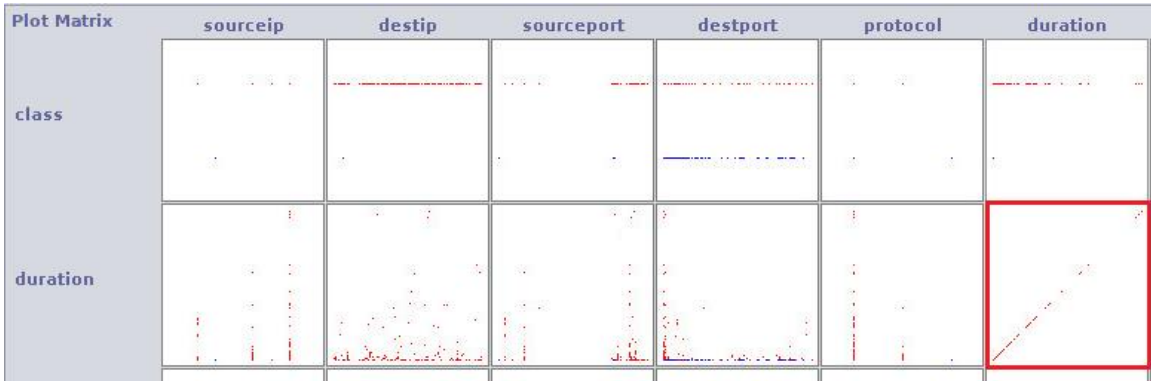
To save the resulting *.arff* file,

Step 1. Within the *Explorer* panel, click the *Visualize* tab located at the top of the *Explorer* panel to switch to the *Visualize* panel .

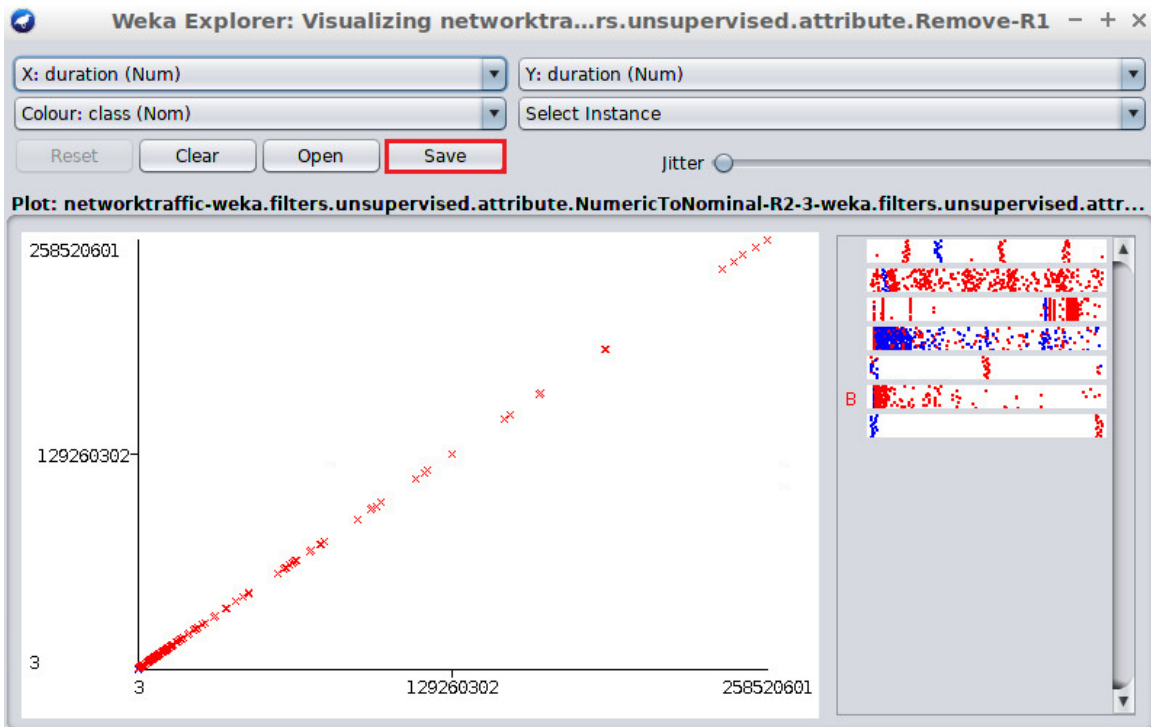


Displayed will be resulting graphs from attribute correlations solved by the machine learning classifier.

Step 2. Select the *duration x duration* graph, found in the sixth column (*duration*) and second row (*duration*).

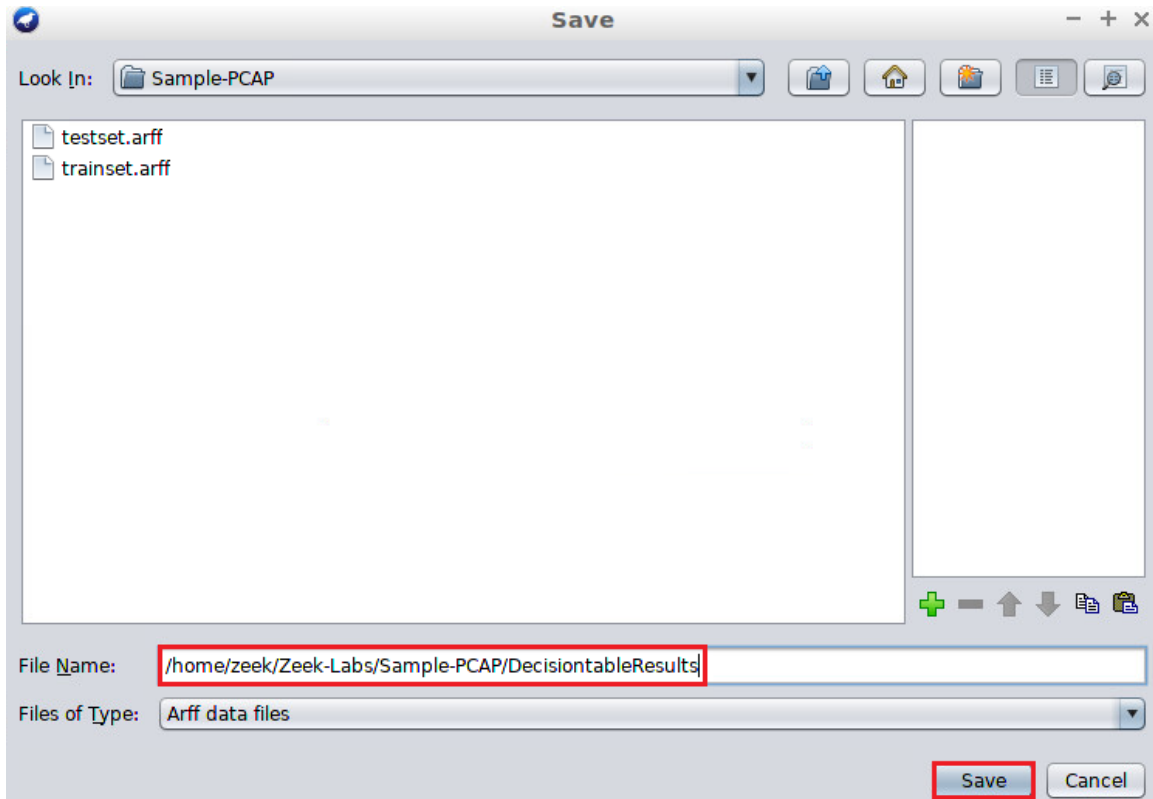


Step 3. Click the *Save* button.



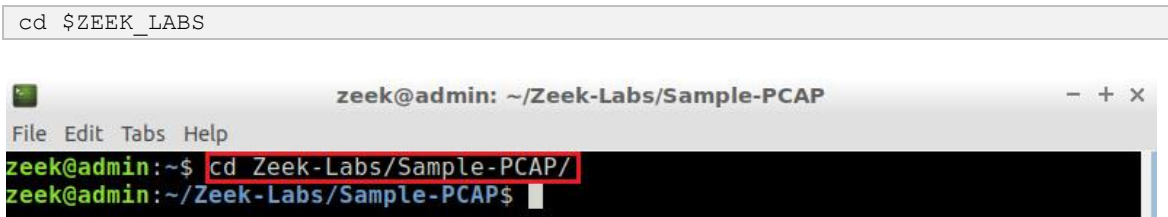
Step 3. Navigate to the Lab workspace directory and save the *DecisionTableResults*. Alternatively, use the GUI to navigate to the lab workspace directory to select the file. Use the *Save* button to save the new *DecisionTableResults* file into Weka.

```
/home/zeek/Zeek-Labs/Sample-PCAP/DecisionTsuableResults
```

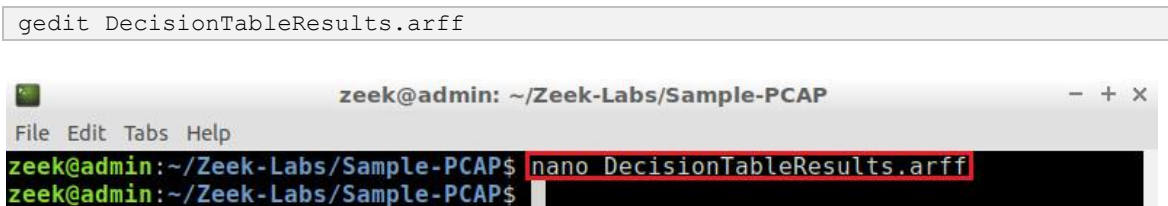


Step 4. Close all of the Weka tabs with the orange x on the top right corner of each panel.

Step 5. Return to the Terminal and navigate to the lab workspace directory.



Step 5. Using a text editor, view the *DecisionTableResults.arff* file.



The file will be opened, and each data object will contain a new classification label.

```

zeek@admin: ~/Zeek-Labs/Sample-PCAP
File Edit Tabs Help
GNU nano 2.9.3 DecisionTableResults.arff
@relation networktraffic-weka.filters.unsupervised.attribute.NumericToNominals
@attribute sourceip {10022,100215,19216813,65552560,172162551,655557251,19216$
@attribute destip {10022,10023,10112,100215,1002255,1721601,2454514,6443557,6$
@attribute sourceport numeric
@attribute destport numeric
@attribute protocol {tcp,udp,icmp}
@attribute duration numeric
@attribute class {1,0}

@data
19216813,19216822,49526,1755,tcp,3,1
172162551,18912644128,50983,3192,udp,259354,0
172162551,2049163158,10630,80,tcp,150006,0
19216813,19216822,49526,9002,tcp,3,1
19216813,19216822,49526,15660,tcp,4,1
19216813,19216822,49526,2607,tcp,4,1
172162551,255255255255,68,67,udp,?,0
1921683131,65549575,56332,80,tcp,6663279,0

```

Traffic found in the first row of data was labeled with a 1, as malicious traffic. Traffic found in the second row of data was labeled with a 0, as benign traffic.

Concluding this lab, we have introduced the capabilities of implementing a machine learning classifier to detect specific anomalies or events. Multiple classifiers can be used for training network security classifiers, and the features within each training dataset can have a profound impact on the classifier's accuracy. By removing, modifying or adding new features you can test the accuracy of a classifier. In this lab, we generated a *Decision Table* that was capable of labeling malicious and benign traffic.

References

1. "Attribute-relation file format", The university of waikato, [Online], Available: <https://www.cs.waikato.ac.nz/~ml/weka/arff.html>