UTSA

The University of Texas at San Antonio™

The Cyber Center for Security and Analytics

UNIVERSITY OF SOUTH CAROLINA

ZEEK INSTRUSION 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

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

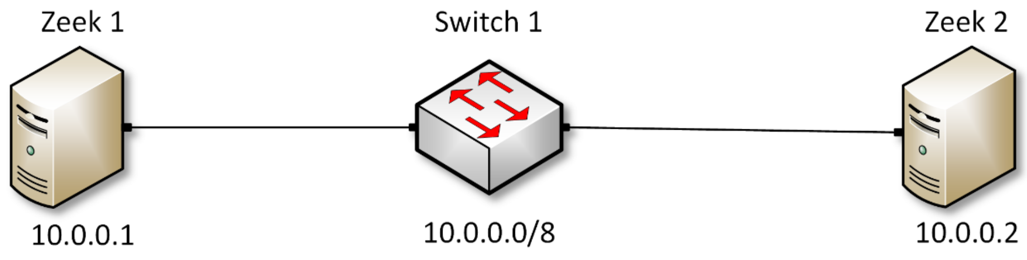


Figure 1. Lab topology.

Lab settings

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

Table 1. Credentials to access the Client machine

Device	Account	Password
Client	admin	password

Table 2. Shell variables and their corresponding absolute paths.

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

Lab roadmap

This lab is organized as follows:

1. Section 1: 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.

```
event zeek_init(){
    /* code */
}
event zeek_done(){
    /* code */
}
event tcp_packet(){
    /* code */
}
event udp_request(){
    /* code */
}
event udp_reply(){
    /* code */
}
```

- `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.

```
module ZeekScript;

export {
    /* Append a new Log stream */
    /* Define a new data type to format new Log stream */
}
```

- `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.

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

- `connection_established` event: activated when a host makes a connection to a receiver.

- `Log::create_stream`: creates a new log stream, with a name, format structure and path.
- `Log::write`: writes included data to the specified log stream.

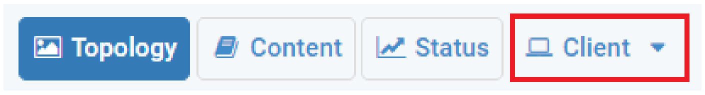
Additional log stream commands are explained in detail in Zeek's official documentation.

2 Log file analysis using Zeek scripts

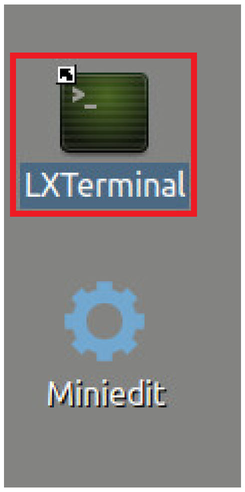
With Zeek's event-driven scripting language, we can create specific event-based filters to be applied during packet capture analysis. This section shows example scripts for network analysis.

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.

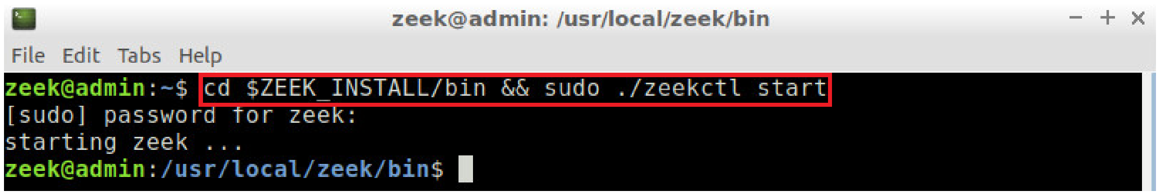


Step 2. The *Client* machine will now open, and the desktop will be displayed. On the left side of the screen, click on the LXTerminal icon as shown below.



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

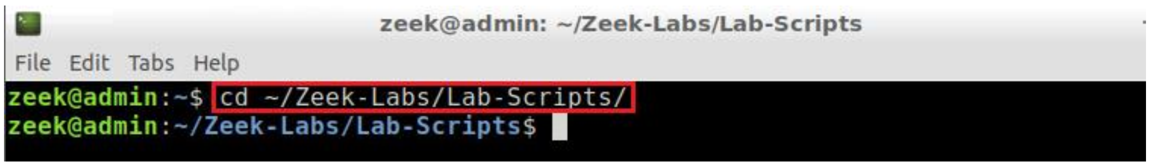
A new instance of Zeek is now active, and we are ready to proceed to the next section of the lab.

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/
```

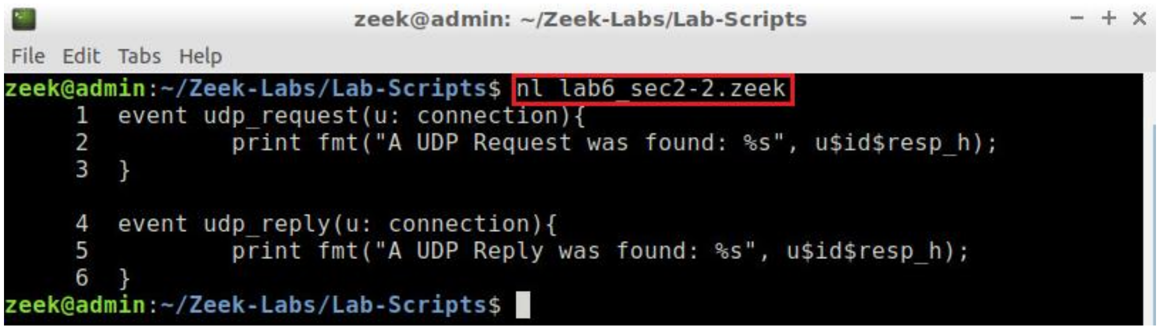


```
zeek@admin:~$ cd ~/Zeek-Labs/Lab-Scripts/
zeek@admin:~/Zeek-Labs/Lab-Scripts$
```

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
```



```
zeek@admin:~/Zeek-Labs/Lab-Scripts$ nl lab6_sec2-2.zeek
     1  event udp_request(u: connection){
     2          print fmt("A UDP Request was found: %s", u$id$resp_h);
     3  }

     4  event udp_reply(u: connection){
     5          print fmt("A UDP Reply was found: %s", u$id$resp_h);
     6  }
zeek@admin:~/Zeek-Labs/Lab-Scripts$
```

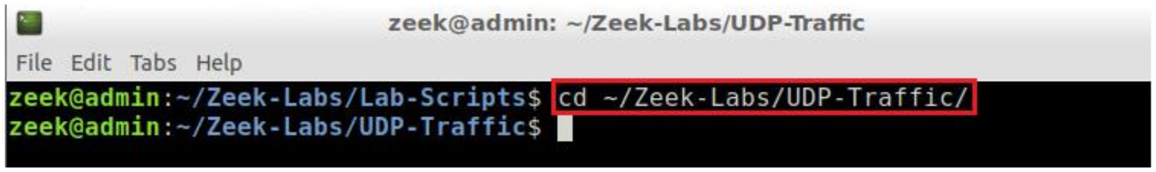
The script is explained as follows. Each number represents the respective line number:

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. `u$id$resp_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.
5. Prints the specified string. `u$id$resp_h` retrieves the destination IP address from the UDP packet.
6. End of the `udp_reply` event.

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

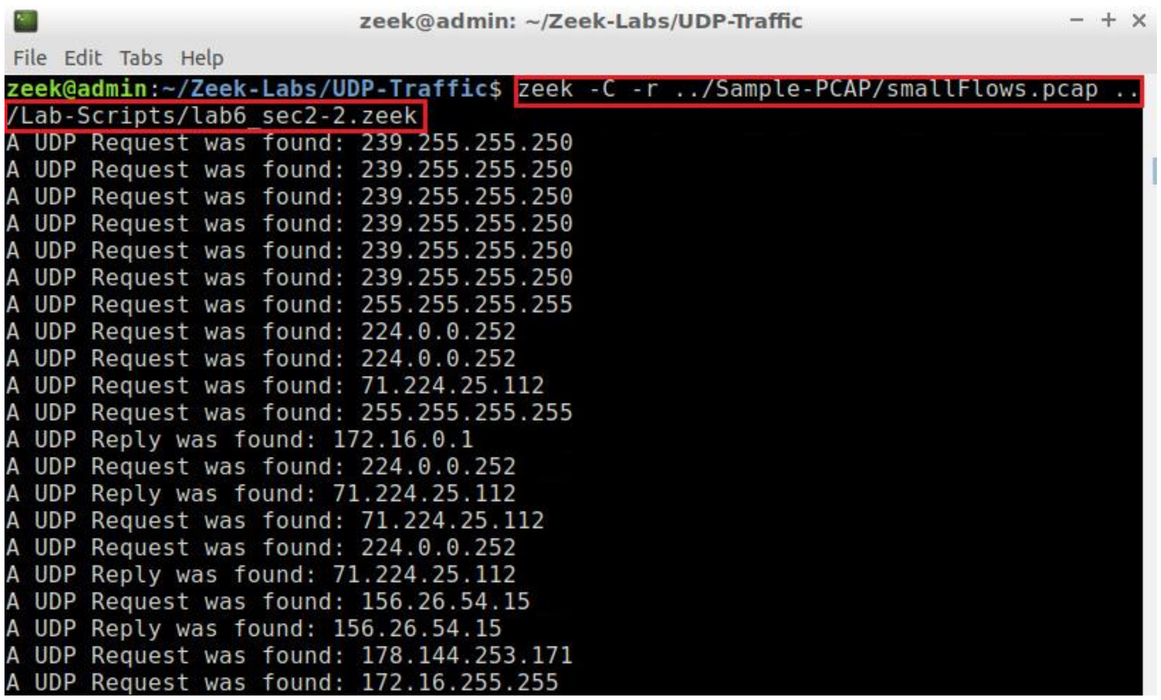
```
cd Zeek-Labs/UDP-Traffic/
```



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



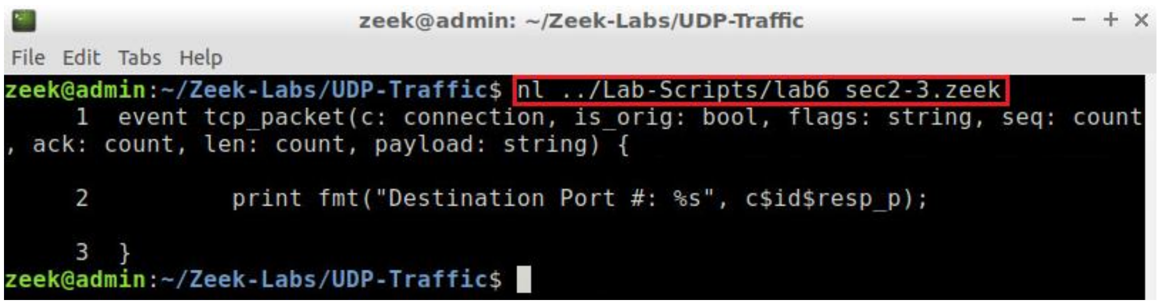
```
zeek@admin:~/Zeek-Labs/UDP-Traffic$ zeek -C -r ../Sample-PCAP/smallFlows.pcap ..
/Lab-Scripts/lab6_sec2-2.zeek
A UDP Request was found: 239.255.255.250
A UDP Request was found: 239.255.255.250
A UDP Request was found: 239.255.255.250
A UDP Request was found: 239.255.255.250
A UDP Request was found: 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
```



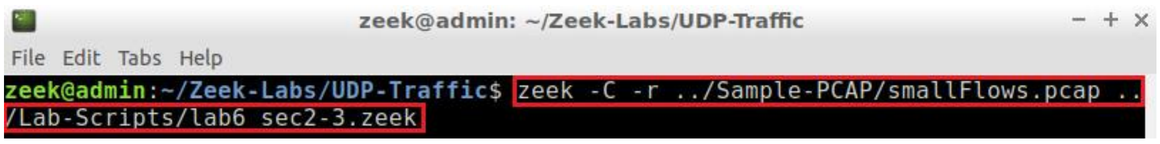
```
zeek@admin:~/Zeek-Labs/UDP-Traffic$ nl ../Lab-Scripts/lab6_sec2-3.zeek
     1  event tcp_packet(c: connection, is_orig: bool, flags: string, seq: count
, ack: count, len: count, payload: string) {

     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. `u$id$resp_h` retrieves the destination IP address from the TCP packet.
3. End of the `tcp_packet` event.

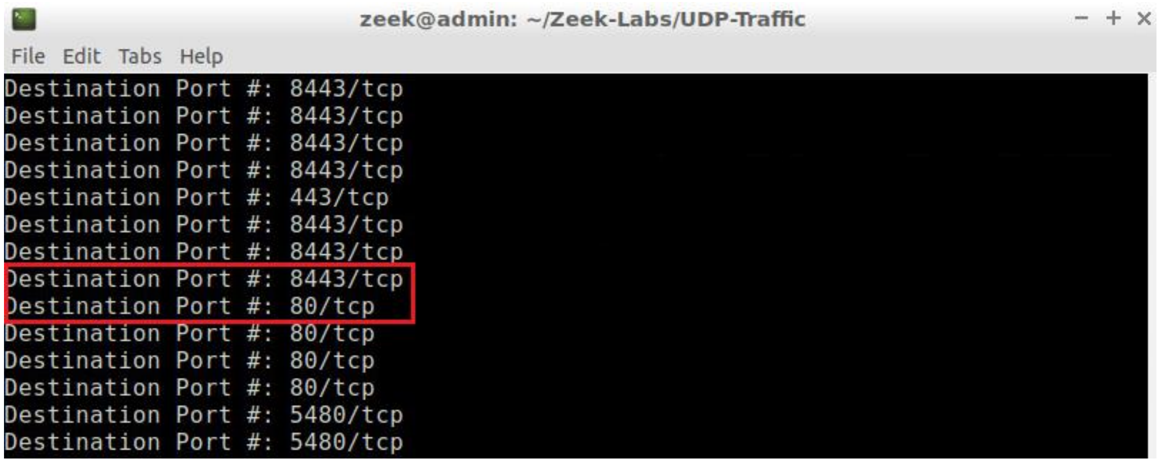
Step 2. Process a packet capture file using the Zeek script. It is possible to use the `tab` key to autocomplete the longer paths.

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



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

The following output is produced:



```
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.

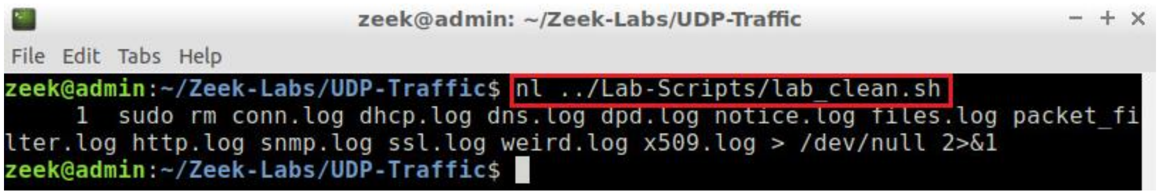
3 Modifying Zeek log streams

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
```

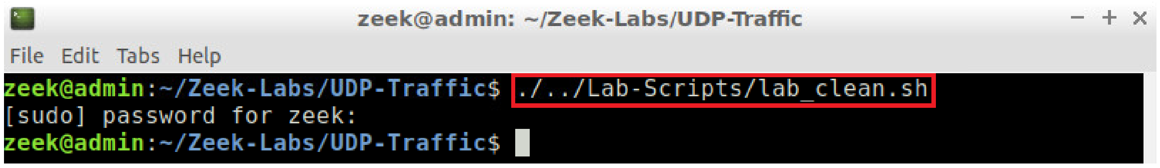


```
zeek@admin:~/Zeek-Labs/UDP-Traffic$ nl ../Lab-Scripts/lab_clean.sh
     1  sudo rm conn.log dhcp.log dns.log dpd.log notice.log files.log packet_fi
lter.log http.log snmp.log ssl.log weird.log x509.log > /dev/null 2>&1
zeek@admin:~/Zeek-Labs/UDP-Traffic$ 
```

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

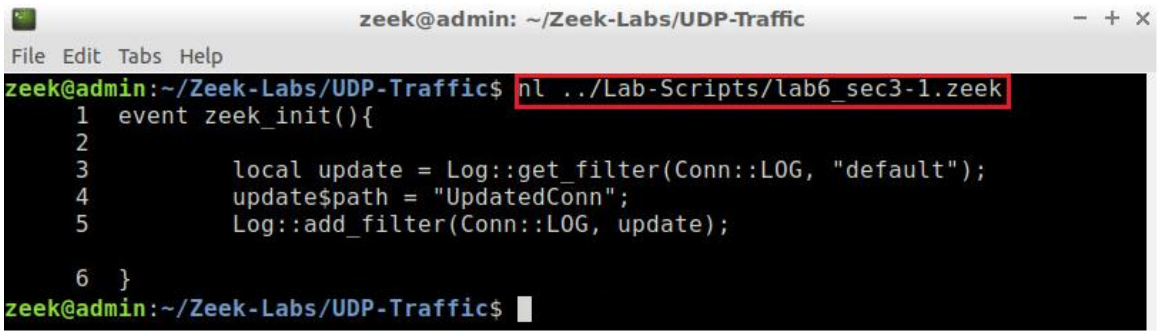
With the workspace directory cleared, we can move to the next section.

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
```



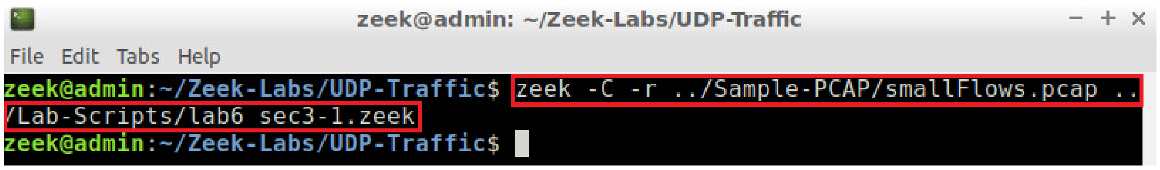
```
zeek@admin:~/Zeek-Labs/UDP-Traffic$ nl ../Lab-Scripts/lab6_sec3-1.zeek
     1  event zeek_init(){
     2
     3          local update = Log::get_filter(Conn::LOG, "default");
     4          update$path = "UpdatedConn";
     5          Log::add_filter(Conn::LOG, update);

     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.

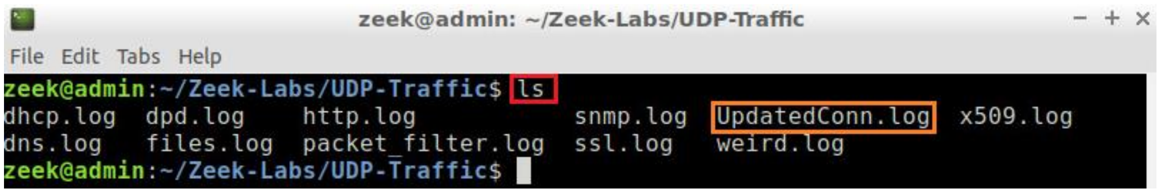
```
zeek -C -r ../Sample-PCAP/smallFlows.pcap ../Lab-Scripts/lab6_sec3-1.zeek
```



```
zeek@admin:~/Zeek-Labs/UDP-Traffic$ 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$ 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$
```

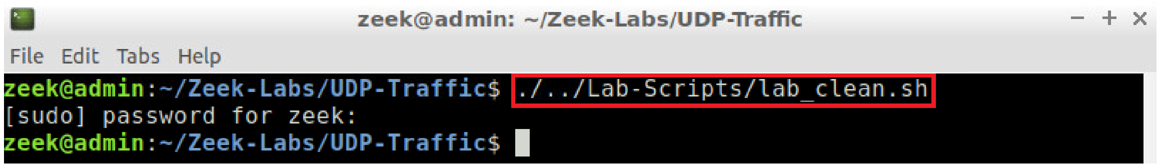
Note the *UpdatedConn.log*, highlighted by the orange box. Since we did not change any formatting, it is an exact replica of the original *conn.log* file.

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.

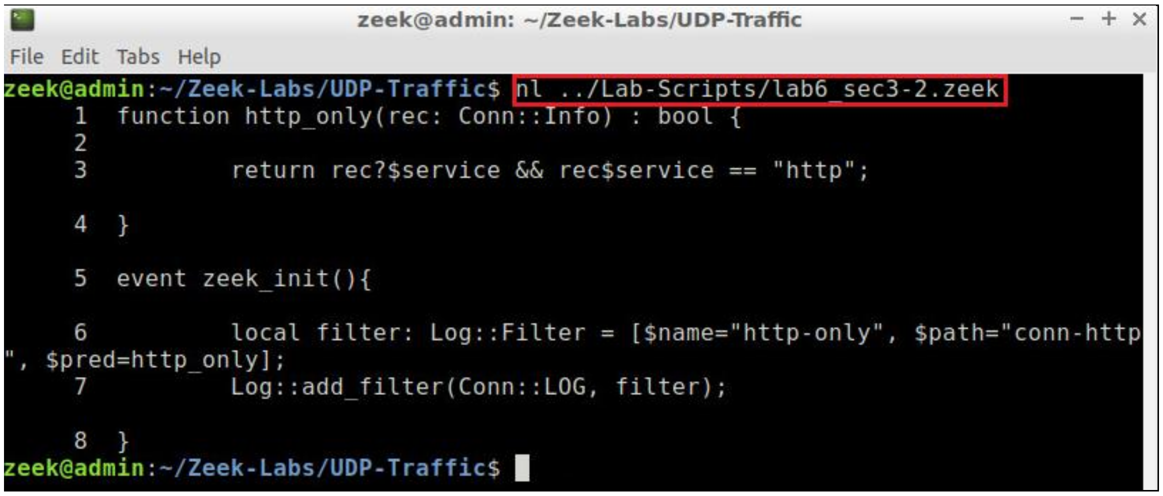
```
./../Lab-Scripts/lab_clean.sh
```



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

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

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



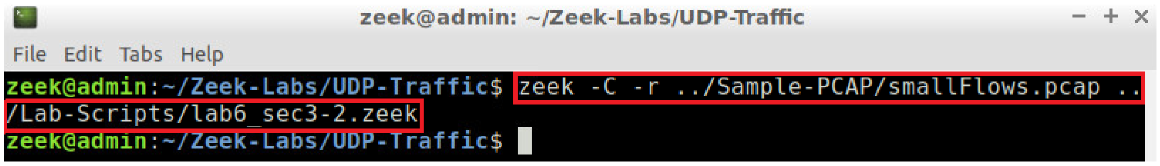
```
zeek@admin:~/Zeek-Labs/UDP-Traffic$ nl ../Lab-Scripts/lab6_sec3-2.zeek
     1  function http_only(rec: Conn::Info) : bool {
     2
     3          return rec?$service && rec$service == "http";

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

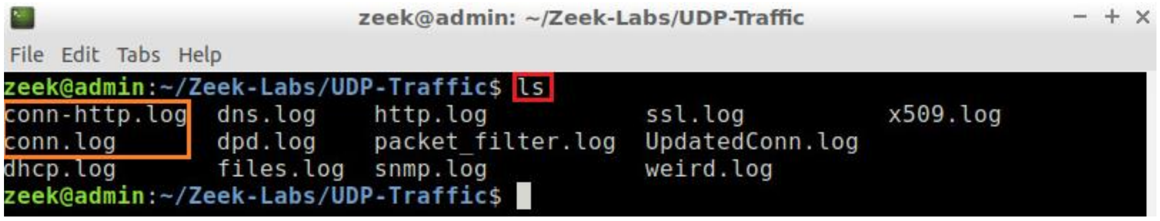
```
zeek -C -r ../Sample-PCAP/ smallFlows.pcap ../Lab-Scripts/lab6_sec3-2.zeek
```



```
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 4. 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$
```

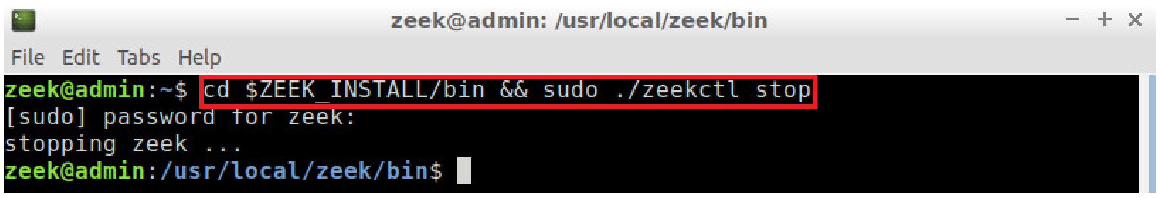
Note the *conn-http.log* file in the first column. This file will have the same formatting as the *conn.log* file; however, it will only contain HTTP traffic. These files are highlighted by the orange box in the proceeding image.

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:~$ 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 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.