The University of Texas at San Antonio™

The Cyber Center for Security and Analytics

UNIVERSITY OF SOUTH CAROLINA

ZEEK INSTRUSION 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

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 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.

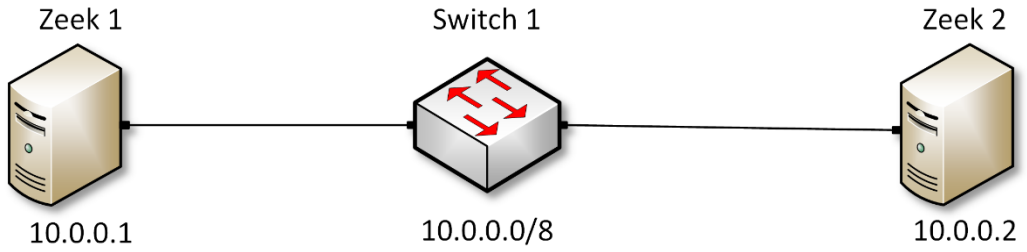


Figure 1. Lab topology.

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 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 experience[1]. 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 problems[2].

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.

Open ▾ 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.

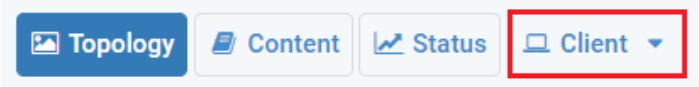
2 Aggregating network capture datasets

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

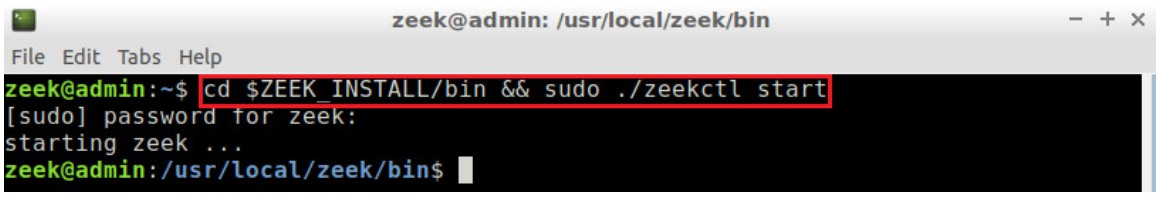
Step 1. From the top of the screen, click on the *Client* button as shown below to enter the *Client* machine.



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`.

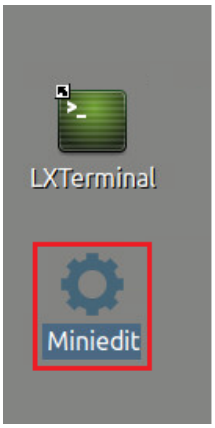
```
cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
```



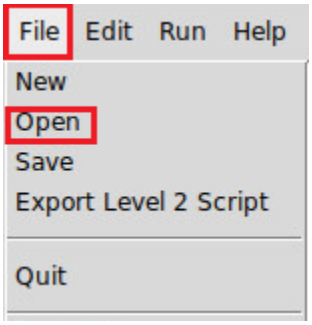
A new instance of Zeek is now active, and we are ready to proceed to the next section of the lab.

2.2 Launching Mininet

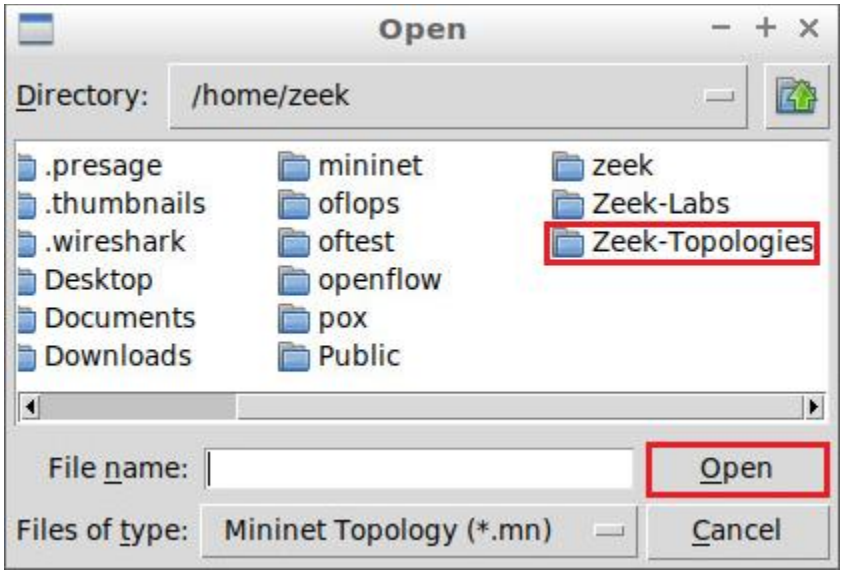
Step 1. From the *Client* machine's desktop, on the left side of the screen, click on the MiniEdit icon as shown below. When prompted for a password, type `password` and hit `Enter`. The MiniEdit editor will now launch.



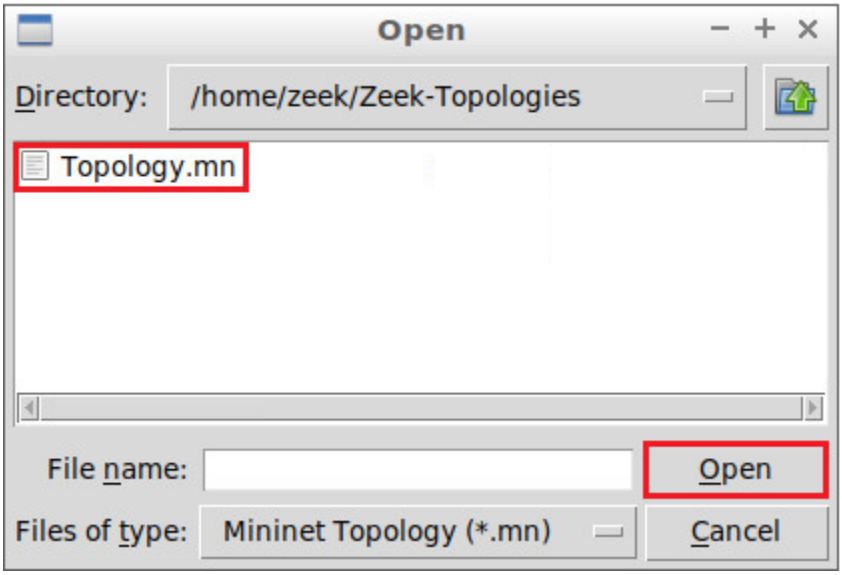
Step 2. The MiniEdit editor will now launch and allow for the creation of new, virtualized lab topologies. Load the correct topology by clicking the `Open` button within the `File` tab on the top left of the MiniEdit editor.



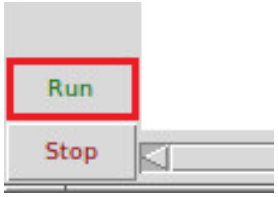
Step 3. Navigate to the Zeek-Topologies directory by scrolling to the right of the active directories and double clicking the Zeek-Topolgies icon, or by clicking the `Open` button.



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

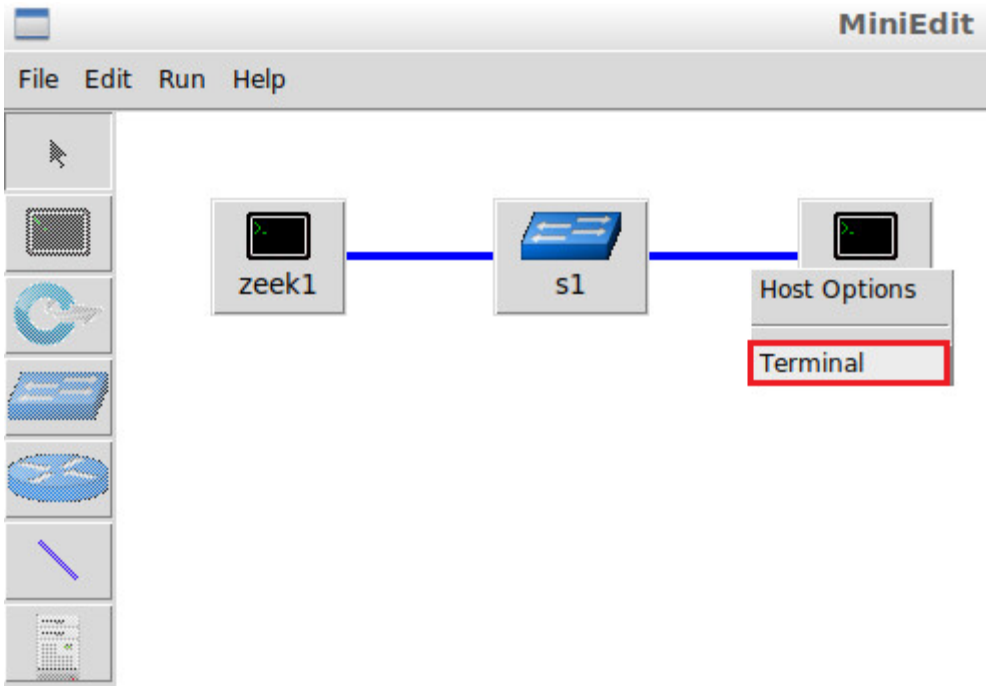


Step 5. To begin running the virtual machines, navigate to the `Run` button, found on the bottom left of the Miniedit editor, and select the `Run` button, as seen in the image below.



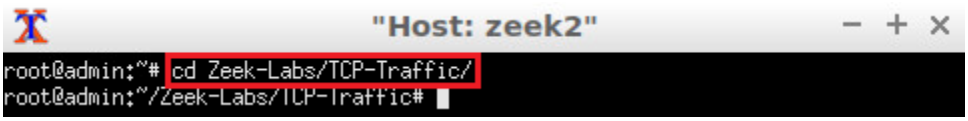
2.3 Setting up the zeek2 virtual machine for live network capture

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



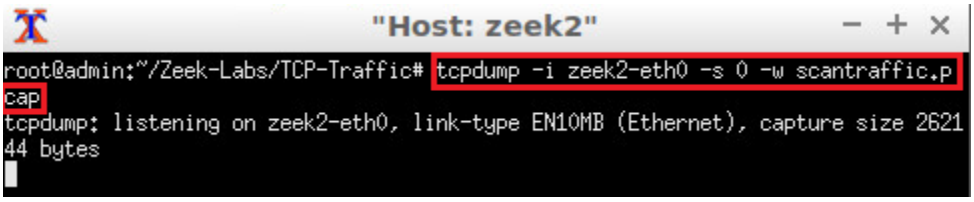
Step 2. Navigate to the TCP-Traffic directory.

```
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
```

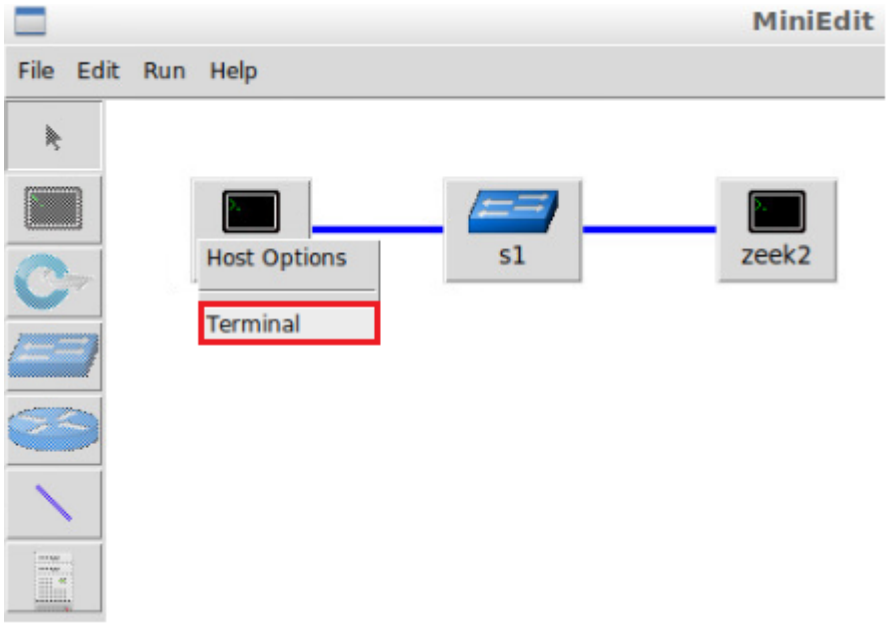


The *zeek2* virtual machine is now ready to begin collecting live network traffic. Next, we will use the *zeek1* machine to generate scan-based network traffic.

2.4 Using the zeek1 virtual machine for network scanning activities

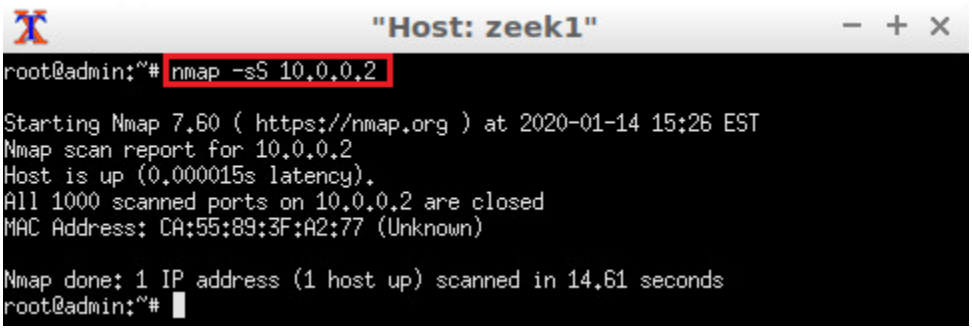
In this section we use the `nmap` software to generate TCP-based scan traffic.

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.

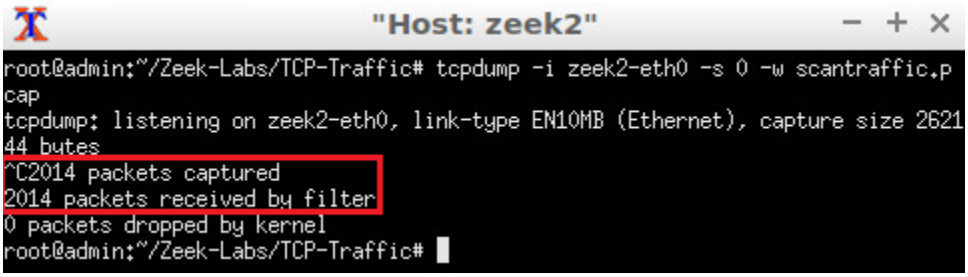
```
nmap -sS 10.0.0.2
```



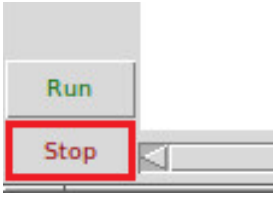
2.4.1 Terminating live network capture

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 we 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 `x` 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.

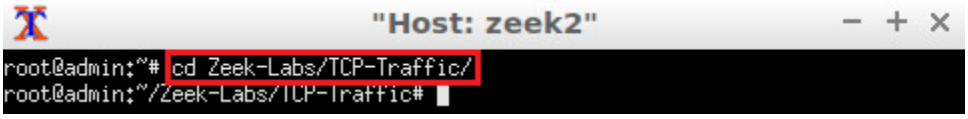
In a real-time environment, at this stage you may include anomaly-specific scripts. Once an anomaly has been processed by Zeek, the resulting log files will need to be reformatted.

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 1. Navigate to the TCP-Traffic directory.

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



Step 2. Process the *scantraffic.pcap* file.

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

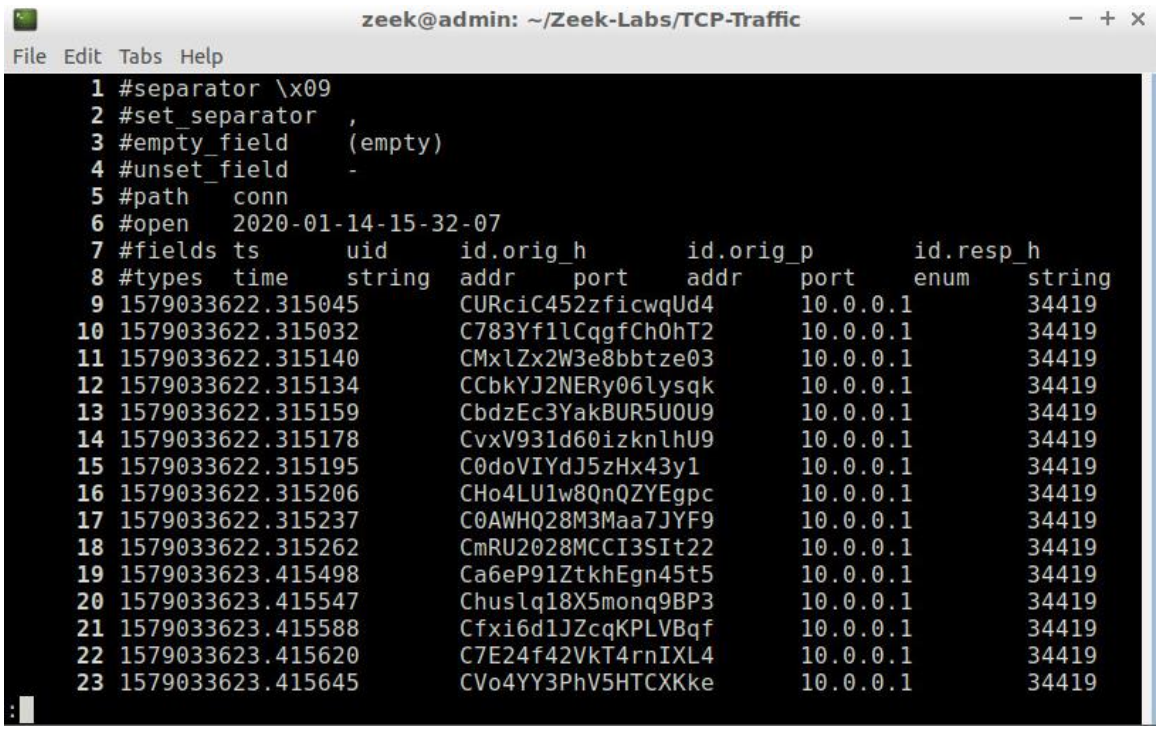
Step 3. Display the contents of the *conn.log* file.

```
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.

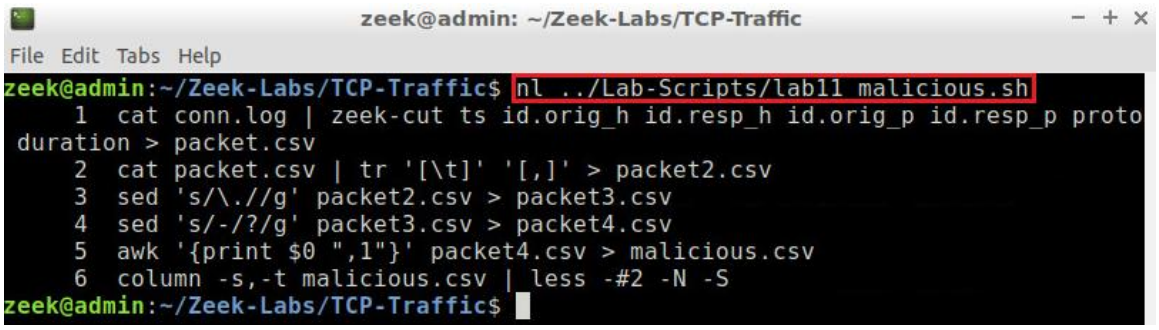


We can see in the previous image that the *conn.log* file is nowhere near the *.arff* file format. We will need to remove the Zeek padding, column names, change the tab delimiter and remove excess column features.

Press the `q` key on your keyboard to exit and return to the Terminal.

Step 4. Display the contents of *lab11_malicious.sh* shell script using the `nl` command.

```
nl ../Lab-Scripts/lab11_malicious.sh
```



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

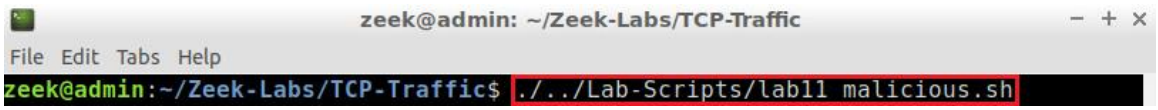
1. Using the `cat` utility, the contents of the *conn.log* file will be passed into the `zeek-cut` utility to remove the log file header and only include the specified columns. The output of the `zeek-cut` utility will be saved to a new file named *packet.csv.* The feature columns we will be using to 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.

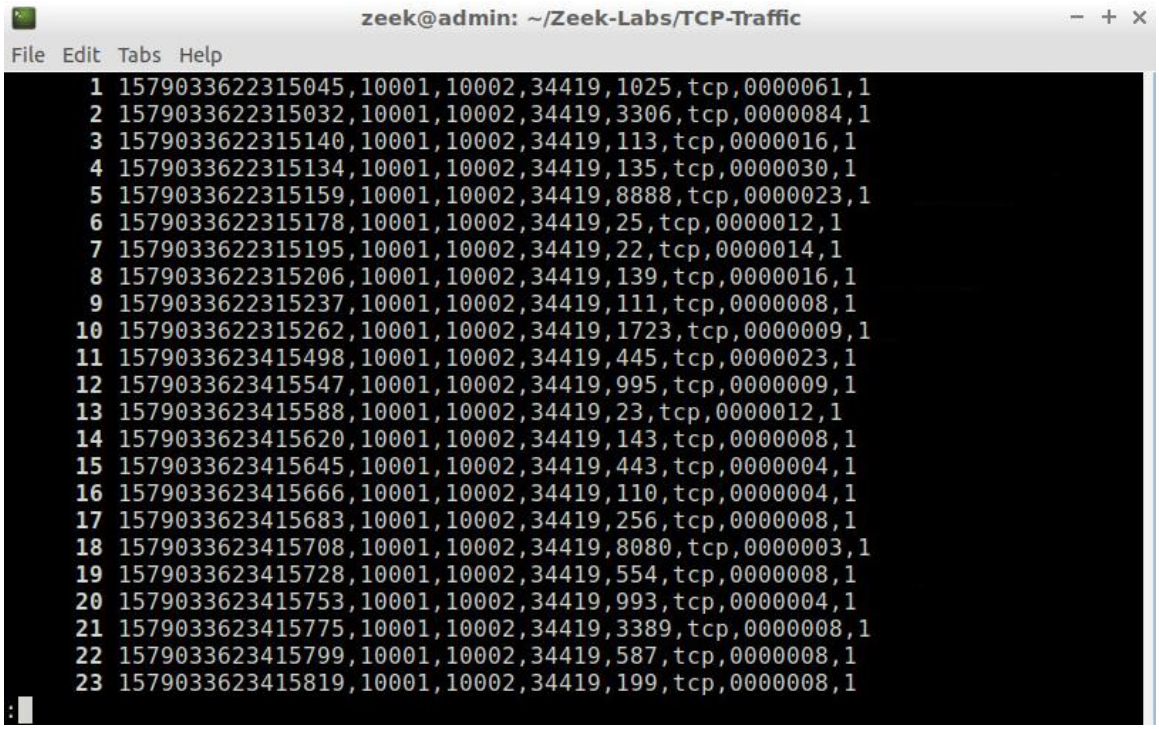
2. Using the `cat` utility, the contents of the *packet.csv* file will be passed into the `tr` utility. The `tr` utility will replace the *packet.csv* file's tab-delimited structure with a comma-delimited structure, and the output will be saved to a new file named *packet2.csv*.
3. Using the `sed` utility, 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*.
4. 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*.
5. 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*.
6. The file contents of *malicious.csv* will be displayed. This command is introduced in the Step 1 of this subsection.

Step 5. 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.



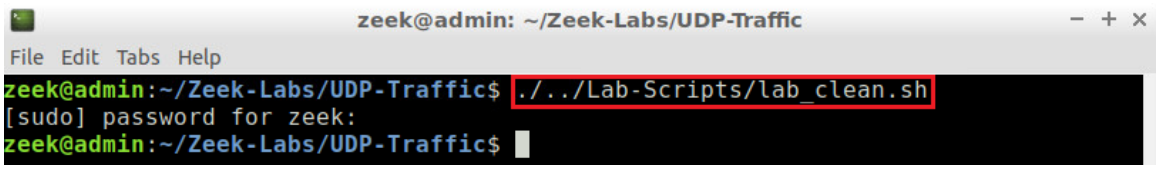
We can see from the above image that the *malicious.csv* file is now properly formatted to fit in the `@DATA` section of an ARFF file. Each row contains an equal number of comma-delimited columns with only numeric characters.

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

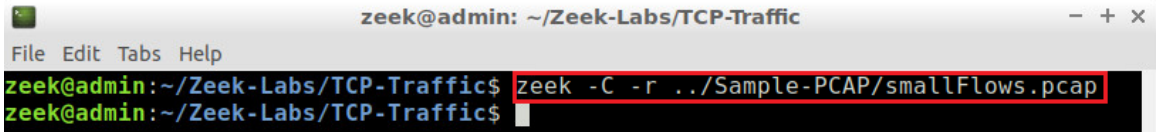
```
./../Lab-Scripts/lab_clean.sh
```



3.2 Preprocessing of the benign dataset

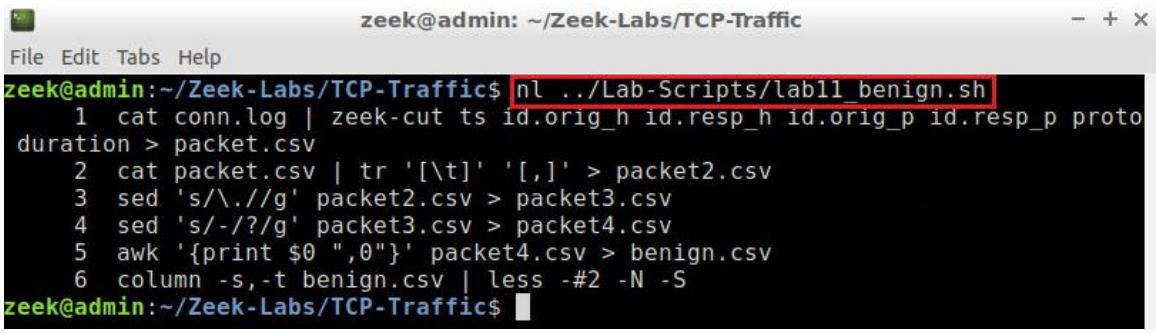
Step 1. Process the *smallFlows.pcap* file using the `zeek -r` command.

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



Step 2. Display the contents *lab11_benign.sh* shell script using the `nl` command.

```
nl ../Lab-Scripts/lab11_benign.sh
```

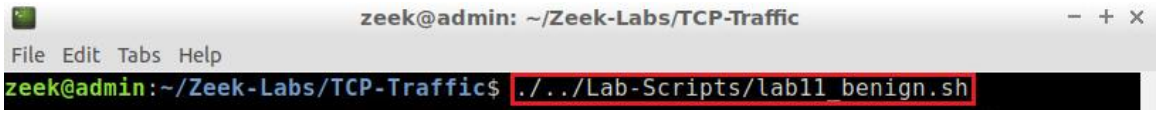


With the exception of Line 5, the script is exactly the same as the one explained in Step 3 of the previous section.

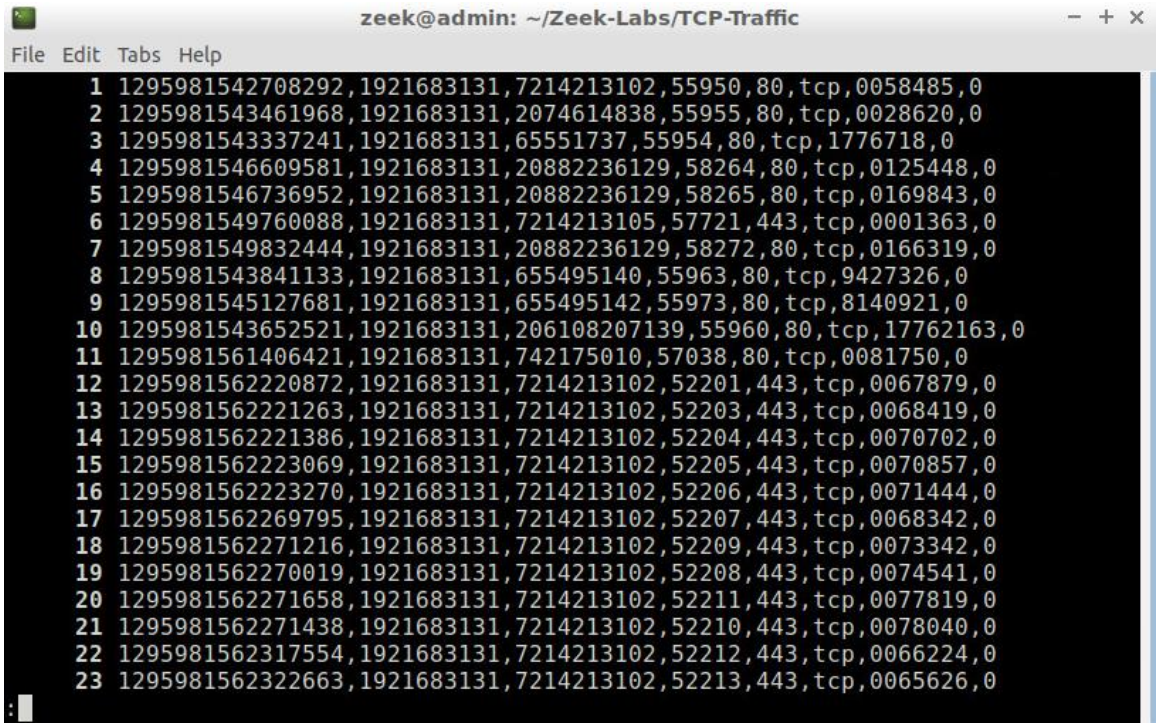
Line 5 has been modified to append `,0` to the end of each row. This value represents the benign class label. The output will be saved to a new file named *benign.csv*.

Step 3. Execute the *lab11_benign.sh* shell script.

```
./../Lab-Scripts/lab1_benign.sh
```



After executing all commands in the script, the *benign.csv* file contents will be displayed on the Terminal as shown in the figure below.



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

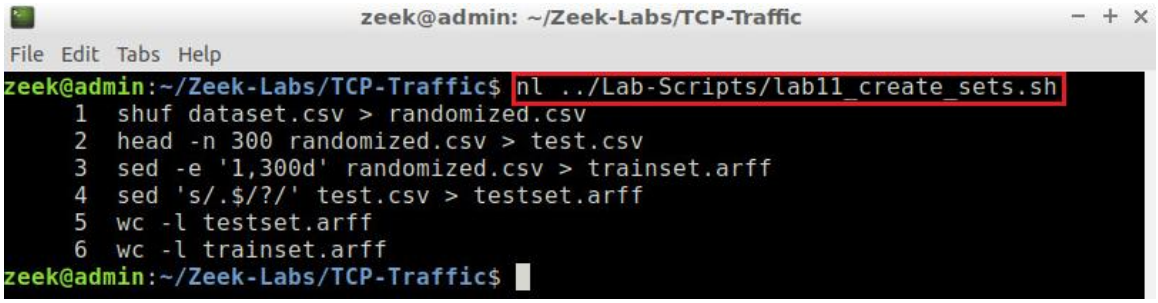
Step 1. Combine the *malicious.csv* and *benign.csv* files into the *dataset.csv* file.

```
cat malicious.csv benign.csv > dataset.csv
```

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
```

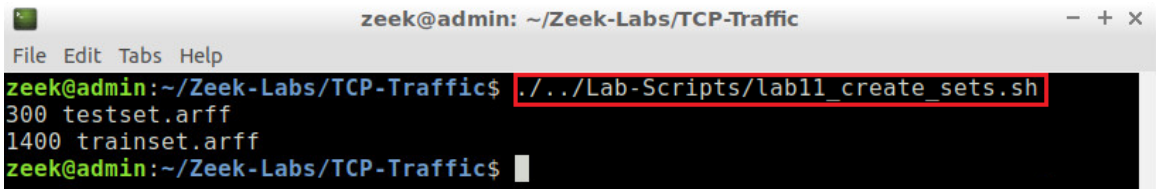


The script is explained as follows. Each number represents the respective line number:

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
```



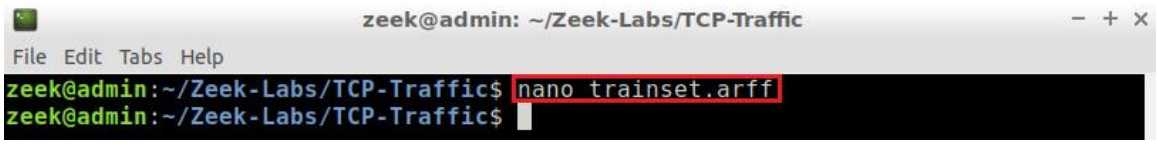
The figure above shows the line count of the *testset.arff* and *trainset.arff* files. The *testset.arff* file contains 300 rows while the *trainset.arff* file contains 1400 rows. The *trainset.arff* file size may be variable due to the number of packets generated during the original TCP SYN scans; however, 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
```

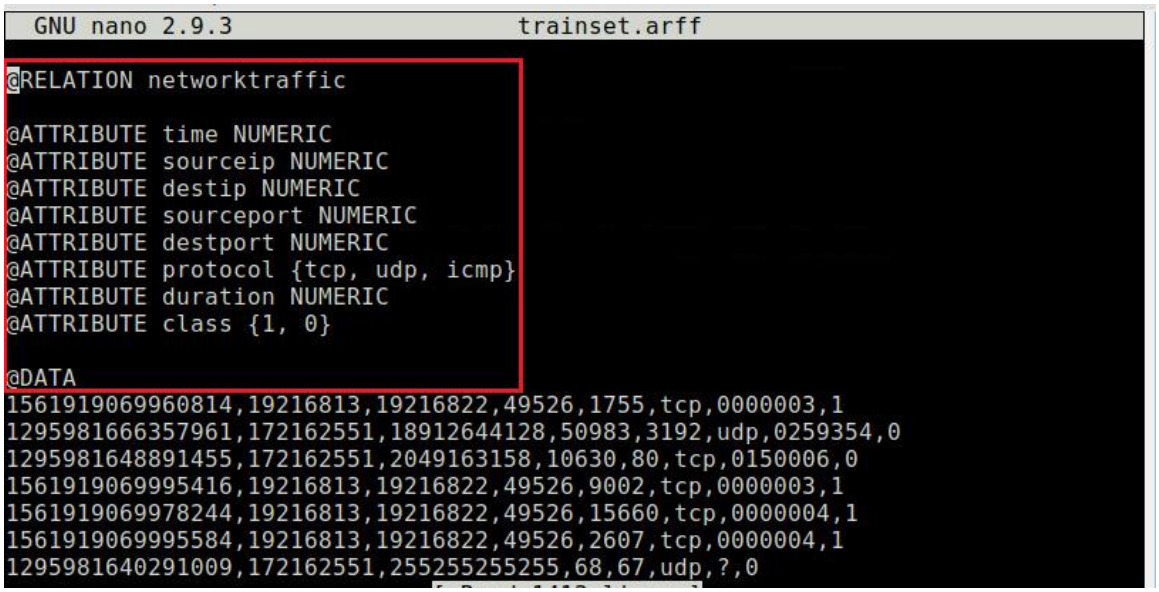


Step 2. Prepend the following headers to the *trainset.arff* file. To type capital letters, it is recommended to hold the *Shift* key while typing rather than using the *Caps* key.

```
@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
```

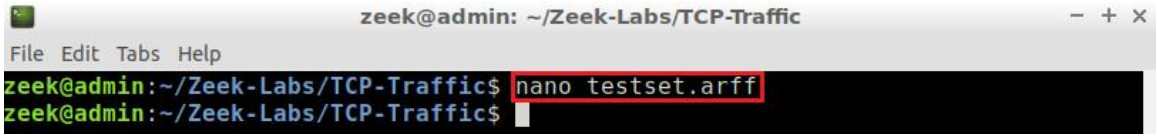


The input training dataset is now a properly formatted *.arff* file and can be input into a machine learning algorithm to train a classifier.

Press `Ctrl+o` and Enter to save the file, then `Ctrl+x` to exit out the nano editor.

Step 3. Using the `nano` text editor, open the *testset.arff* file for editing.

```
nano testset.arff
```



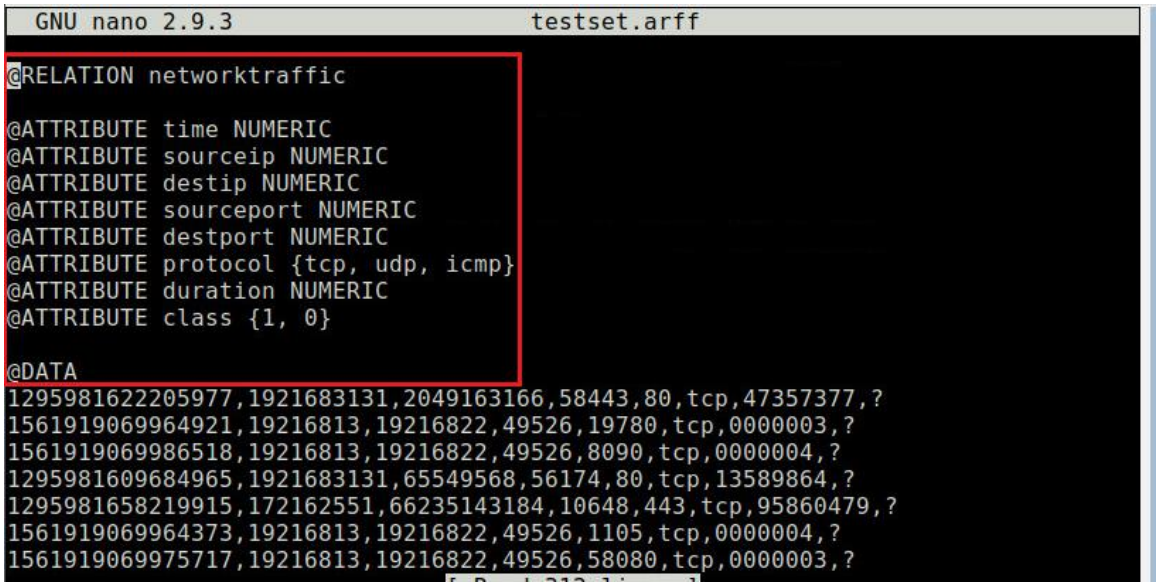
Step 4. Prepend the following headers to the *testset.arff* file. To type capital letters, it is recommended to hold the *Shift* key while typing rather than using the *Caps* key.

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
```



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.

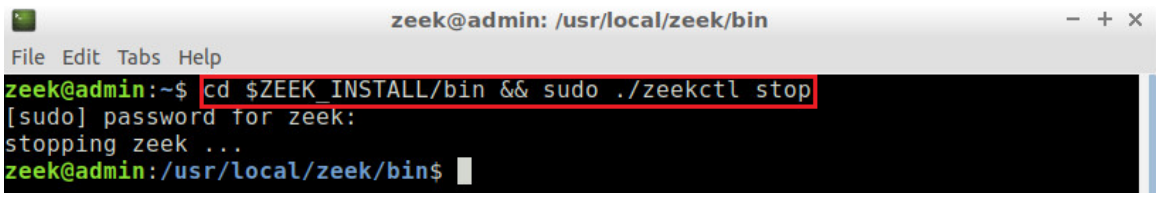
Press `Ctrl+o` and Enter to save the file, then `Ctrl+x` to exit out the nano editor.

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
```



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