

Configuration Manual

MSc Research Project
Cybersecurity

Mudiaga Agbroko
Student ID: x23207485

School of Computing
National College of Ireland

Supervisor: Kamil Mahajan

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Mudiaga Agbroko
Student ID: X23207485
Programme: Cybersecurity **Year:** 2024
Module: Research Project
Lecturer: Kamil Mahajan
Submission Due Date: 12/12/2024
Project Title: Utilising Artificial Intelligence in Enhancing Zero-Day Attacks Detection
Word Count: 1244 **Page Count:** 24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Mudiaga Agbroko

Date: 05/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Mudiaga Agbroko
Student ID: x23207485

1 Introduction

This document outlines the hardware and software configuration employed for the implementation of the research project, “Utilising Artificial Intelligence in Enhancing Zero-Day Attacks”. The steps taken in the project’s implementation are also detailed as shown in Figure 1.

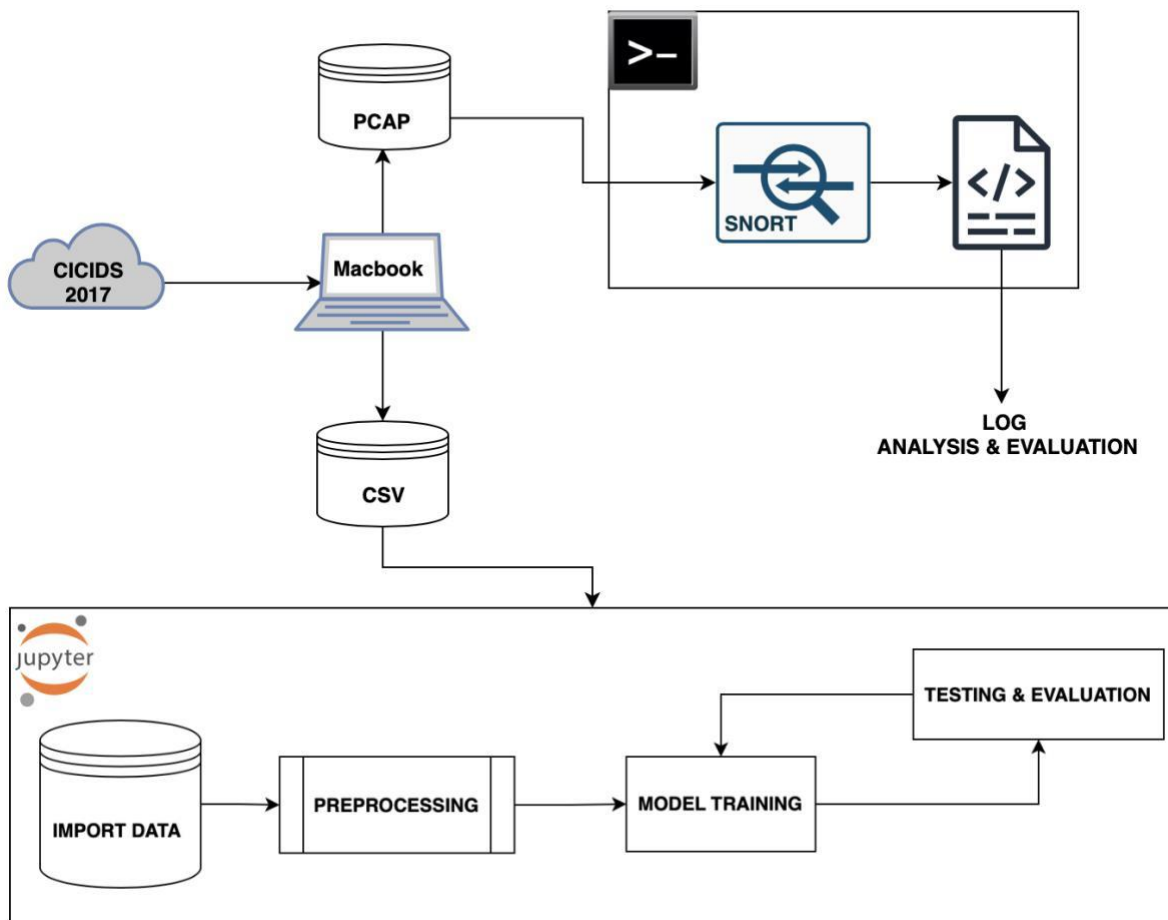


Figure 1: The architecture of the research project’s implementation.

2 System Specification

2.1 Hardware Specification

- PC: MacBook Pro 2017
- Processor: 2.3 GHz Dual-Core Intel Core i5
- Graphics: Intel Iris Plus Graphics 640
- Memory: 8 GB
- Storage: 256 GB SSD






2.2 Software Specification

- Terminal 2.3
- Homebrew 4.4.10
- Wireshark 4.4.2
- Snort 3.6.0
- Microsoft Excel 16.91
- Jupyter Notebook 7.3.1

3 Dataset

The CIC-IDS2017 dataset (Sharafaldin, Lashkari, and Ghorbani, 2018) consisting of benign and cyber-attack traffic was employed in the research project. The Monday subset of the dataset was excluded because it contained only benign traffic, focusing on just Tuesday, Wednesday, Thursday, and Friday datasets. For the research project, the PCAP and machine learning CSV files were downloaded manually from the dataset website as shown in Figure 2 and Figure 3. The PCAP files were used on snort while the machine-learning CSV files were used in training and testing the machine-learning models.

- Dataset Download link: <http://205.174.165.80/CICDataset/CIC-IDS-2017/Dataset/CIC-IDS-2017/>

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 GeneratedLabelledFlows.md5	2024-02-01 16:27	61	
 GeneratedLabelledFlows.zip	2024-02-01 16:28	271M	
 MachineLearningCSV.md5	2024-02-01 16:28	57	
 MachineLearningCSV.zip	2024-02-01 16:28	224M	

Apache/2.4.41 (Ubuntu) Server at 205.174.165.80 Port 80

Figure 2: Download links of the CIC-IDS2017 dataset CSV files.

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 Friday-WorkingHours.md5	2024-02-01 16:28	59	
 Friday-WorkingHours.pcap	2024-02-01 16:36	8.2G	
 Monday-WorkingHours.md5	2024-02-01 16:36	59	
 Monday-WorkingHours.pcap	2024-02-01 16:48	10G	
 Thursday-WorkingHours.md5	2024-02-01 16:48	61	
 Thursday-WorkingHours.pcap	2024-02-01 16:57	7.7G	
 Tuesday-WorkingHours.md5	2024-02-01 16:57	60	
 Tuesday-WorkingHours.pcap	2024-02-01 17:08	10G	
 Wednesday-workingHours.md5	2024-02-01 17:08	62	
 Wednesday-workingHours.pcap	2024-02-01 17:20	12G	

Apache/2.4.41 (Ubuntu) Server at 205.174.165.80 Port 80

Figure 3: Download links of the CIC-IDS2017 dataset PCAP files.

4 Research Workflow

4.1 Merging the CIC-IDS2017 PCAP Files

Upon downloading the Tuesday, Wednesday, Thursday, and Friday PCAP files of the CIC-IDS2017 dataset. The next step was to merge the PCAP files and process them using Snort. Wireshark's mergecap with the -w flag was employed in combining the PCAP files into a single PCAP file. To get Wireshark, the Homebrew package manager was installed via the terminal then Wireshark was installed. Upon installing Wireshark, the PCAP files were combined on the terminal.

```
kingmudis@MA-Mac ~ % /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
[==> Checking for `sudo` access (which may request your password)... ]
Password:
==> This script will install:
/usr/local/bin/brew
/usr/local/share/doc/homebrew
/usr/local/share/man/man1/brew.1
/usr/local/share/zsh/site-functions/_brew
/usr/local/etc/bash_completion.d/brew
/usr/local/Homebrew
==> The following new directories will be created:
/usr/local/sbin
/usr/local/opt
/usr/local/var/homebrew/linked
/usr/local/Cellar
/usr/local/Caskroom
/usr/local/Frameworks
```

Figure 4: Installing Homebrew package manager through the terminal.


```
kingmudis@MA-Mac ~ % brew install wireshark
Warning: Treating wireshark as a formula. For the cask, use homebrew/cask/
wireshark or specify the '--cask' flag. To silence this message, use the
'--formula' flag.
==> Downloading https://ghcr.io/v2/homebrew/core/wireshark/manifests/4.4.2
#####
100.0%
==> Fetching dependencies for wireshark: c-ares, pcre2, python-packaging,
mpdecimal, ca-certificates, openssl@3, readline, sqlite, xz, python@3.13,
libunistring, gettext, glib, gmp, libidn2, libtasn1, nettle, p11-kit, libe
vent, libnghttp2, unbound, gnutls, libpgp-error, libgcrypt, libmaxminddb,
libnghttp3, libsmi, libssh, lua and speexdsp
```

Figure 5: Installing Wireshark using the Homebrew package manager.

```
kingmudis@MA-Mac ~ % mergecap -w ~/Downloads/pcap/cic_ids2017.pcap ~/Downl
oads/pcap/Tuesday-WorkingHours.pcap ~/Downloads/pcap/Wednesday-workingHour
s.pcap ~/Downloads/pcap/Thursday-WorkingHours.pcap ~/Downloads/pcap/Friday
-WorkingHours.pcap
kingmudis@MA-Mac ~ %
```

Figure 6: Installing Wireshark using the Homebrew package manager.

4.2 Installing and Configuring Snort IDS/IPS

The Snort open-source IDS/IPS was employed in processing the CIC-IDS2017 PCAP file. The first step was to download the Snort software using the Homebrew package manager on the terminal. The next step is to make Snort functional by manually updating the /dev/bpf* permissions and making it readable by non-root users. Snort is configured using the Snort v3.0 community rules downloaded from <https://www.snort.org/downloads>. To configure Snort to process the PCAP file using the community rules, the community rules are moved to the Snort directory then the path is manually added to the configuration file.

```
[kingmudis@MA-Mac ~ % brew install snort
==> Downloading https://formulae.brew.sh/api/formula.jws.json
##### 100.0%
==> Downloading https://formulae.brew.sh/api/cask.jws.json
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/snort/manifests/3.6.0.0
##### 100.0%
==> Fetching dependencies for snort: daq, hwloc, jemalloc, libdnet, libpcap, lua
jit, pcre and vectorscan
==> Downloading https://ghcr.io/v2/homebrew/core/daq/manifests/3.0.17
##### 100.0%
==> Fetching daq
==> Downloading https://ghcr.io/v2/homebrew/core/daq/blobs/sha256:eefe2f055bfabe
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/hwloc/manifests/2.11.2
##### 100.0%
==> Fetching hwloc
==> Downloading https://ghcr.io/v2/homebrew/core/hwloc/blobs/sha256:02ca60d14701
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/jemalloc/manifests/5.3.0
##### 100.0%
==> Fetching jemalloc
==> Downloading https://ghcr.io/v2/homebrew/core/jemalloc/blobs/sha256:66b5f3a4c
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/libdnet/manifests/1.18.0
```

Figure 7: Installing Snort and its dependencies using the Homebrew package manager.

```

==> snort
For snort to be functional, you need to update the permissions for /dev/bpf*
so that they can be read by non-root users. This can be done manually using:
    sudo chmod o+r /dev/bpf*
or you could create a startup item to do this for you.
[kingmudis@MA-Mac ~ % sudo chmod o+r /dev/bpf*
[Password:
kingmudis@MA-Mac ~ % █

```

Figure 8: Manually updating the /dev/bpf* permissions so Snort can be functional.

```

kingmudis@MA-Mac ~ % sudo cp ~/Downloads/snort3-community-rules/snort3-community.
rules /usr/local/etc/snort
kingmudis@MA-Mac ~ % sudo nano /usr/local/etc/snort/snort.lua

```

UW PICO 5.09	File: /usr/local/etc/snort/snort.lua	Modified
--------------	--------------------------------------	----------

```

-- 5. configure detection
-----

references = default_references
classifications = default_classifications

ips =
{
    include = '/usr/local/etc/snort/snort3-community.rules',
    -- use this to enable decoder and inspector alerts
    --enable_built_in_rules = true,

    -- use include for rules files; be sure to set your path
    -- note that rules files can include other rules files
    -- (see also related path vars at the top of snort_defaults.lua)

    variables = default_variables
}

-- use these to configure additional rule actions
-- react = { }
-- reject = { }

-- use this to enable payload injection utility
-- payload_injector = { }

```

Figure 9: Copying the community rules to the Snort directory and adding the path to the Snort configuration file.

4.3 Processing the CIC-IDS2017 PCAP File using Snort

After installing and configuring Snort with the community rules, the PCAP file is processed to generate alerts when an attack is detected. The first command outputs alerts and statistics on the console while the second exports the alert logs in CSV format.

```
kingmudis@MA-Mac ~ % snort -c /usr/local/etc/snort/snort.lua -r ~/Downloads/pcap/cic_ids2017.pcap -A alert_fast
```

```
o")~ Snort++ 3.6.0.0
```

```
Loading /usr/local/etc/snort/snort.lua:
```

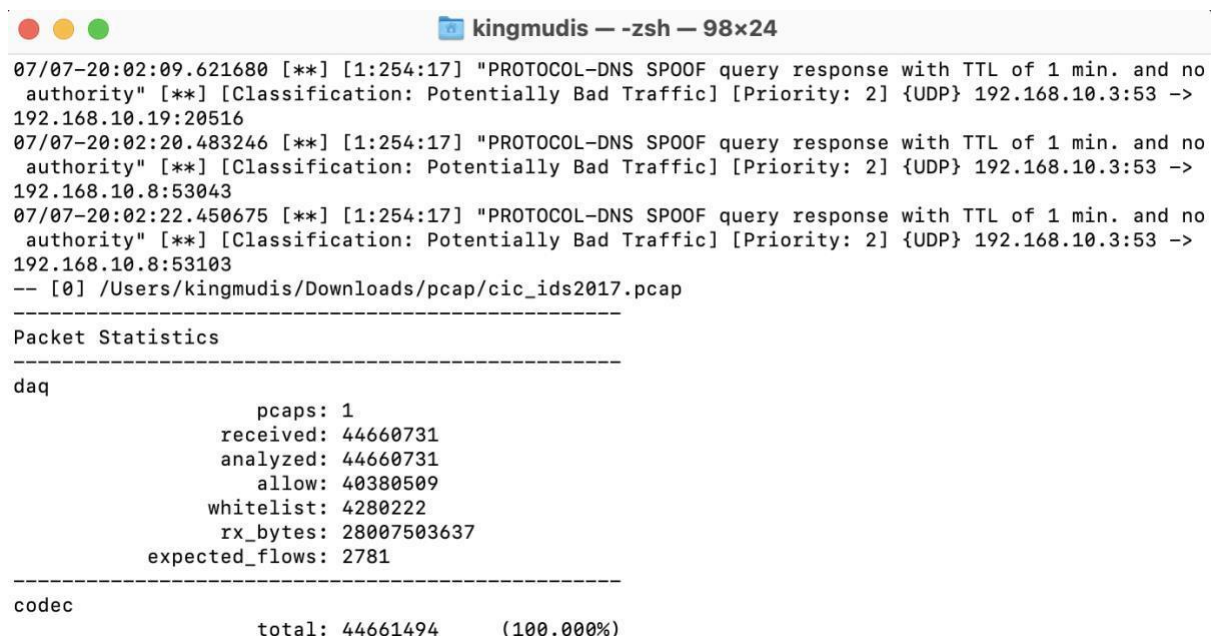
Figure 10: Copying the community rules to the Snort directory and adding the path to the Snort configuration file.

```
kingmudis@MA-Mac ~ % snort -c /usr/local/etc/snort/snort.lua -r ~/Downloads/pcap/cic_ids2017.pcap -A alert_fast
```

```
o")~ Snort++ 3.6.0.0
```

```
Loading /usr/local/etc/snort/snort.lua:
```

Figure 11: Analysing the CIC-IDS2017 PCAP file and outputting the alerts on the console.



```
kingmudis — -zsh — 98x24
07/07-20:02:09.621680 [**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min. and no authority" [**] [Classification: Potentially Bad Traffic] [Priority: 2] {UDP} 192.168.10.3:53 -> 192.168.10.19:20516
07/07-20:02:20.483246 [**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min. and no authority" [**] [Classification: Potentially Bad Traffic] [Priority: 2] {UDP} 192.168.10.3:53 -> 192.168.10.8:53043
07/07-20:02:22.450675 [**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min. and no authority" [**] [Classification: Potentially Bad Traffic] [Priority: 2] {UDP} 192.168.10.3:53 -> 192.168.10.8:53103
-- [0] /Users/kingmudis/Downloads/pcap/cic_ids2017.pcap

Packet Statistics
-----
daq
      pcaps: 1
    received: 44660731
    analyzed: 44660731
      allow: 40380509
    whitelist: 4280222
    rx_bytes: 28007503637
  expected_flows: 2781

-----
codec
      total: 44661494      (100.000%)
```

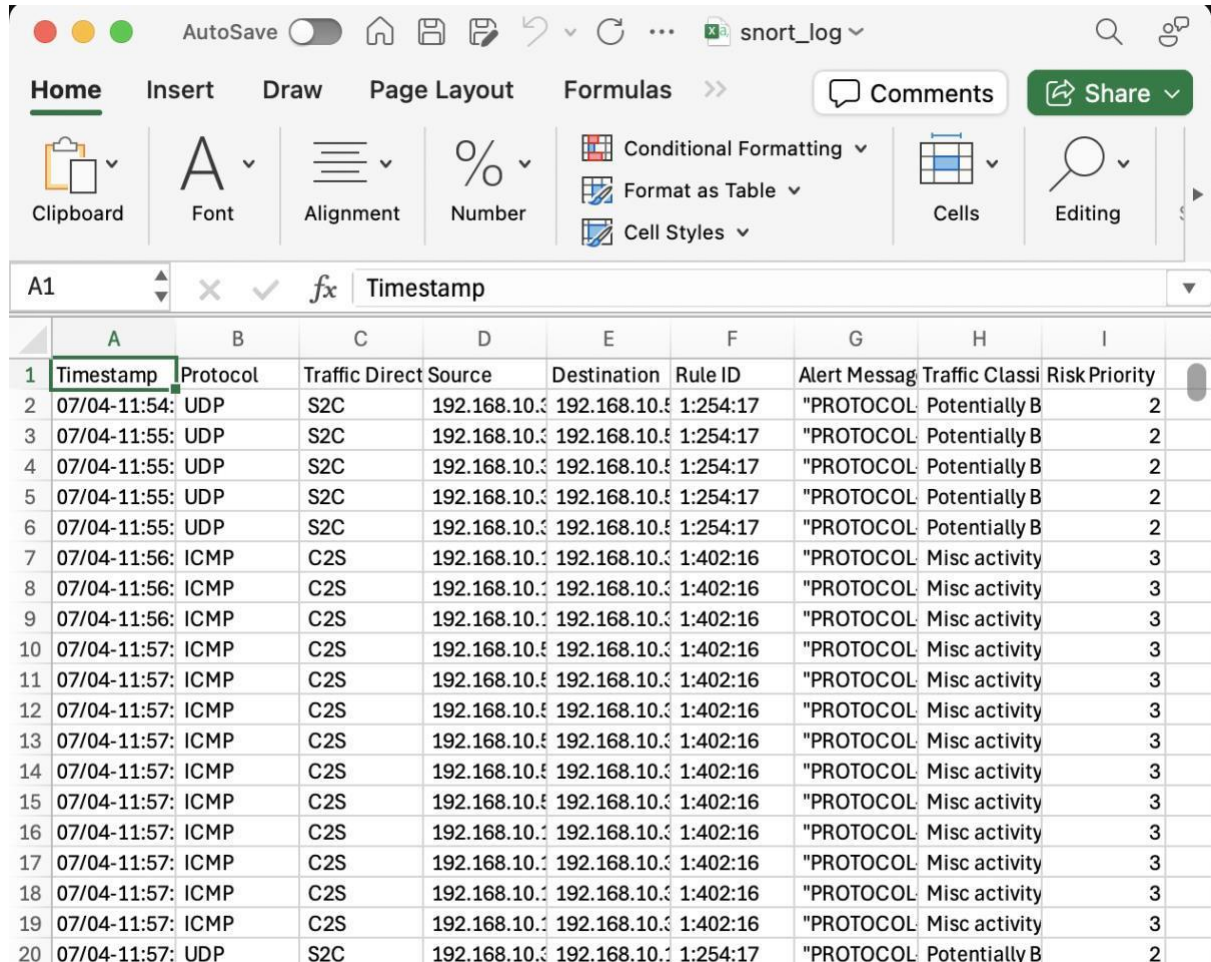
Figure 12: Subset of the output of the command ran in Figure 11.

```
kingmudis@MA-Mac ~ % snort -q -c /usr/local/etc/snort/snort.lua -R /usr/local/etc/snort/snort3-community.rules -r ~/Downloads/pcap/cic_ids2017.pcap -A alert_csv --lua "alert_csv = {fields = 'timestamp proto dir src_ap ds t_ap rule msg class priority '}" > ~/Downloads/logs/snort_log.csv
kingmudis@MA-Mac ~ %
```

Figure 13: Analysing the CIC-IDS2017 PCAP file in quiet mode and exporting the alerts logs in CSV format with specified fields.

4.4 Initial Exploration with Microsoft Excel

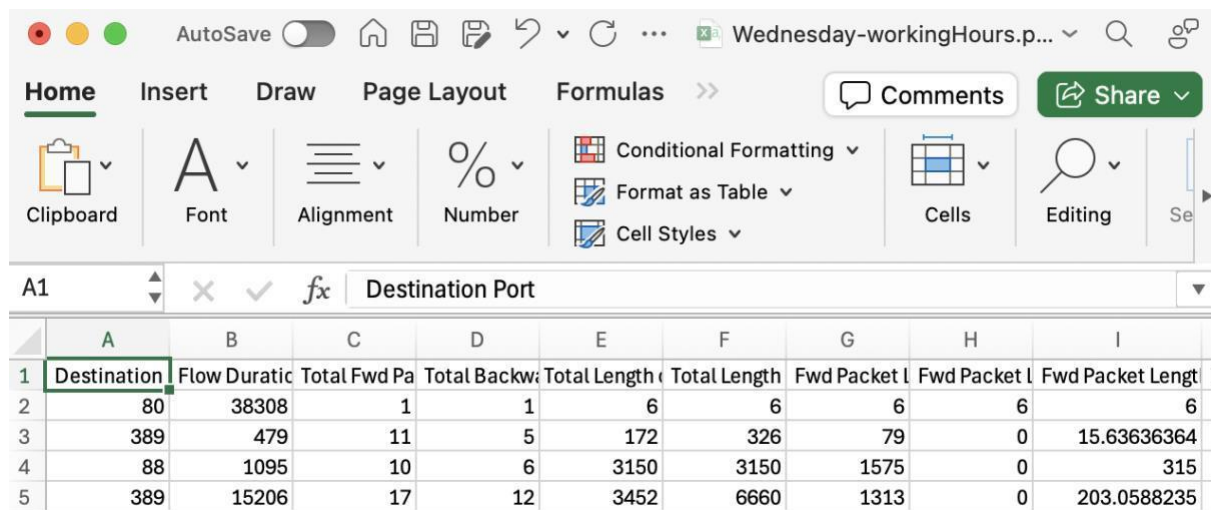
Microsoft Excel was used to conduct an initial exploration of the generated Snort alert to get an overview of the log entries. It was also employed in manually adding header fields to the generated log. It was also used to explore the downloaded CIC-IDS2017 CSV files for machine learning.



The screenshot shows the Microsoft Excel interface with the 'snort_log' file open. The formula bar shows 'Timestamp' in cell A1. The spreadsheet contains the following data:

	A	B	C	D	E	F	G	H	I
1	Timestamp	Protocol	Traffic Direction	Source	Destination	Rule ID	Alert Message	Traffic Classification	Risk Priority
2	07/04-11:54:	UDP	S2C	192.168.10.5	192.168.10.5	1:254:17	"PROTOCOL	Potentially B	2
3	07/04-11:55:	UDP	S2C	192.168.10.5	192.168.10.5	1:254:17	"PROTOCOL	Potentially B	2
4	07/04-11:55:	UDP	S2C	192.168.10.5	192.168.10.5	1:254:17	"PROTOCOL	Potentially B	2
5	07/04-11:55:	UDP	S2C	192.168.10.5	192.168.10.5	1:254:17	"PROTOCOL	Potentially B	2
6	07/04-11:55:	UDP	S2C	192.168.10.5	192.168.10.5	1:254:17	"PROTOCOL	Potentially B	2
7	07/04-11:56:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
8	07/04-11:56:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
9	07/04-11:56:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
10	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
11	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
12	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
13	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
14	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
15	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
16	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
17	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
18	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
19	07/04-11:57:	ICMP	C2S	192.168.10.5	192.168.10.5	1:402:16	"PROTOCOL	Misc activity	3
20	07/04-11:57:	UDP	S2C	192.168.10.5	192.168.10.5	1:254:17	"PROTOCOL	Potentially B	2

Figure 14: Using Microsoft Excel to Add header fields to the generated Snort alert log.



The screenshot shows the Microsoft Excel interface with the file 'Wednesday-workingHours.p...' open. The formula bar shows 'Destination Port' in cell A1. The spreadsheet contains the following data:

	A	B	C	D	E	F	G	H	I
1	Destination	Flow Duration	Total Fwd Pa	Total Backw	Total Length	Total Length	Fwd Packet L	Fwd Packet L	Fwd Packet Leng
2	80	38308	1	1	6	6	6	6	6
3	389	479	11	5	172	326	79	0	15.63636364
4	88	1095	10	6	3150	3150	1575	0	315
5	389	15206	17	12	3452	6660	1313	0	203.0588235

Figure 15: Exploring the Wednesday subset of the CIC-IDS2017 dataset using Excel.

4.5 Analysing & Visualising the Snort Alert Logs

To get insight into the Snort Alert logs, Jupyter Notebook was employed to analyse and visualise the generated logs. Jupyter Notebook and other Python modules were installed using the “python3 -m pip install -U jupyter matplotlib numpy pandas scipy sci-kit-learn seaborn” and running it using the “Jupyter notebook” command. The “Jupyter server list” command was used to display the Jupyter server URL and token. Upon accessing the Jupyter server using the URL, the Snort log was moved to the research_project folder on the Downloads directory for easier access, and a new notebook was created titled ‘workflow.ipynb’ before commencing analysis and visualisation.

[3]:

```
#Importing relevant Python Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:

```
#ANALYSIS & VISUALISATION OF THE SNORT LOG GENERATED FROM THE CIC-IDS2017 PCAP FILE

#Loading the csv log
snort_alerts = pd.read_csv('snort_log.csv')

#Statistics from Snort Analysis of CICIDS-2017 PCAP file
total_traffic_analysed = 44660731
alerts_triggered = snort_alerts.shape[0]
benign_traffic = total_traffic_analysed - alerts_triggered
snort_throughput = 50887
snort_runtime = 877.645486
```

Figure 16: Importing the Python libraries, loading the Snort Log into the dataframe, and storing log statistics in variables.

[12]:

```
#Displaying first 5 rows of the Snort Logs
snort_alerts.head(5)
```

[12]:

	Timestamp	Protocol	Traffic Direction	Source	Destination	Rule ID	Alert Message
0	07/04-11:54:56.360696	UDP	S2C	192.168.10.3:53	192.168.10.5:52825	1:254:17	"PROTOCOL-DNS SPOOF query response with TTL o...
1	07/04-11:55:02.029403	UDP	S2C	192.168.10.3:53	192.168.10.5:54941	1:254:17	"PROTOCOL-DNS SPOOF query response with TTL o...
2	07/04-11:55:02.135535	UDP	S2C	192.168.10.3:53	192.168.10.5:60081	1:254:17	"PROTOCOL-DNS SPOOF query response with TTL o...
3	07/04-11:55:05.743099	UDP	S2C	192.168.10.3:53	192.168.10.5:54459	1:254:17	"PROTOCOL-DNS SPOOF query response with TTL o...
4	07/04-11:55:05.743833	UDP	S2C	192.168.10.3:53	192.168.10.5:65359	1:254:17	"PROTOCOL-DNS SPOOF query response with TTL o...

Figure 17: Displaying the top 5 rows of the Snort logs.

[14]:

```
#Outputting the structure of the logs
print('\nStructure of the Snort Log: ')
print(f'Columns: {snort_alerts.shape[1]}')
print(f'Rows: {snort_alerts.shape[0]}')
```

Structure of the Snort Log:

Columns: 9

Rows: 266327

[15]:

```
#Concise summary of the Snort log's structure
snort_alerts.info()
```

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 266327 entries, 0 to 266326

Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype
0	Timestamp	266327 non-null	object
1	Protocol	266327 non-null	object
2	Traffic Direction	266327 non-null	object
3	Source	266327 non-null	object
4	Destination	266327 non-null	object
5	Rule ID	266327 non-null	object
6	Alert Message	266327 non-null	object
7	Traffic Classification	266327 non-null	object
8	Risk Priority	266327 non-null	int64

dtypes: int64(1), object(8)

memory usage: 18.3+ MB

Figure 18: Exploratory Data Analysis of the Snort log.

```
#Visualisation Snort Log traffic distribution using Pie Chart
```

```
snort_traffic_labels = 'MALICIOUS', 'BENIGN'
snort_traffic_values = [alerts_triggered, benign_traffic]
sns.set_palette('hls', 8)
plt.figure(figsize = (5, 5))
plt.pie(snort_traffic_values, labels = snort_traffic_labels, autopct = '%1.1f%%',
        textprops={'fontsize': 10})
plt.title('BENIGN vs MALICIOUS (Snort Log)')
plt.legend(snort_traffic_labels, loc = 'best')
plt.show()
```

Figure 19: Employing a Pie Chart to visualise the traffic distribution in the CIC-IDS2017 PCAP File.

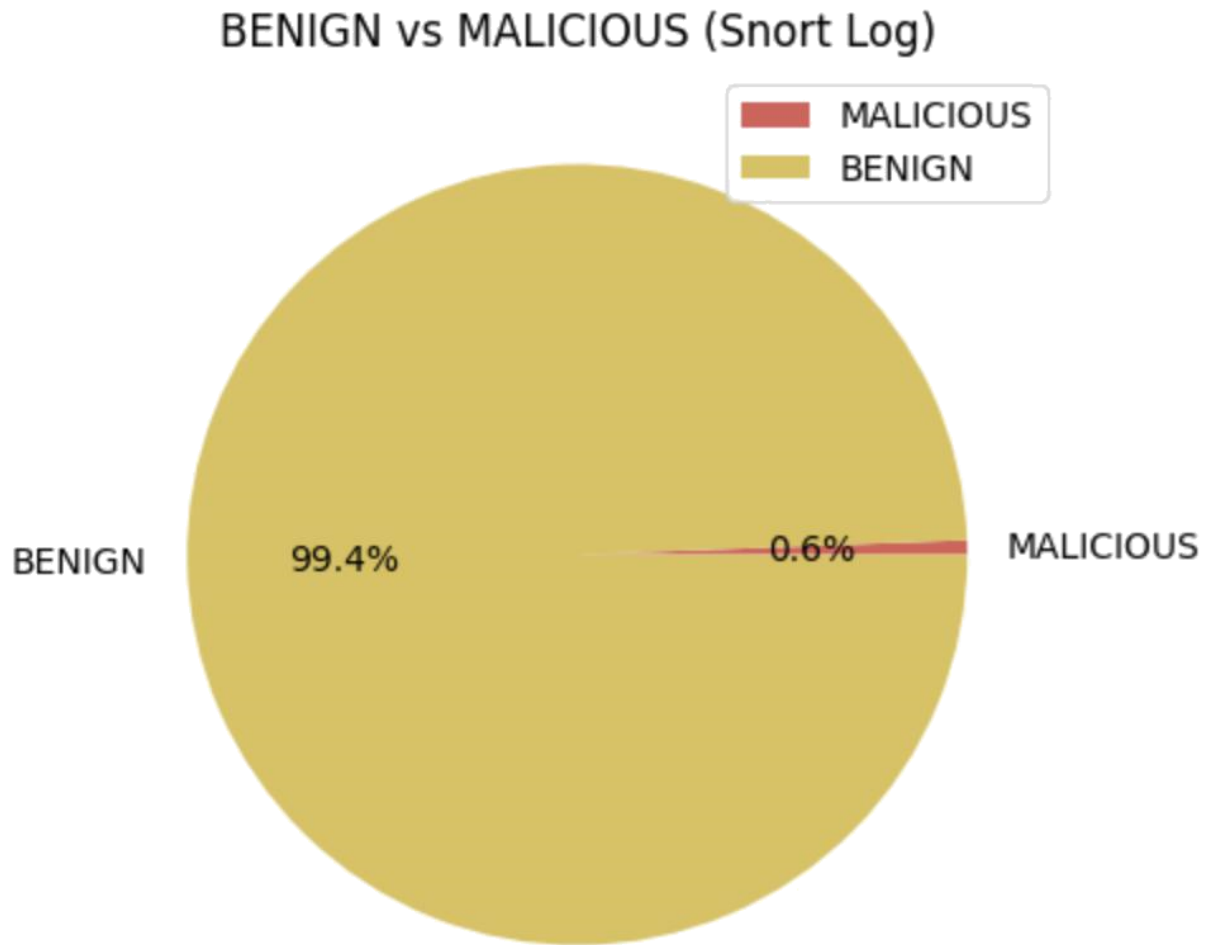


Figure 20: Traffic distribution of the CIC-IDS2017 PCAP file.

```
#Displaying Snort log's risk priorities distribution
risk_priority = { #Using Low, Medium, High as the Risk Priorities instead of 1,2,3
    1: 'High Risk',
    2: 'Medium Risk',
    3: 'Low Risk'
}
snort_alerts['Risk Priority'] = snort_alerts['Risk Priority'].map(risk_priority)
#The Counts Risk Priorities of the Logs
snort_alerts['Risk Priority'].value_counts()
```

```
[23]:
```

Risk Priority	
High Risk	175822
Medium Risk	85903
Low Risk	4602

Name: count, dtype: int64

Figure 21: Attack's risk priorities of the Snort log.


```
#Snort log's Malicious Traffic Classification Distribution
snort_alerts['Traffic Classification'].value_counts()
```

```
[24]:
```

```
Traffic Classification
Attempted Administrator Privilege Gain      175425
Potentially Bad Traffic                     84029
Misc activity                              3240
Detection of a Network Scan                1359
Attempted Information Leak                 1043
Information Leak                           441
Executable code was detected              383
Attempted Denial of Service                314
Access to a potentially vulnerable web application  67
Web Application Attack                     9
Successful Administrator Privilege Gain     5
A system call was detected                 4
An attempted login using a suspicious username was detected  3
Misc Attack                               2
A suspicious string was detected           2
Generic Protocol Command Decode            1
Name: count, dtype: int64
```

Figure 22: Displaying the Snort log's traffic classification distribution.

```
#Visualising the Snort log's traffic classification distribution.
traffic_distribution = snort_alerts['Traffic Classification'].value_counts()
snort_log_threshold = 0.010
snort_log_percent = traffic_distribution / traffic_distribution.sum()
snort_log = snort_log_percent[snort_log_percent < snort_log_threshold].index.tolist()
traffic_distribution['Others'] = traffic_distribution[snort_log ].sum()
traffic_distribution.drop(snort_log , inplace = True)
sns.set_palette('hls', 8)
plt.figure(figsize = (6.5, 6.5))
plt.pie(traffic_distribution.values, labels = traffic_distribution.index,
        autopct = '%1.1f%%', textprops={'fontsize': 10})
plt.title('Attacks Distribution (Snort Log)')
plt.legend(traffic_distribution.index, loc = 'best')
plt.show()
```

Figure 23: Visualising the Snort log's traffic classification distribution.

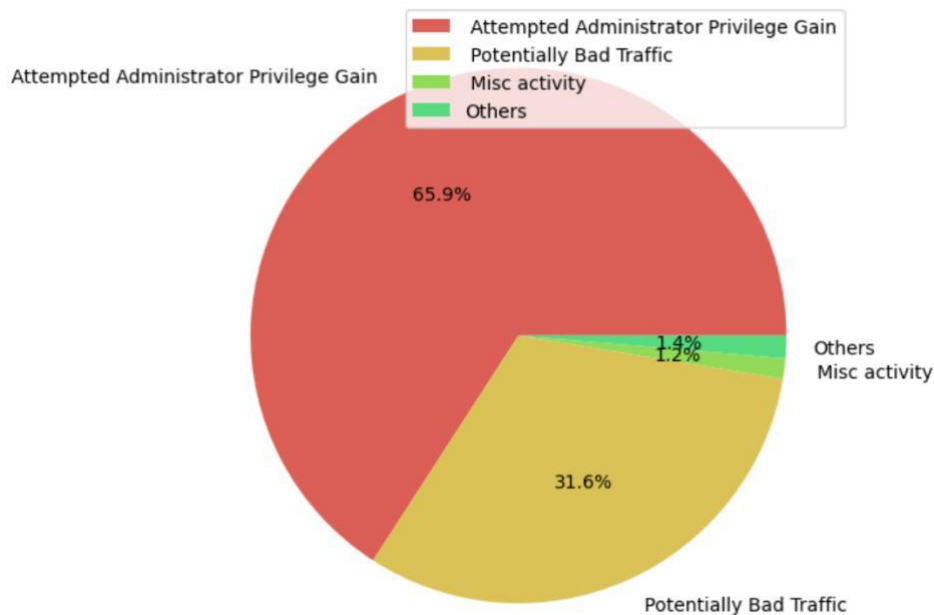


Figure 24: The Snort log's traffic classification distribution

4.6 Employing Machine Learning Models on the CIC-IDS2017 Dataset

After analysing the CIC-IDS2017 PCAP file using Snort open-source IDS/IPS, the next step was to evaluate the performance of three machine models on the machine-learning CSV files of the CIC-IDS dataset. The Monday subset of the dataset was excluded because it contained only benign traffic, focusing on just Tuesday, Wednesday, Thursday, and Friday. The machine learning CSV files were loaded as a dataframe on Jupyter Notebook, then the following steps were performed:

- Exploratory Data Analysis.
- Data Cleaning and Preprocessing.
- Feature Engineering.
- Models' Training and Testing.
- Performance Evaluation.

```
#Importing the ML-Labeled CIC-IDS2017 dataset
#Monday traffic was ignored due absence of attack traffic

tuesday = pd.read_csv('Tuesday-WorkingHours.pcap_ISCX.csv')
wednesday = pd.read_csv('Wednesday-workingHours.pcap_ISCX.csv')
thursday = pd.read_csv('Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv')
thursday_afternoon = pd.read_csv('Thursday-WorkingHours-Afternoon-Infiltration.pcap_ISCX.csv')
friday = pd.read_csv('Friday-WorkingHours-Morning.pcap_ISCX.csv')
friday_noon = pd.read_csv('Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv')
friday_afternoon = pd.read_csv('Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv')

#Compiling datasets
dataset_compiled = [tuesday, wednesday, thursday, thursday_afternoon, friday, friday_noon,
                    friday_afternoon]
cic_ids2017 = pd.concat(dataset_compiled)

#Removing individual datasets to free up used memory
del tuesday, wednesday, thursday, thursday_afternoon, friday, friday_noon, friday_afternoon,
dataset_compiled

#Outputting the structure of the logs
print('\nStructure CIC_IDS2017 Dataset: ')
print(f'Columns: {cic_ids2017.shape[1]}')
print(f'Rows: {cic_ids2017.shape[0]}')
```

Structure CIC_IDS2017 Dataset:
Columns: 79
Rows: 2300825

Figure 25: Loading the CIC-IDS2017 dataset into a dataframe on Jupyter Notebook.

```
#Ensure all columns are visible
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

cic_ids2017.head(5) #View the top 5 rows of the dataset
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std
0	88	640	7	4	440	358	220	0	62.857143	107.349008
1	88	900	9	4	600	2944	300	0	66.666667	132.287566
2	88	1205	7	4	2776	2830	1388	0	396.571429	677.274651
3	88	511	7	4	452	370	226	0	64.571429	110.276708
4	88	773	9	4	612	2944	306	0	68.000000	134.933317

Figure 26: Ensuring all columns, rows are visible and displaying the top 5 rows.

```
cic_ids2017.info() #A concise overview of the dataset
```

```
<class 'pandas.core.frame.DataFrame'>  
Index: 2300825 entries, 0 to 225744  
Data columns (total 79 columns):  
#   Column                                Dtype  
---  -  
0   Destination Port                     int64  
1   Flow Duration                         int64  
2   Total Fwd Packets                     int64  
3   Total Backward Packets                int64  
4   Total Length of Fwd Packets           int64  
5   Total Length of Bwd Packets           int64  
6   Fwd Packet Length Max                 int64  
7   Fwd Packet Length Min                 int64  
8   Fwd Packet Length Mean                float64  
9   Fwd Packet Length Std                 float64  
10  Bwd Packet Length Max                  int64
```

Figure 27: Exploratory Data Analysis of the loaded CIC-IDS2017 dataset

```
#Removing whitespace from columns heading  
column_heading = {column: column.strip() for column in cic_ids2017.columns}  
cic_ids2017.rename(columns = column_heading, inplace = True)
```

Figure 28: Removing whitespace from column name.

```
#Visualising traffic distribution in the CIC-IDS2017 dataset  
  
#Group traffic into BENIGN & MALICIOUS  
cic_ids_copy = cic_ids2017.copy() #Making a copy of the dataframe  
cic_ids_copy['Label'] = cic_ids_copy['Label'].replace({  
    'BENIGN': 'BENIGN',  
    'DoS Hulk': 'MALICIOUS',  
    'DDoS': 'MALICIOUS',  
    'PortScan': 'MALICIOUS',  
    'DoS GoldenEye': 'MALICIOUS',  
    'FTP-Patator': 'MALICIOUS',  
    'DoS slowloris': 'MALICIOUS',  
    'DoS Slowhttptest': 'MALICIOUS',  
    'SSH-Patator': 'MALICIOUS',  
    'Bot': 'MALICIOUS',  
    'Web Attack @ Brute Force': 'MALICIOUS',  
    'Web Attack @ XSS': 'MALICIOUS',  
    'Infiltration': 'MALICIOUS',  
    'Web Attack @ Sql Injection': 'MALICIOUS',  
    'Heartbleed': 'MALICIOUS'  
})  
  
#Visualisation the Traffic distribution using Pie-Chart  
  
cic_ids_traffic_counts = cic_ids_copy['Label'].value_counts()  
sns.set_palette('hls', 8)  
plt.figure(figsize = (6.5, 6.5))  
plt.pie(cic_ids_traffic_counts.values, labels = cic_ids_traffic_counts.index,  
        autopct = '%1.1f%%', textprops={'fontsize': 10})  
plt.title('BENIGN vs MALICIOUS (CIC_IDS2017)')  
plt.legend(cic_ids_traffic_counts.index, loc = 'best')  
plt.show()
```

Figure 29: Visualising CIC-IDS2017 dataset's traffic distribution.

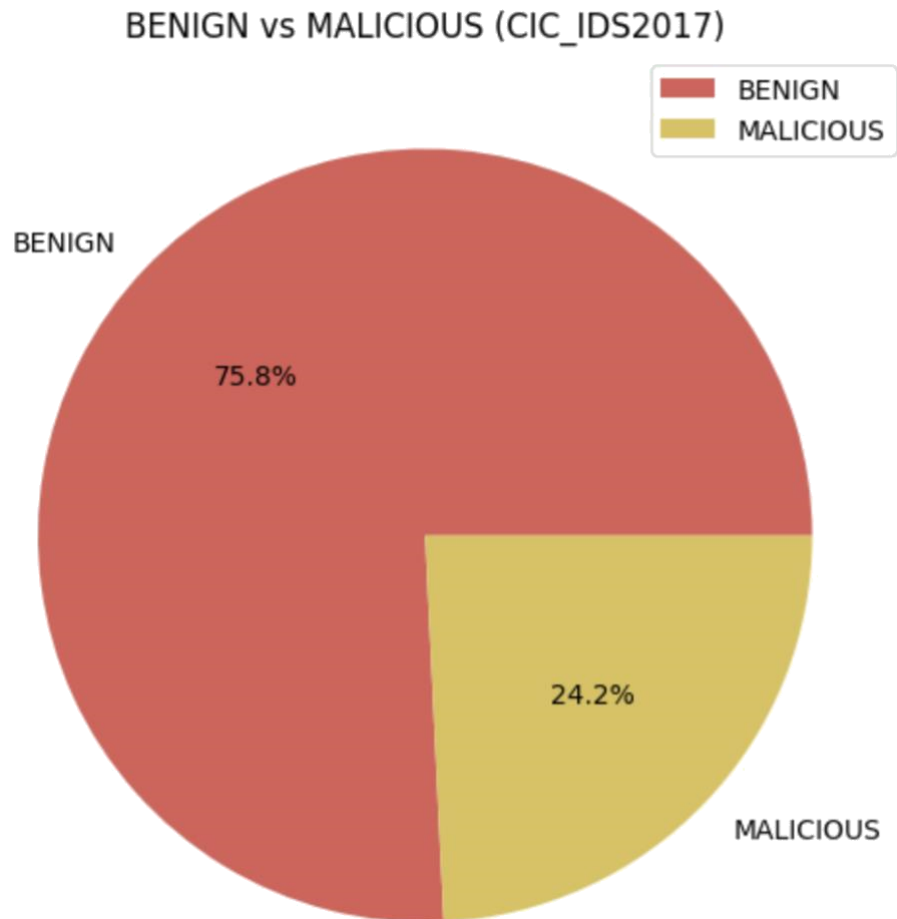


Figure 30: CIC-IDS2017 dataset's distribution of benign & malicious traffic.

```
cic_ids2017['Label'].value_counts() #Displaying traffic labels and their frequency
```

[15]:

Label	
BENIGN	1743179
DoS Hulk	231073
PortScan	158930
DDoS	128027
DoS GoldenEye	10293
FTP-Patator	7938
SSH-Patator	5897
DoS slowloris	5796
DoS Slowhttptest	5499
Bot	1966
Web Attack 0 Brute Force	1507
Web Attack 0 XSS	652
Infiltration	36
Web Attack 0 Sql Injection	21
Heartbleed	11

Name: count, dtype: int64

Figure 31: Count of CIC-IDS2017 traffic.

```
#Visualising the CIC-IDS2017 malicious traffic distribution
```

```
cic_ids_traffic = cic_ids2017.loc[cic_ids2017['Label'] != 'BENIGN']
cic_ids_attacks = cic_ids_traffic['Label'].value_counts()
attack_threshold = 0.05
attack_percentages = cic_ids_attacks / cic_ids_attacks.sum()
cic_ids_ss = attack_percentages[attack_percentages < attack_threshold].index.tolist()
cic_ids_attacks['Others'] = cic_ids_attacks[cic_ids_ss].sum()
cic_ids_attacks.drop(cic_ids_ss, inplace = True)
sns.set_palette('hls', 8)
plt.figure(figsize = (6.5, 6.5))
plt.pie(cic_ids_attacks.values, labels = cic_ids_attacks.index, autopct = '%1.1f%%',
        textprops={'fontsize': 10})
plt.title('Malicious Traffic Distribution (CIC-IDS2017)')
plt.legend(cic_ids_attacks.index, loc = 'best')
plt.show()
```

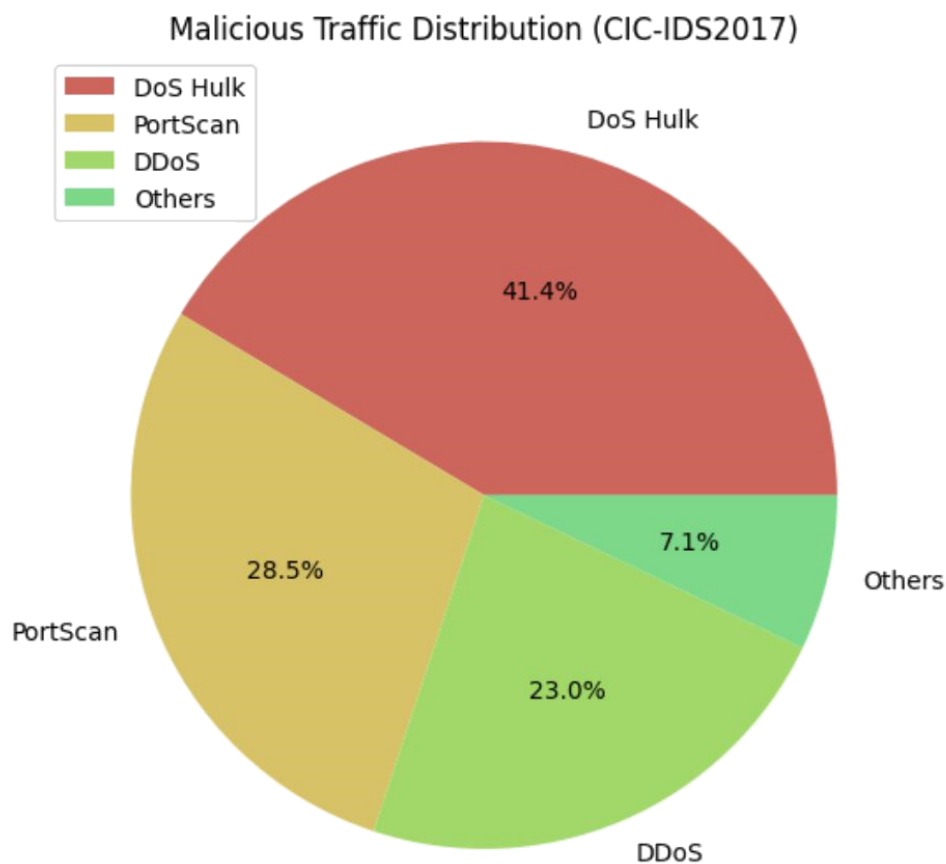


Figure 32: Visualising CIC-IDS2017 dataset's malicious traffic distribution.

```
#Checking for duplicate entries in the dataset
duplicate_entries = cic_ids2017.duplicated().sum()
print (f'Number of Duplicate Entries: {duplicate_entries}')
```

Number of Duplicate Entries: 281095

```
cic_ids2017.drop_duplicates(inplace = True) #Removing duplicates
```

Figure 33: Handling duplicates.


```

#Displaying Columns with Null and Infinite values
print('Number of Null values: ')
null_values = cic_ids2017.isna().sum()
print(null_values.loc[null_values > 0])

print('\n Number of Infinite Values: ')
numerical_columns = cic_ids2017.select_dtypes(include=[np.number])
infinite_values = np.isinf(numerical_columns).sum()
print(infinite_values[infinite_values > 0])

```

Number of Null values:
Flow Bytes/s 302
dtype: int64

Number of Infinite Values:
Flow Bytes/s 955
Flow Packets/s 1257
dtype: int64

```

#Handling missing Null and Infinite values by imputting the median since they are numerical
null_values = cnull_values = cic_ids2017['Flow Bytes/s'].median()
cic_ids2017['Flow Bytes/s'] = cic_ids2017['Flow Bytes/s'].fillna(null_values)

cic_ids2017['Flow Bytes/s'] = cic_ids2017['Flow Bytes/s'].replace([np.inf, -np.inf],
                                                                cic_ids2017['Flow Bytes/s'].median())
cic_ids2017['Flow Packets/s'] = cic_ids2017['Flow Packets/s'].replace([np.inf, -np.inf],
                                                                cic_ids2017['Flow Packets/s'].median())

```

Figure 34: Handling missing and infinite values.

```

#Rechecking for missing values
print('Number of Null values: ')
null_values = cic_ids2017.isna().sum()
print(null_values.loc[null_values > 0])

print('\n Number of Infinite Values: ')
numerical_columns = cic_ids2017.select_dtypes(include=[np.number])
infinite_values = np.isinf(numerical_columns).sum()
print(infinite_values[infinite_values > 0])

```

Number of Null values:
Series([], dtype: int64)

Number of Infinite Values:
Series([], dtype: int64)

Figure 35: Rechecking for missing values.

```
cic_ids2017.shape
```

(2019730, 79)

```

#Checking number of unique values in each columns on the dataset
unique_columns = cic_ids2017.nunique()
print(unique_columns)

```

Figure 36: Checking shape and number of unique values on each column.


```
#Removing columns with just a single unique value
cic_ids2017.drop(columns=['Bwd PSH Flags', 'Bwd URG Flags', 'Fwd Avg Bytes/Bulk',
                        'Fwd Avg Packets/Bulk', 'Fwd Avg Bulk Rate', 'Bwd Avg Bytes/Bulk',
                        'Bwd Avg Packets/Bulk', 'Bwd Avg Bulk Rate'], inplace=True)
```

Figure 37: Removing features with just a single unique value due to irrelevance.

```
#Grouping similar attacks together
cic_ids2017['Label'] = cic_ids2017['Label'].replace({
    'BENIGN': 'Benign', 'DoS Hulk': 'DoS/DDoS', 'DDoS': 'DoS/DDoS', 'PortScan': 'MALICIOUS',
    'DoS GoldenEye': 'DoS/DDoS', 'FTP-Patator': 'FTP/SSH Patator Attack',
    'DoS slowloris': 'DoS/DDoS', 'DoS Slowhttptest': 'DoS/DDoS', 'SSH-Patator': 'FTP/SSH Patator',
    'Bot': 'Bot Attack', 'Web Attack @ Brute Force': 'Brute Force Attack',
    'Web Attack @ XSS': 'Injection Attack', 'Infiltration': 'Infiltration Attack',
    'Web Attack @ Sql Injection': 'Injection Attack', 'Heartbleed': 'Heartbleed Attack'
})
cic_ids2017['Label'].value_counts()
```

Label	
Benign	1593852
DoS/DDoS	321764
MALICIOUS	90819
FTP/SSH Patator Attack	9152
Bot Attack	1953
Brute Force Attack	1470
Injection Attack	673
Infiltration Attack	36
Heartbleed Attack	11
Name: count, dtype: int64	

Figure 38: Grouping similar attacks.

```
#Standardising the features using StandardScaler to improve the efficiency of the models.
from sklearn.preprocessing import StandardScaler #Importing StandardScaler
selected_features = cic_ids2017_copy.drop('Label', axis=1) #Excluding the target feature.
target_feature = cic_ids2017_copy['Label']
scaler = StandardScaler()
standardised_features = scaler.fit_transform(selected_features)
```

```
#Reducing the dimensions of the dataset using the Analysis of Principal Component Technique
from sklearn.decomposition import IncrementalPCA #Importing the dimension reduction class
selected_dimension = len(selected_features.columns) // 2 #Specifying the dimension
dimension_reduction = IncrementalPCA(n_components = selected_dimension, batch_size = 1000)
ratio = len(selected_features) // batch_size
for segment in np.array_split(standardised_features, ratio):
    dimension_reduction.partial_fit(segment)
```

```
#Merging target features with standardised and reduced features
standardised_reduced = dimension_reduction.transform(standardised_features)
cic_ids_merged = pd.DataFrame(
    standardised_reduced,
    columns=[f'PC{index+1}' for index in range(selected_dimension)]
)
cic_ids_merged['Label'] = target_feature.values
```

Figure 39: Standardising and reducing the dimensions of the datasets to improve the models' efficiency.

```
cic_ids_merged['Label'].value_counts() #Viewing imbalance in the traffic classification
```

```
Label
Benign                1593852
DoS/DDoS              321764
MALICIOUS             90819
FTP/SSH Patator Attack 9152
Bot Attack            1953
Brute Force Attack    1470
Injection Attack       673
Infiltration Attack    36
Heartbleed Attack      11
Name: count, dtype: int64
```

Figure 40: Traffic classification before undersampling.

```
#To prevent imbalance Undersampling traffic class to max 5000 rows
traffic_count = cic_ids_merged ['Label'].value_counts()
traffic_values = traffic_count.index

sampled_traffic = cic_ids_merged[cic_ids_merged['Label'].isin(traffic_values)]
cic_ids_undersampled = []
for traffic in traffic_values:
    traffic_captured = sampled_traffic[sampled_traffic['Label'] == traffic]
    if len(traffic_captured) > 3000:
        samples = min(len(traffic_captured), 5000)
        traffic_captured = traffic_captured.sample(n=samples, random_state=0)
    cic_ids_undersampled.append(traffic_captured)
```

Figure 41: Undersampling majoring traffic class.

```
#Displaying traffic classification after undersampling
cic_ids_imbalance = pd.concat(cic_ids_undersampled, ignore_index = True)
cic_ids_imbalance['Label'].value_counts()
```

```
Label
Benign                5000
DoS/DDoS              5000
MALICIOUS             5000
FTP/SSH Patator Attack 5000
Bot Attack            1953
Brute Force Attack    1470
Injection Attack       673
Infiltration Attack    36
Heartbleed Attack      11
Name: count, dtype: int64
```

Figure 42: Traffic classification after undersampling.

```
#Oversampling the minority traffic class to balance dataset
from imblearn.over_sampling import SMOTE
X = cic_ids_imbalance .drop('Label', axis=1)
y = cic_ids_imbalance ['Label']
sm = SMOTE(sampling_strategy='auto', random_state=42)
X_sam, y_sam = sm.fit_resample(X, y)
```

Figure 43: Traffic classification after undersampling.

```
#Viewing balanced traffic classification
cic_ids_balance = pd.DataFrame(X_sam)
cic_ids_balance['Label'] = y_sam
cic_ids_balances = cic_ids_balance.sample(frac=1)
cic_ids_balance['Label'].value_counts()
```

```
Label
Benign                5000
DoS/DDoS              5000
MALICIOUS             5000
FTP/SSH Patator Attack 5000
Bot Attack            5000
Brute Force Attack    5000
Injection Attack      5000
Infiltration Attack   5000
Heartbleed Attack     5000
Name: count, dtype: int64
```

Figure 44: Viewing balanced traffic classification.

```
#Splitting datasets into 80:20 for training and testing models.
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import time
features = cic_ids_balance.drop('Label', axis = 1)
targets = cic_ids_balance['Label']

X_train, X_test, y_train, y_test = train_test_split(
    features, targets, test_size = 0.20, random_state = 42
)
```

Figure 45: Splitting dataset into 80:20 for training and testing models.

```
#Employing Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
d_tree = DecisionTreeClassifier (max_depth = 10)
d_tree.fit(X_train, y_train)
d_tree_start = time.time()
d_tree_predictions = d_tree.predict(X_test)
d_tree_stop = time.time()
d_tree_time = round(d_tree_stop - d_tree_start, 3)
d_tree_throughput = round(len(X_test) / d_tree_time, 3)
d_tree_accuracy = round(accuracy_score(y_test, d_tree_predictions), 3)
```

Figure 46: Employing decision tree classifier on the CIC-IDS2017 dataset.

```

#Employing K-Nearest Neighbors Classifier
from sklearn.neighbors import KNeighborsClassifier
k_nearest = KNeighborsClassifier(n_neighbors = 7)
k_nearest.fit(X_train, y_train)
k_nearest_start = time.time()
k_nearest_predictions = d_tree.predict(X_test)
k_nearest_predictions = k_nearest.predict(X_test)
k_nearest_stop = time.time()
k_nearest_time = round(k_nearest_stop - k_nearest_start, 3)
k_nearest_throughput = round(len(X_test) / k_nearest_time, 3)
k_nearest_accuracy = round(accuracy_score(y_test, k_nearest_predictions), 3)

```

```

#Employing Random Forest
from sklearn.ensemble import RandomForestClassifier

r_forest = RandomForestClassifier(n_estimators = 10, max_depth = 10, max_features = None,
                                random_state = 42)

r_forest.fit(X_train, y_train)
r_forest_start = time.time()
r_forest_predictions = r_forest.predict(X_test)
r_forest_stop = time.time()
r_forest_time = round(r_forest_stop - r_forest_start, 3)
r_forest_throughput = round(len(X_test) / r_forest_time, 3)
r_forest_accuracy = round(accuracy_score(y_test, r_forest_predictions), 3)

```

Figure 47: Employing K neighbors and random forest classifiers on the CIC-IDS2017 dataset.

```

#Evaluating the models performance using their accuracy.
print (f'Decision Tree: {d_tree_accuracy }')
print (f'K-Nearest Neighbor: {k_nearest_accuracy }')
print (f'Random Forest: {r_forest_accuracy }')

```

```

Decision Tree: 0.904
K-Nearest Neighbor: 0.929
Random Forest: 0.919

```

Figure 48: Displaying the models' accuracy.

```

#Visualising model's performance
palette = sns.color_palette('hls', n_colors = 8)
labels = ['Decision Trees', 'K Nearest Neighbours', 'Random Forest']
scores = [d_tree_accuracy, k_nearest_accuracy, r_forest_accuracy ]
fig, ax = plt.subplots(figsize = (9, 3))
ax.barh(labels, scores, color = palette)
ax.set_xlim([0, 1])
ax.set_xlabel('Accuracy')
ax.set_title('ML Model Performance Comparison')
for i, v in enumerate(scores):
    ax.text(v + 0.01, i, str(round(v, 4)), ha = 'left', va = 'center')

plt.show()

```

Figure 49: Visualising the model's performance.

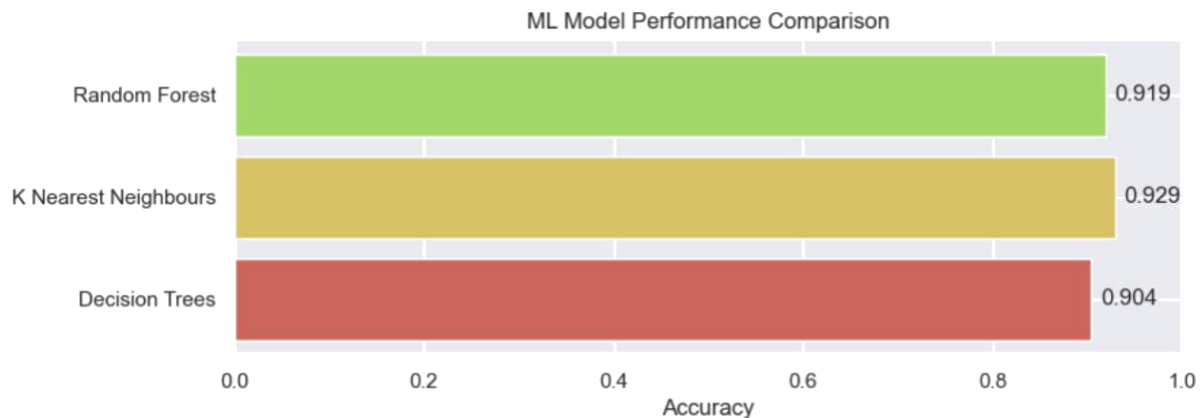


Figure 50: Visualising the model's performance.

```
print (f'Runtime(seconds)')
print (f'Snort: {snort_runtime }')
print (f'Decision Tree: {d_tree_time}')
print (f'K-Nearest Neighbor: {k_nearest_time} ')
print (f'Random Forest: {r_forest_time}')
print('\n')
print (f'Throughput(events/seconds)')
print (f'Snort: {snort_throughput } ')
print (f'Decision Tree: {d_tree_throughput}')
print (f'K-Nearest Neighbor: {k_nearest_throughput}')
print (f'Random Forest: {r_forest_throughput}')
```

```
Runtime(seconds)
Snort: 877.645
Decision Tree: 0.006
K-Nearest Neighbor: 1.256
Random Forest: 0.015
```

```
Throughput(events/seconds)
Snort: 50887
Decision Tree: 1500000.0
K-Nearest Neighbor: 7165.605
Random Forest: 600000.0
```

Figure 51: Displaying Snort's and models' performance.

```
#Visualizing Snort's and model's runtime
y = ['SNORT', 'DECISION TREE', 'KNN', 'RANDOM FOREST']
x = [snort_throughput, d_tree_throughput, k_nearest_throughput, r_forest_throughput]
palette = sns.color_palette('hls', 8)
colors = palette[:len(y)]
plt.barh(y, x, color=colors)
plt.xlabel("Events per second")
plt.title("SNORT VS ML (THROUGHPUT)")
```

Figure 52: Visualising Snort's and models' runtime.

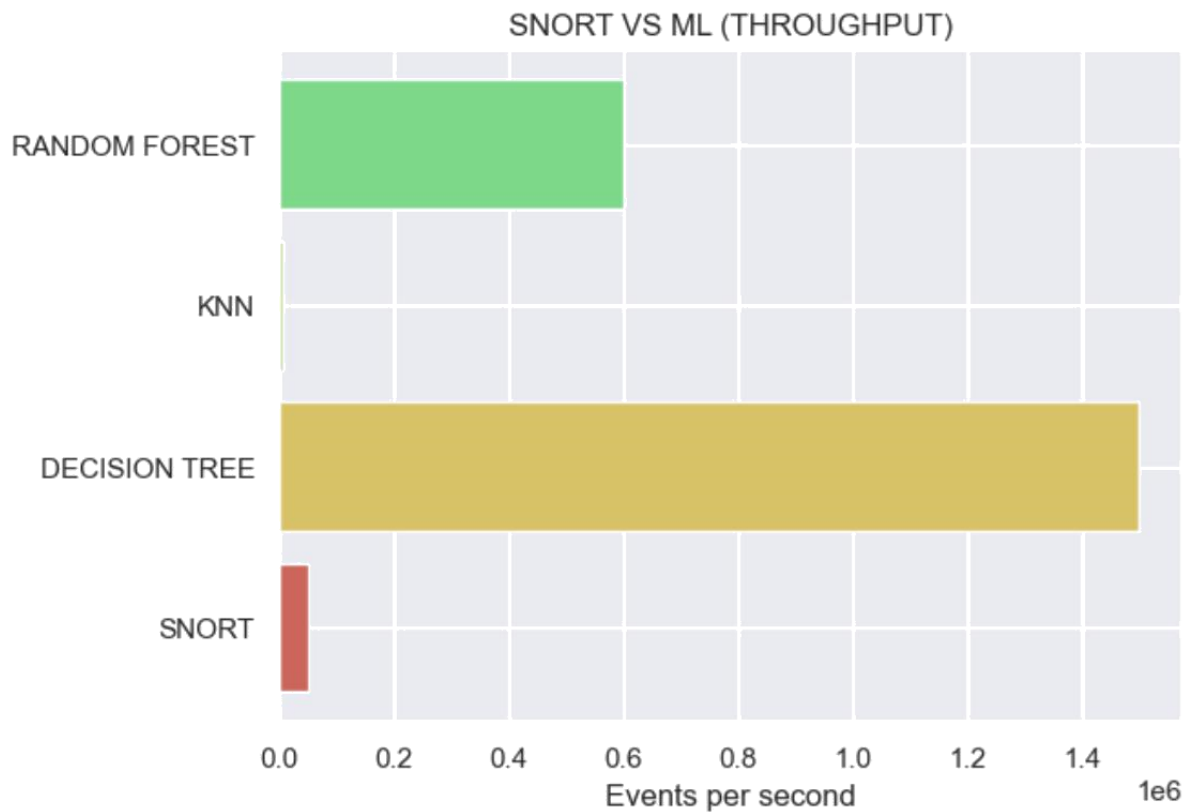


Figure 53: Snort's and ML throughput.

References

Sharafaldin, I., Lashkari, A.H. and Ghorbani, A.A. (2018) 'Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization', in 4th International Conference on Information Systems Security and Privacy (ICISSP). ICISSP 2018, Portugal.