VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE3045 - NETWORK SECURITY AND CRYPTOGRAPHY FUNDAMENTALS

PROJECT TITLE

ANOMALY - BASED INTRUSION DETECTION SYSTEM

SUBMITTED BY:

DEEKSHITHA L (19MIA1030)

P SUBHASHRI (19MIA1008)

SHIVANI GOKULRAM NAIDU (19MIA1006)

UNDER THE GUIDANCE

OF:

Dr. S. RENUKA DEVI

ABSTARCT

With the constant growth of the Internet, computer attacks are increasing not only in numbers but also in diversity: ransomware are on the rise like never before, and zeroday exploits become so critical that they are gaining media coverage. Antiviruses and firewalls are no longer sufficient to ensure the protection of a company network, which should be based on multiple layers of security. One of the most important layers, designed to protect its target against any potential attack through a continuous monitoring of the system, is provided by an Intrusion Detection System (IDS).

During the past two decades, much research has been focused on anomaly-based IDSs. Indeed, their ability to detect unknown attacks is significant in a context where attacks are becoming more numerous and diverse. An IDS is the high-tech equivalent of a burglar alarm, one that is configured to monitor Information gateways, hostile activities, and known intruders. There are two general approaches to detecting intrusions: anomaly detection (also called behaviour-based) and signature based (also named misuse or pattern based). Signature based techniques identify and store signature patterns of known intrusions. Pattern recognition techniques are efficient and accurate in detecting known intrusions but cannot detect novel intrusions whose signature patterns are unknown. Anomaly detection techniques can detect both novel and known attacks if they demonstrate large differences from the norm profile. Since anomaly detection techniques signal all anomalies as intrusions, false alarms are expected when anomalies are caused by behavioural irregularity instead of intrusions. Hence, pattern recognition techniques and anomaly detection techniques are often used together to complement each other. In this project, we will be configuring and using SNORT as an intrusion detection system; and also create machine learning models to predict the anomaly based intrusions.

INTRODUCTION

An Intruder is a hacker in network which always try to access security methods so that it performs unnecessary/unauthorized activities. The Attackers can be of two types are Active attackers and Passive attackers. The active attackers during attack make changes on network rules/ regulations. The Passive Attackers only see the network behaviour and does not make any changes. Inorder to overcome these threats, Intrusion detection systems are developed. Intrusion detection means detecting unauthorized use or attacks upon a System or Network.

Intrusion detection system (IDS) is a system that monitors and analyses data to detect any intrusion in the system or network. High volume, variety and high speed of data generated in the network have made the data analysis process to detect attacks by traditional techniques very difficult. Machine learning models can perform many tasks, three of which are particularly interesting for intrusion detection: classification, regression, and reconstruction. Classification categorizes entries into several classes, such as “normal” or “attack”, or even different families of attacks. Regression (also called “prediction”) is used to determine continuous values, including a probability that an input is an attack. Finally, reconstruction is specific to a certain type of neural network. This task tries to reconstruct the input data by compressing and decompressing them to force the network to learn the features (representation learning).

In our project, we have created and tested some Snort rules, which generates the alerts when something abnormal is happening. We are also creating machine learning models using historical data collected over a period of time. These models can predict if it's an attack or normal, given several information about the network. The best performing model can be saved for future analysis. In this work, we have used feature selection, and built an intrusion detection model by using Naive Baye Classifier, KNeighbors Classifier, Logistic Regression and Random Forest Classifier.

What is Intrusion Detection System?

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for harmful activity or policy breaching. Any malicious venture or violation is normally reported either to an administrator or collected centrally using a security information and event management (SIEM) system.

Types of IDS:

Depending upon the level of analysis IDS is classified into following types:

- Network intrusion detection system (NIDS)
- Host intrusion detection system (HIDS)
- Protocol-based intrusion detection system (PIDS)
- Application Protocol-based Intrusion Detection System (APIDS)
- Hybrid intrusion detection system

IDS RESPONSES AGAINST ATTACK

Whenever IDS detects any intrusions or attacks, it reacts as per the preconfigured settings. The responses can range from mere alert notifications to blocking of the attacks based on the severity. The appropriate reactions on the threats are a key issue for safety and efficacy. Generally, the responses can be of three types

i. Active response:

IDS by itself cannot block attacks, however, can take such actions which can lead to stopping of attacks. Such actions can be for example, sending TCP reset packets to the machine(s) which is being the target of attack, reconfiguring router/firewall as to block the malicious connection. In extreme cases, IDS can even block all the network traffic to avoid potential damage to the firm.

ii. Passive response:

Passive solutions deliver information to IDS administrator on the current situation and leave the decision to take appropriate steps to his discretion. Many commercial systems rely on this kind of reactions. Examples for this kind of actions can be simple alarm messages and notifications. Notifications can be sent on email, cellular phone or via SNMP messages.

iii. Mixed response:

Mixed responses combine both active as well as the passive responses appropriately as per the needs of situation.

OPERATING SYSTEM:

1. Windows 10
2. Linux VMware
3. Ubuntu VMware

SOFTWARE USED:

1. Snort
2. WinPcap
3. Wireshark
4. Jupyter Notebook

1. SNORT:

SNORT is a libpcap based lightweight network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It is an open source ids, which can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, OS fingerprinting attempts, and much more.

Snort can be configured by three main modes.

- **Sniffer:** In this mode, SNORT simply eavesdrop the packets and displays them like tcpdump program. Depending on the flags used with SNORT, we can determine how detailed information we want to avail.

- **Packet logger:** Whenever the SNORT user wants to record the packets captured by the IDS, SNORT has to be run in the Packet logger mode, specifying the directory name where the packets are to be logged. It logs packets either in tcpdump format (binary) or in decoded ASCII format.

- **Intrusion Detection mode:** In this mode, SNORT will not record every packet that it sniffs but logs only those events which triggered its rules

2. WIRESHARK:

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets; it runs on Linux, macOS, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. Wireshark is a network protocol analyzer, or an application that captures packets from a network connection, such as from your computer to your home office or the internet.

Wireshark does three things:

i. **Packet Capture:** Wireshark listens to a network connection in real time and then grabs entire streams of traffic – quite possibly tens of thousands of packets at a time.

ii. **Filtering:** Wireshark is capable of slicing and dicing all of this random live data using filters. By applying a filter, we can obtain just the information you need to see.

iii. **Visualization:** Wireshark also allows us to visualize entire conversations and network streams.

3. Jupyter Notebook

The Jupyter Notebook is a web application for creating and sharing documents that contain code, visualizations, and text. It can be used for data science, statistical modeling, machine learning, and much more.

IMPLEMENTATION

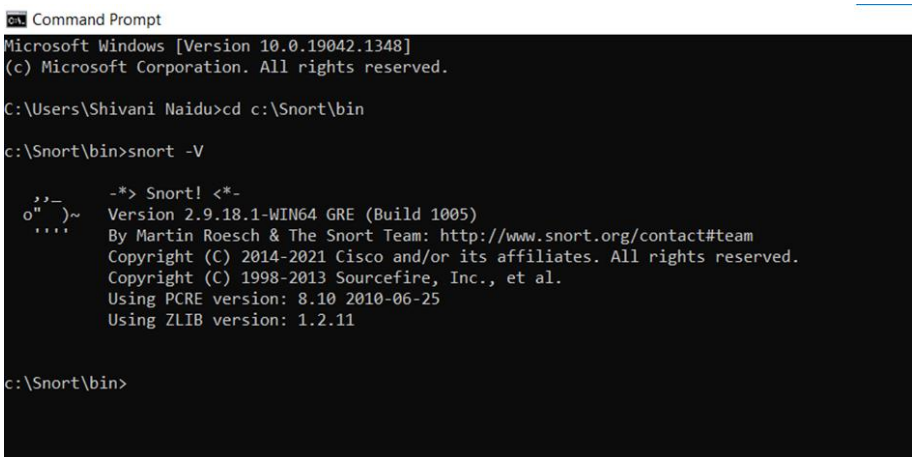
❖ WORKING OF SNORT:

Snort is the most widely deployed IDS/IPS technology worldwide, as it has the combining benefits of signature, protocol, and anomaly-based inspection.

Snort uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes a modular plug-in architecture. Snort has a real-time alerting capability as well, with alerts being sent to syslog, a separate "alert" file.

Steps involved in implementing Snort as Intrusion Detection System (IDS)

- Necessary softwares like WinPcap and Snort are installed on Windows OS.

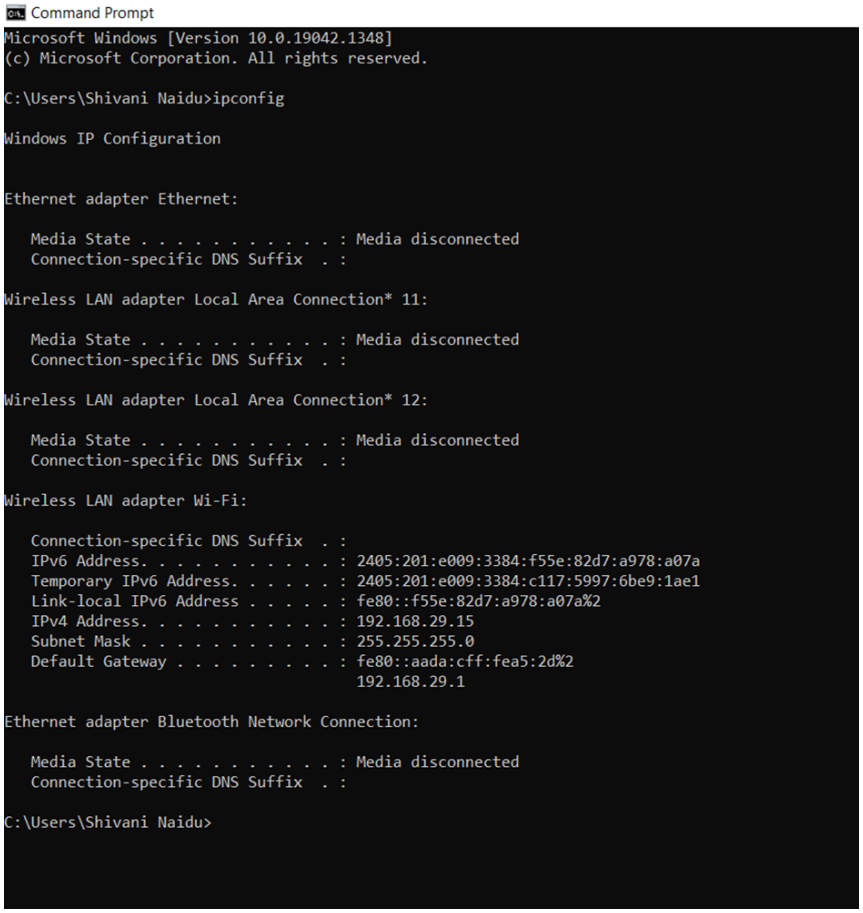


Verifying the Snort version.

- Configuring Snort with Windows:

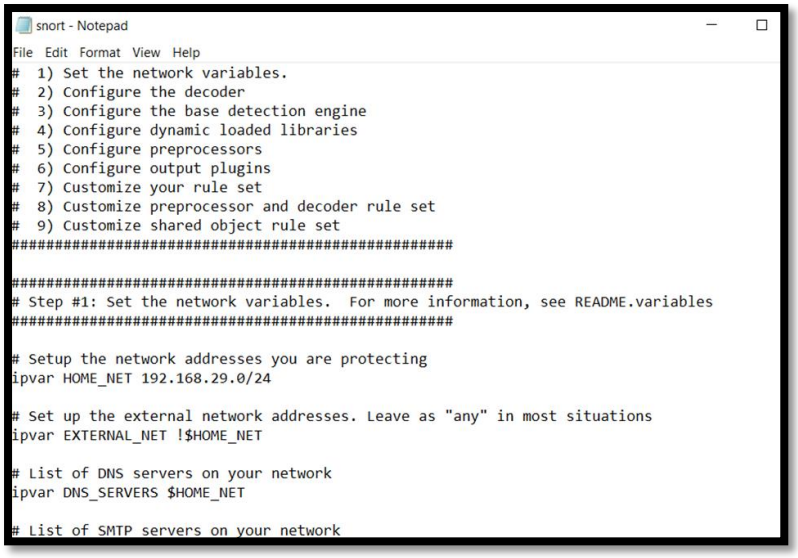
  We are configuring our HOME_NET value, the network we wanted to protect.

Getting IPv4 address using “ipconfig” command



In the snort.conf file, the **ipvar HOME_NET** settings are updated by changing the IP address to be actual class C subnet and save the file.

Now, Snort is ready to run.



Changing the IP address part to match our Windows Server IP.

To run Snort:

```
snort -T -i eth0 -c c:\Snort\etc\snort.conf
```

Here we are telling Snort to test (-T) the configuration file (-c points to its location) on the eth0 interface. This will produce a lot of output.

```
++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
0 Snort rules read
    0 detection rules
    0 decoder rules
    0 preprocessor rules
0 Option Chains linked into 0 Chain Headers
0 Dynamic rules
++++++++++++++++++++++++++++++++++++++++++++++++++
```

Snort in IDS mode:

Now, starting Snort in IDS mode to display alerts to the console:

```
snort -A console -q -c c:\Snort\etc\snort.conf -i eht0
```

Here, we are pointing Snort to the configuration file it should use (**-c**) and specifying the interface (**-i eth0**). The **-A** console option prints alerts to standard output, and **-q** is for "quiet" mode.

Checking if our rule is working:

Pinging the Windows Server from KALI LINUX VM

```
root@attackserver:~# ping 192.168.132.130
PING 192.168.132.130 (192.168.132.130) 56(84) bytes of data.
64 bytes from 192.168.132.130: icmp_req=1 ttl=64 time=1.32 ms
64 bytes from 192.168.132.130: icmp_req=2 ttl=64 time=0.949 ms
64 bytes from 192.168.132.130: icmp_req=3 ttl=64 time=0.906 ms
64 bytes from 192.168.132.130: icmp_req=4 ttl=64 time=1.35 ms
64 bytes from 192.168.132.130: icmp_req=5 ttl=64 time=0.877 ms
^C
--- 192.168.132.130 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4010ms
rtt min/avg/max/mdev = 0.877/1.082/1.353/0.212 ms
root@attackserver:~#
```

See alerts generated for every ICMP Echo request and Echo reply message, with the message text we specified in the **msg** option:

```
stu@ubuntu:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i eth0
02/22-13:07:06.471944  [**] [1:1000001:1] ICMP test [**] [Classification: Generi
c ICMP event] [Priority: 3] {ICMP} 192.168.132.133 -> 192.168.132.130
02/22-13:07:06.472059  [**] [1:1000001:1] ICMP test [**] [Classification: Generi
c ICMP event] [Priority: 3] {ICMP} 192.168.132.130 -> 192.168.132.133
02/22-13:07:07.474545  [**] [1:1000001:1] ICMP test [**] [Classification: Generi
c ICMP event] [Priority: 3] {ICMP} 192.168.132.133 -> 192.168.132.130
02/22-13:07:07.474604  [**] [1:1000001:1] ICMP test [**] [Classification: Generi
c ICMP event] [Priority: 3] {ICMP} 192.168.132.130 -> 192.168.132.133
02/22-13:07:08.477023  [**] [1:1000001:1] ICMP test [**] [Classification: Generi
c ICMP event] [Priority: 3] {ICMP} 192.168.132.133 -> 192.168.132.130
```

We can also see the source IP address of the host responsible for the alert-generating activity.

Writing the Snort rule:

RULE:

```
alert tcp 192.168.x.x any -> $HOME_NET 21 (msg:"FTP connection attempt"; sid:1000002; rev:1;)
```

local.rules

```
#alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:1000001;
rev:1; classtype:icmp-event;)
alert tcp 192.168.132.133 any -> $HOME_NET 21 (msg:"FTP connection
attempt"; sid:1000002; rev:1;)
```

Here we changed the protocol to TCP, used a specific source IP, set the destination port number to 21 (default port for FTP connections) and changed the alert message text.

Now, running Snort in IDS mode again with one more option, telling Snort to log generated alerts in the ASCII format rather than the default pcap.

COMMAND:

```
sudo snort -A console -q -c /etc/snort/snort.conf -i eht0 -K ascii
```

```
root@attackserver:~# ftp 192.168.132.130
ftp: connect: Connection refused
ftp>
```

```
stu@ubuntu:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i eth0 -K ascii
02/22-14:09:47.453755  [**] [1:1000002:1] FTP connection attempt [**] [Priority:
 0] {TCP} 192.168.132.133:45562 -> 192.168.132.130:21
```

We can see that an alert has been generated.

- Run the following command to do the listing of the Snort log directory:

```
dir C:\Snort\log\
```

Command Prompt

```
icrosoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. All rights reserved.

:\Users\Shivani Naidu>dir C:\Snort\log\
 Volume in drive C is OS
 Volume Serial Number is 9E0C-63C9

 Directory of C:\Snort\log

)3-12-2021  13:29    <DIR>          .
)3-12-2021  13:29    <DIR>          ..
)3-12-2021  13:25    <DIR>          13.69.109.130
)3-12-2021  12:53    <DIR>          13.89.178.27
)3-12-2021  12:12    <DIR>          192.168.29.1
)3-12-2021  13:29    <DIR>          192.168.29.15
)3-12-2021  12:12    <DIR>          192.168.29.196
)3-12-2021  12:12    <DIR>          192.168.29.99
)3-12-2021  13:28    <DIR>          20.44.239.154
)3-12-2021  12:50    <DIR>          20.50.73.10
)3-12-2021  13:29    <DIR>          40.70.229.150
)3-12-2021  12:55    <DIR>          51.105.71.136
)3-12-2021  13:14    <DIR>          52.113.206.194
)3-12-2021  12:51    <DIR>          52.114.12.0
)3-12-2021  13:26    <DIR>          52.114.12.142
)3-12-2021  13:26    <DIR>          52.114.142.202
)3-12-2021  13:29    <DIR>          52.114.40.82
)3-12-2021  13:25    <DIR>          52.182.143.211
)1-12-2021  21:17                 0 alert.ids
4-10-2021  21:39                24 snort.log.1634227763
4-10-2021  21:52                24 snort.log.1634228575
```

dir C:\Snort\log\192.168.29.15\

Command Prompt

```
03-12-2021  13:06            35,229 snort.log.1638516898
03-12-2021  13:10            12,918 snort.log.1638517130
03-12-2021  13:23             2,304 snort.log.1638518002
              26 File(s)    111,464,094 bytes
              18 Dir(s)  164,804,878,336 bytes free

C:\Users\Shivani Naidu>

C:\Users\Shivani Naidu>dir C:\Snort\log\192.168.29.15\
 Volume in drive C is OS
 Volume Serial Number is 9E0C-63C9

 Directory of C:\Snort\log\192.168.29.15

03-12-2021  13:29    <DIR>          .
03-12-2021  13:29    <DIR>          ..
30-11-2021  23:12             1,745 TCP_51913-21.ids
03-12-2021  13:25             1,260 TCP_63658-443.ids
03-12-2021  12:48               628 TCP_63659-443.ids
03-12-2021  13:26               643 TCP_63687-443.ids
03-12-2021  12:14               316 TCP_63827-443.ids
03-12-2021  13:14               942 TCP_64044-443.ids
03-12-2021  12:51               603 TCP_64286-443.ids
03-12-2021  12:51               314 TCP_64287-443.ids
03-12-2021  12:51               315 TCP_64288-443.ids
03-12-2021  12:53               315 TCP_64302-443.ids
03-12-2021  12:53               315 TCP_64303-443.ids
03-12-2021  12:55               316 TCP_64311-443.ids
03-12-2021  13:14               319 TCP_64408-443.ids
03-12-2021  13:14               349 TCP_64415-21.ids
```

type C:\Snort\log\192.168.29.15\TCP_51913-21.ids

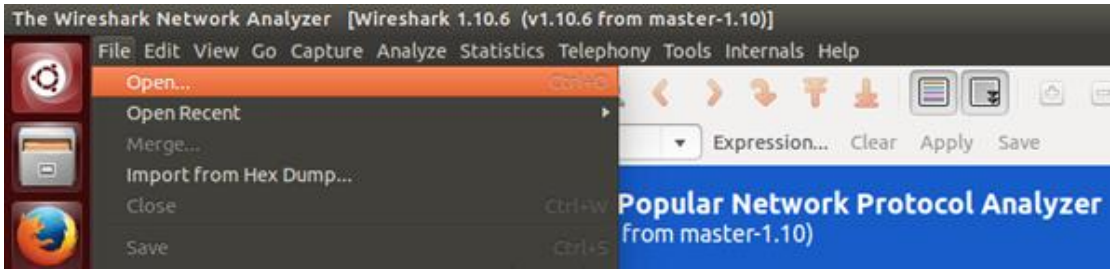
Command Prompt

```
03-12-2021  13:25               317 TCP_64502-443.ids
03-12-2021  13:25               317 TCP_64503-443.ids
03-12-2021  13:26               317 TCP_64511-443.ids
03-12-2021  13:26               316 TCP_64512-443.ids
03-12-2021  13:28               309 TCP_64517-443.ids
03-12-2021  13:29               316 TCP_64518-443.ids
03-12-2021  13:29               314 TCP_64519-443.ids
              25 File(s)         12,233 bytes
               2 Dir(s)  164,801,097,728 bytes free

C:\Users\Shivani Naidu>type C:\Snort\log\192.168.29.15\TCP_51913-21.ids
[**] ГÇ¥FTP connection attemptГÇ¥ [**]
11/30-23:12:03.901126 192.168.29.15:51913 -> 192.168.29.1:21
TCP TTL:128 TOS:0x0 ID:64417 IpLen:20 DgmLen:52 DF
******S* Seq: 0xDF2F3D1D  Ack: 0x0  Win: 0xFAF0  TcpLen: 32
TCP Options (6) => MSS: 1460 NOP WS: 8 NOP NOP SackOK
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] ГÇ¥FTP connection attemptГÇ¥ [**]
11/30-23:12:04.901920 192.168.29.15:51913 -> 192.168.29.1:21
TCP TTL:128 TOS:0x0 ID:64418 IpLen:20 DgmLen:52 DF
******S* Seq: 0xDF2F3D1D  Ack: 0x0  Win: 0xFAF0  TcpLen: 32
TCP Options (6) => MSS: 1460 NOP WS: 8 NOP NOP SackOK
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] ГÇ¥FTP connection attemptГÇ¥ [**]
11/30-23:12:06.902016 192.168.29.15:51913 -> 192.168.29.1:21
TCP TTL:128 TOS:0x0 ID:64419 IpLen:20 DgmLen:52 DF
******S* Seq: 0xDF2F3D1D  Ack: 0x0  Win: 0xFAF0  TcpLen: 32
TCP Options (6) => MSS: 1460 NOP WS: 8 NOP NOP SackOK
```

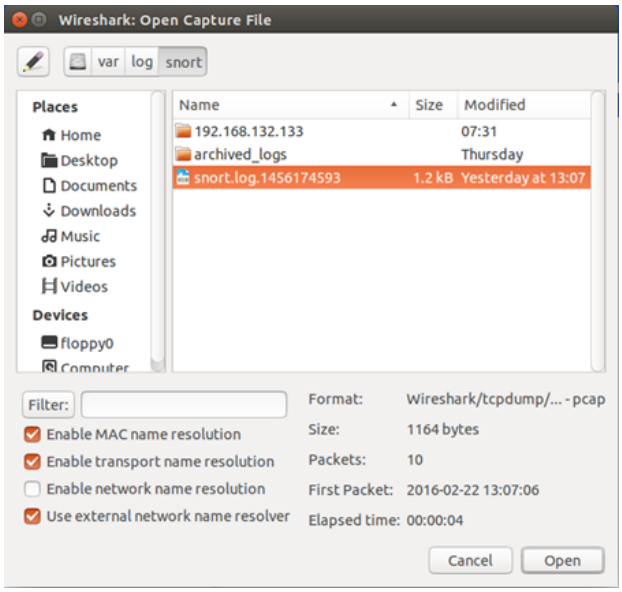
Thus Snort alerts us by generating messages when an intrusion happens.

USING WIRESHARK TO CAPTURE THE PACKETS & ANALYSE NETWORK TRAFFIC

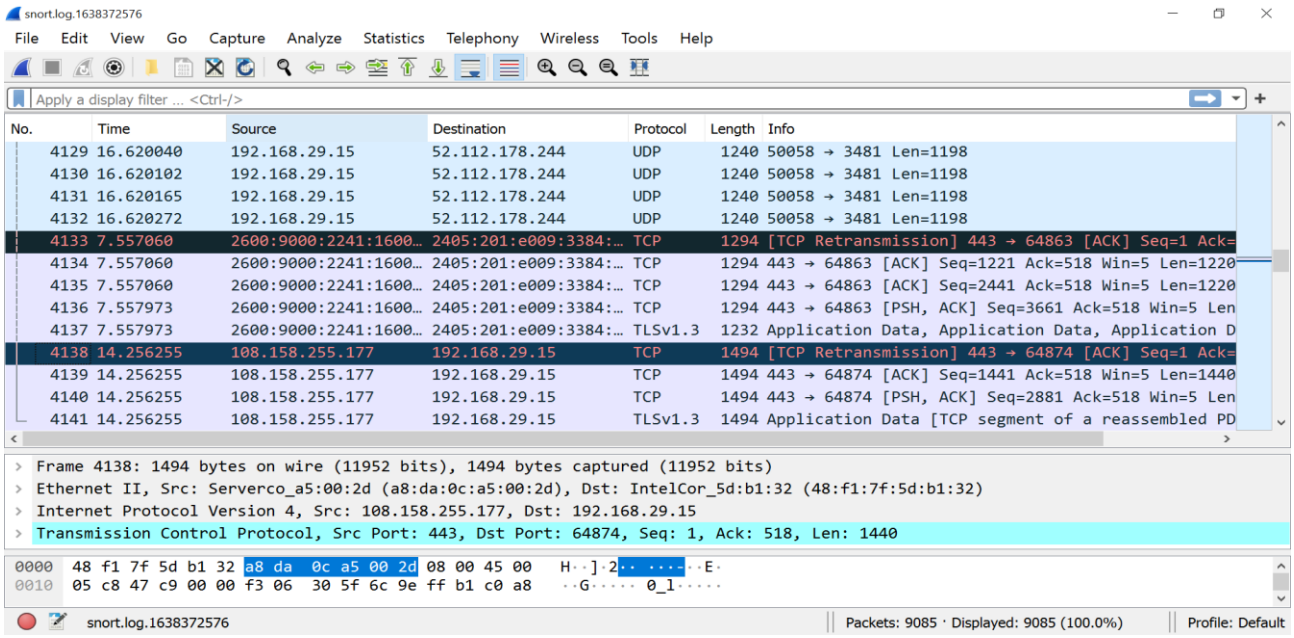
At the Wireshark main window, go to File → Open.

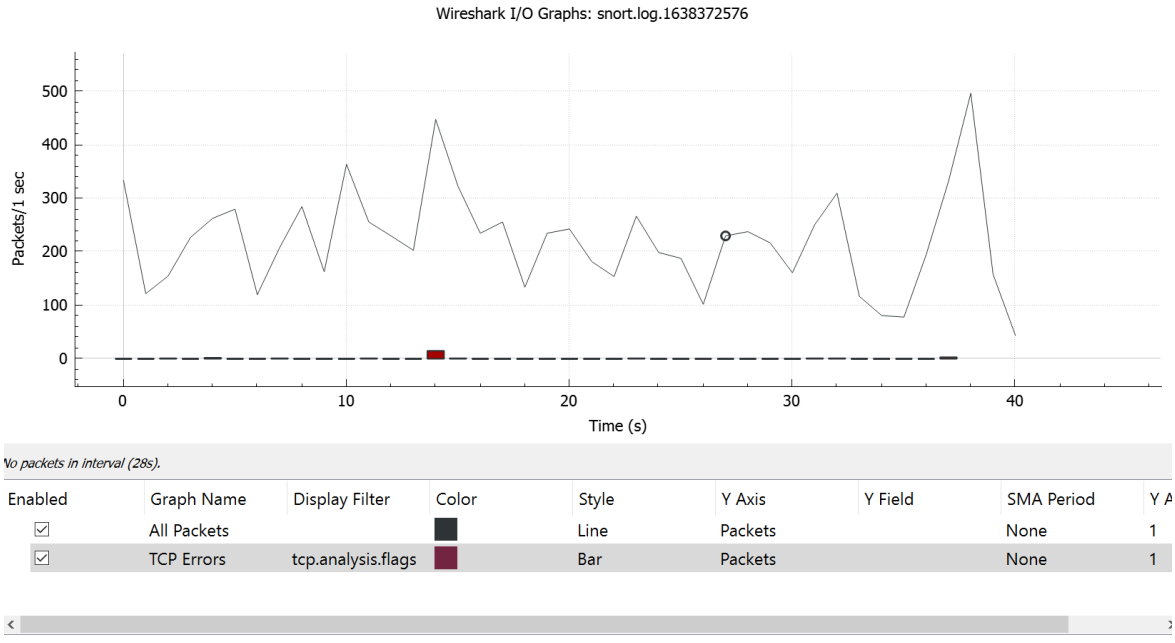


Browse to the snort directory, select the snort.log.* file and click Open.



We can look at the contents of each packet.





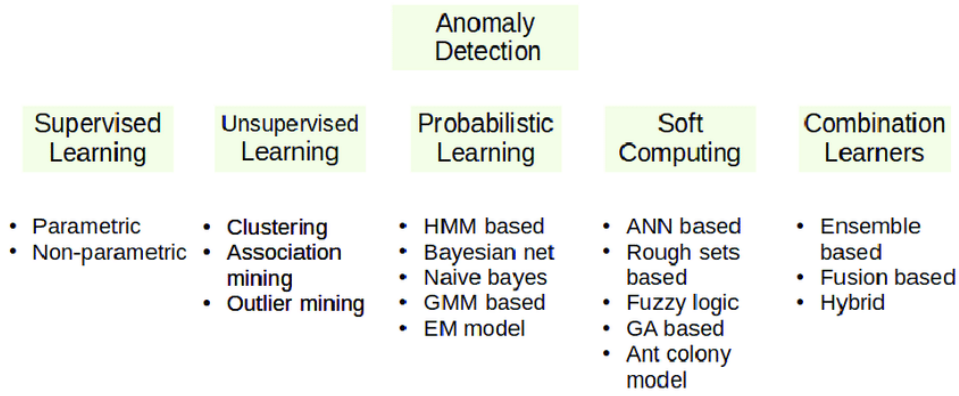
With I/O graph we can interpret the progress of packets and the network traffic.

With the help of this tool, we come to know the normal behaviour of the network. This can be used to create a probability table that helps us to detect the anomalies in the network, as an anomaly detection approach uses normal traffic and signals and any deviation from this is detected as suspicious.

❖ MACHINE LEARNING TECHNIQUES FOR ANOMALY DETECTION:

In the IDS/NIDS systems a statistical based anomaly detection technique is used to represent the expected normal behaviour of a subject and variance due to noises. The statistical-based anomaly detection technique overcomes the problems with rule-based anomaly detection technique in handling noises and variances. However, the statistical technique in IDS/NIDS is a univariate technique that is applied to only one behaviour measure, where as many intrusions involve multiple subjects and multiple actions having impact on multiple behaviour measures. Hence, a multivariate anomaly detection technique is needed for intrusion detection. Network anomaly detectors look for unusual traffic rather than unusual system calls. ADAM (Audit Data and Mining) is an anomaly detector trained on both attack-free traffic and traffic with labelled attacks.

Recently, there exist numerous approaches for the classification of machine learning methods for IDS design. A broad taxonomy suggests five ML methods for anomaly-based intrusion detection, as shown in Figure below.



Machine learning methods for anomaly-based intrusion detection.

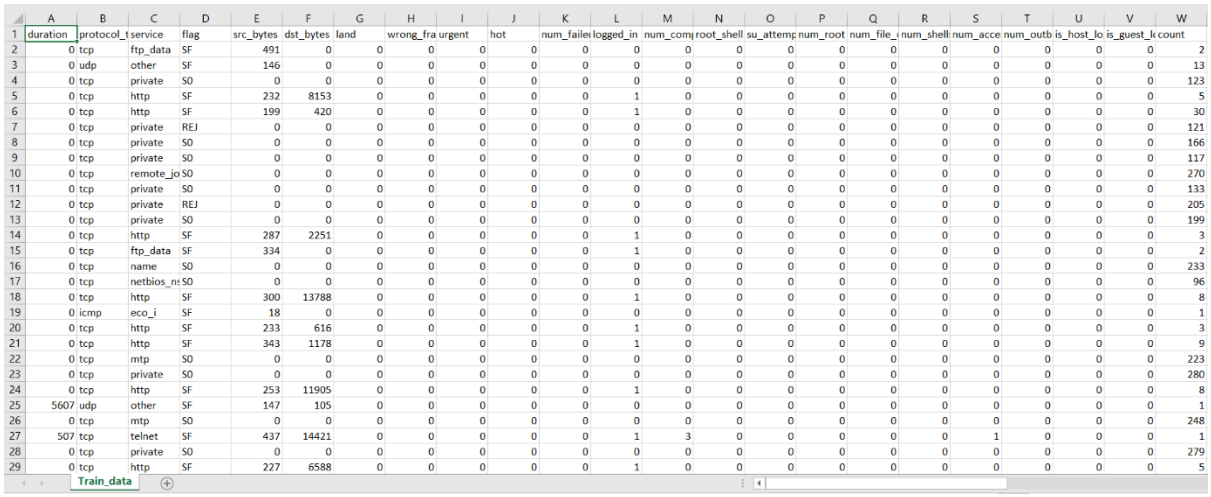
ABOUT THE DATASET:

In this project, we are using KDD dataset. The datasets contain a total of 24 training attack types, with an additional 14 types in the test data only.

Dataset Description:

Attribute Name	Description	Type
duration	length (number of seconds) of the connection	continuous
protocol type	type of the protocol, e.g., tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete
count	number of connections to the same host as the current connection in the past two seconds	continuous
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Screenshots:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	duration	protocol_t	service	flag	src_bytes	dst_bytes	land	wrong_fra	urgent	hot	num_faile	logged_in	num_com	root_shell	su_attemp	num_root	num_file_	num_shell	num_acce	num_outb	is_host_lo	is_guest_l	count
2	0	tcp	ftp_data	SF	491	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
3	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13
4	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	123
5	0	tcp	http	SF	232	8153	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	5
6	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	30
7	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	121
8	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	166
9	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	117
10	0	tcp	remote_jo	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	270
11	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	133
12	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	205
13	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	199
14	0	tcp	http	SF	287	2251	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3
15	0	tcp	ftp_data	SF	334	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2
16	0	tcp	name	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	233
17	0	tcp	netbios_ns	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96
18	0	tcp	http	SF	300	13788	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	8
19	0	icmp	eco_i	SF	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
20	0	tcp	http	SF	233	616	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3
21	0	tcp	http	SF	343	1178	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	9
22	0	tcp	mtp	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	223
23	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	280
24	0	tcp	http	SF	253	11905	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	8
25	5607	udp	other	SF	147	105	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
26	0	tcp	mtp	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	248
27	507	tcp	telnet	SF	437	14421	0	0	0	0	0	1	3	0	0	0	0	0	1	0	0	0	1
28	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	279
29	0	tcp	http	SF	227	6588	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	5

Train_data

X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP
srv_count	serror_rati	srv_serror	rerror_rat	srv_rerror	same_srv_	diff_srv_ra	srv_diff_h	dst_host_c	dst_host_s	dst_host_s	dst_host_c	dst_host_s	dst_host_s	dst_host_s	dst_host_s	dst_host_r	dst_host_s	class
2	0	0	0	0	1	0	0	150	25	0.17	0.03	0.17	0	0	0	0.05	0	normal
1	0	0	0	0	0.08	0.15	0	255	1	0	0.6	0.88	0	0	0	0	0	normal
6	1	1	0	0	0.05	0.07	0	255	26	0.1	0.05	0	0	1	1	0	0	anomaly
5	0.2	0.2	0	0	1	0	0	30	255	1	0	0.03	0.04	0.03	0.01	0	0.01	normal
32	0	0	0	0	1	0	0.09	255	255	1	0	0	0	0	0	0	0	normal
19	0	0	1	1	0.16	0.06	0	255	19	0.07	0.07	0	0	0	0	1	1	anomaly
9	1	1	0	0	0.05	0.06	0	255	9	0.04	0.05	0	0	1	1	0	0	anomaly
16	1	1	0	0	0.14	0.06	0	255	15	0.06	0.07	0	0	1	1	0	0	anomaly
23	1	1	0	0	0.09	0.05	0	255	23	0.09	0.05	0	0	1	1	0	0	anomaly
8	1	1	0	0	0.06	0.06	0	255	13	0.05	0.06	0	0	1	1	0	0	anomaly
12	0	0	1	1	0.06	0.06	0	255	12	0.05	0.07	0	0	0	0	1	1	anomaly
3	1	1	0	0	0.02	0.06	0	255	13	0.05	0.07	0	0	1	1	0	0	anomaly
7	0	0	0	0	1	0	0.43	8	219	1	0	0.12	0.03	0	0	0	0	normal
2	0	0	0	0	1	0	0	2	20	1	0	1	0.2	0	0	0	0	anomaly
1	1	1	0	0	0	0.06	0	255	1	0	0.07	0	0	1	1	0	0	anomaly
16	1	1	0	0	0.17	0.05	0	255	2	0.01	0.06	0	0	1	1	0	0	anomaly
9	0	0.11	0	0	1	0	0.22	91	255	1	0	0.01	0.02	0	0	0	0	normal
1	0	0	0	0	1	0	0	1	16	1	0	1	1	0	0	0	0	anomaly
3	0	0	0	0	1	0	0	66	255	1	0	0.02	0.03	0	0	0.02	0	normal
10	0	0	0	0	1	0	0.2	157	255	1	0	0.01	0.04	0	0	0	0	normal
23	1	1	0	0	0.1	0.05	0	255	23	0.09	0.05	0	0	1	1	0	0	anomaly
17	1	1	0	0	0.06	0.05	0	238	17	0.07	0.06	0	0	0.99	1	0	0	anomaly
10	0	0	0	0	1	0	0.2	87	255	1	0	0.01	0.02	0	0	0	0	normal
1	0	0	0	0	1	0	0	255	1	0	0.85	1	0	0	0	0	0	normal
2	1	1	0	0	0.01	0.06	0	255	2	0.01	0.06	0	0	1	1	0	0	anomaly
1	0	0	0	0	1	0	0	255	25	0.1	0.05	0	0	0.53	0	0.02	0.16	normal
7	1	1	0	0	0.03	0.06	0	255	13	0.05	0.07	0	0	1	1	0	0	anomaly
22	0	0	0	0	1	0	0.18	43	255	1	0	0.02	0.14	0	0	0.56	0.57	normal

METHODOLOGY:

IMPORTING THE DEPENDENCIES:

First, imported the required libraries. The dataset chosen is a clean dataset. Thus printed the information of the whole dataset. Here we could see that there are no null values, since it is a clean dataset.

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25192 entries, 0 to 25191
Data columns (total 42 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   duration                     25192 non-null  int64
 1   protocol_type                25192 non-null  object
 2   service                      25192 non-null  object
 3   flag                         25192 non-null  object
 4   src_bytes                    25192 non-null  int64
 5   dst_bytes                    25192 non-null  int64
 6   land                         25192 non-null  int64
 7   wrong_fragment               25192 non-null  int64
 8   urgent                       25192 non-null  int64
 9   hot                          25192 non-null  int64
 10  num_failed_logins            25192 non-null  int64
 11  logged_in                    25192 non-null  int64
 12  num_compromised              25192 non-null  int64
 13  root_shell                   25192 non-null  int64
 14  su_attempted                 25192 non-null  int64
 15  num_root                     25192 non-null  int64
 16  num_file_creations           25192 non-null  int64
 17  num_shells                   25192 non-null  int64
 18  num_access_files             25192 non-null  int64
 19  num_outbound_cmds            25192 non-null  int64
 20  is_host_login                25192 non-null  int64
 21  is_guest_login               25192 non-null  int64
 22  count                        25192 non-null  int64
 23  srv_count                    25192 non-null  int64
 24  serror_rate                  25192 non-null  float64
 25  srv_serror_rate              25192 non-null  float64
 26  rerror_rate                  25192 non-null  float64
 27  srv_rerror_rate              25192 non-null  float64
 28  same_srv_rate                25192 non-null  float64
 29  diff_srv_rate                25192 non-null  float64
 30  srv_diff_host_rate           25192 non-null  float64
 31  dst_host_count               25192 non-null  int64
 32  dst_host_srv_count           25192 non-null  int64
 33  dst_host_same_srv_rate       25192 non-null  float64
 34  dst_host_diff_srv_rate       25192 non-null  float64
 35  dst_host_same_src_port_rate  25192 non-null  float64
 36  dst_host_srv_diff_host_rate  25192 non-null  float64
 37  dst_host_serror_rate         25192 non-null  float64
 38  dst_host_srv_serror_rate     25192 non-null  float64
 39  dst_host_rerror_rate         25192 non-null  float64
 40  dst_host_srv_rerror_rate     25192 non-null  float64
 41  class                        25192 non-null  object
dtypes: float64(15), int64(23), object(4)
memory usage: 8.1+ MB
```

PREPROCESSING:

At this stage, two main steps were executed with the help of pandas and numpy python libraries.

- Redundant/unwanted columns were dropped.
- Categorical features were transformed into numerical ones, which is called encoding.

HANDLING CATEGORICAL DATA - Encoding

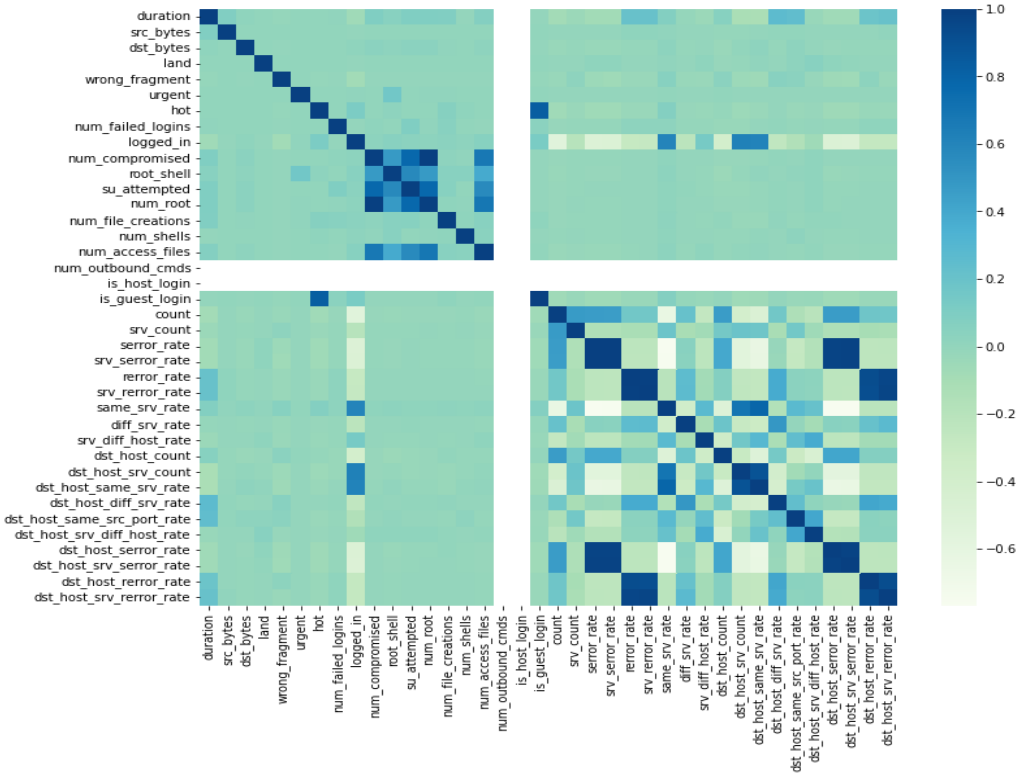
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

# extract categorical attributes from both training and test sets
cattrain = train_df.select_dtypes(include=['object']).copy()
cattest = test_df.select_dtypes(include=['object']).copy()

# encode the categorical attributes
traincat = cattrain.apply(encoder.fit_transform)
testcat = cattest.apply(encoder.fit_transform)

# separate target column from encoded data
enctrain = traincat.drop(['class'], axis=1)
cat_Ytrain = traincat[['class']].copy()
```

FEATURE SELECTION:

Feature selection is a critical step that affects the ML model performance directly. Reducing the number of features has two main benefits in developing machine learning models, like helps to build a more accurate model by removing the noise caused by unnecessary features and also decreases model training time.

Here feature selection is done using filter method, which measures each feature's correlation (independent variable) and the target (dependent) variable separately. The most prominent advantages of the filter method feature selection are that they are not computationally expensive and less prone to over-fitting.

HEATMAP:



From the heatmap, we can notice that correlation for 'num_outbounds_cmds' with other features is constant. So we are dropping that column from both the train and test set.

```
train_df['num_outbound_cmds'].unique()
```

```
array([0])
```

```
train_df.drop('num_outbound_cmds', axis = 1, inplace = True)
```

```
test_df.drop('num_outbound_cmds', axis = 1, inplace = True)
```

FEATURE SELECTION

```
highly_correlated_df = (correlation.abs() > 0.9) & (correlation.abs() < 1.0)
```

```
[19] corr_var_index = (highly_correlated_df == True).any()
     corr_var_names = corr_var_index[corr_var_index == True].index

     de_duplicate = []
     corr_pairs = []

     for i in corr_var_index.index:
         row = highly_correlated_df[i]
         de_duplicate.append(i)
         for j in corr_var_names:
             if j not in de_duplicate and row[j] == True:
                 print(i,"&",j,": ", correlation.loc[i,j])
                 corr_pairs.append((i,j))
```

```
num_compromised & num_root :  0.9989564860270104
serror_rate & srv_serror_rate :  0.9932900594139392
serror_rate & dst_host_serror_rate :  0.978021838092273
serror_rate & dst_host_srv_serror_rate :  0.979601949917278
srv_serror_rate & dst_host_serror_rate :  0.9763633089078823
srv_serror_rate & dst_host_srv_serror_rate :  0.9846211977178592
rerror_rate & srv_rerror_rate :  0.9891336910125826
rerror_rate & dst_host_rerror_rate :  0.9289919679471647
rerror_rate & dst_host_srv_rerror_rate :  0.964886035790209
srv_rerror_rate & dst_host_rerror_rate :  0.9195367413800786
srv_rerror_rate & dst_host_srv_rerror_rate :  0.9698414193761191
dst_host_serror_rate & dst_host_srv_serror_rate :  0.9847058568468164
dst_host_rerror_rate & dst_host_srv_rerror_rate :  0.9253904301448883
```

By analysing the correlation values, the features with little or no variation are dropped.

Columns dropped:
'num_root', 'srv_serror_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate'

Both the both train and test data are scaled using StandardScaler.

MODEL TRAINING AND TESTING

This phase consisted of the development of machine learning models.

After the evaluation of those models, they were applied to the data set for comparison.

Ensemble learning methods are meta-learners that utilize multiple ML models on the same data set to get more accurate results to reduce bias, noise, and variance.

As the base classifier, we used Naive Bayes, KNN, logistic regression, and for ensemble learning model we used Random Forest.

```
from sklearn.svm import SVC
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
```

❖ **Naive Bayes Classifier:**

```
from sklearn.naive_bayes import BernoulliNB

# Train Gaussian Naive Baye Model
BNB_Classifier = BernoulliNB()
BNB_Classifier.fit(X_train, Y_train)
```

❖ **KNeighbors Classifier:**

```
# Train KNeighborsClassifier Model
KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
KNN_Classifier.fit(X_train, Y_train);
```

❖ **Logistic Regression:**

```
# Train LogisticRegression Model
LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0)
LGR_Classifier.fit(X_train, Y_train);
```

❖ **Random Forest Classifier:**

```
from sklearn.ensemble import RandomForestClassifier

# Train  Random Forest Model
RFC_Classifier = RandomForestClassifier()
RFC_Classifier.fit(X_train, Y_train)
```

MODEL EVALUATION:

```
from sklearn import metrics

models = []
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('LogisticRegression', LGR_Classifier))
models.append(('Random Forest Classifier', RFC_Classifier))

for i, v in models:
    scores = cross_val_score(v, X_train, Y_train, cv=10)
    accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
    confusion_matrix = metrics.confusion_matrix(Y_train, v.predict(X_train))
    classification = metrics.classification_report(Y_train, v.predict(X_train))
    print()
    print('============================== {} Model Evaluation ==============================='.format(i))
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

The Accuracy, Cross validation scores and the classification report of each models are given below.

```
============================== Naive Baye Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.8918282523927685

Model Accuracy:
 0.8919267602838287

Confusion matrix:
 [[ 7966  1360]
 [  818 10009]]

Classification report:
               precision    recall  f1-score   support

      anomaly       0.91      0.85      0.88      9326
       normal       0.88      0.92      0.90     10827

    accuracy                           0.89     20153
   macro avg       0.89      0.89      0.89     20153
weighted avg       0.89      0.89      0.89     20153
```

```
============================== KNeighborsClassifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.991663835716255

Model Accuracy:
 0.9941944127425197

Confusion matrix:
 [[ 9254    72]
 [   45 10782]]

Classification report:
               precision    recall  f1-score   support

      anomaly       1.00      0.99      0.99      9326
       normal       0.99      1.00      0.99     10827

     accuracy                           0.99     20153
    macro avg       0.99      0.99      0.99     20153
 weighted avg       0.99      0.99      0.99     20153
```

```
============================== LogisticRegression Model Evaluation ==============================

Cross Validation Mean Score:
 0.954200441135925

Model Accuracy:
 0.9545973304222697

Confusion matrix:
 [[ 8769   557]
 [  358 10469]]

Classification report:
               precision    recall  f1-score   support

      anomaly       0.96      0.94      0.95      9326
       normal       0.95      0.97      0.96     10827

     accuracy                           0.95     20153
    macro avg       0.96      0.95      0.95     20153
 weighted avg       0.95      0.95      0.95     20153
```

```
============================== Random Forest Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.996973369372563

Model Accuracy:
 0.9999503795960899

Confusion matrix:
 [[ 9325     1]
 [    0 10827]]

Classification report:
               precision    recall  f1-score   support

      anomaly       1.00      1.00      1.00      9326
       normal       1.00      1.00      1.00     10827

     accuracy                           1.00     20153
    macro avg       1.00      1.00      1.00     20153
 weighted avg       1.00      1.00      1.00     20153
```

TESTING THE TRAINED MODELS:

Finally, trained models were tested for accuracy. The results are presented in the following section.

```
for i, v in models:
    accuracy = metrics.accuracy_score(Y_test, v.predict(X_test))
    confusion_matrix = metrics.confusion_matrix(Y_test, v.predict(X_test))
    classification = metrics.classification_report(Y_test, v.predict(X_test))
    print()
    print('============================== {} Model Test Results ==============================='.format(i))
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

```
============================== Naive Baye Classifier Model Test Results ==============================

Model Accuracy:
 0.8960111133161341

Confusion matrix:
 [[2080  337]
 [ 187 2435]]

Classification report:
               precision    recall  f1-score   support

     anomaly       0.92      0.86      0.89      2417
      normal       0.88      0.93      0.90      2622

    accuracy                           0.90      5039
   macro avg       0.90      0.89      0.90      5039
weighted avg       0.90      0.90      0.90      5039
```

```
============================== KNeighborsClassifier Model Test Results ==============================

Model Accuracy:
 0.993054177416154

Confusion matrix:
 [[2399   18]
 [  17 2605]]

Classification report:
               precision    recall  f1-score   support

     anomaly       0.99      0.99      0.99      2417
      normal       0.99      0.99      0.99      2622

    accuracy                           0.99      5039
   macro avg       0.99      0.99      0.99      5039
weighted avg       0.99      0.99      0.99      5039
```

```
============================== LogisticRegression Model Test Results ==============================

Model Accuracy:
 0.9507838856916054

Confusion matrix:
 [[2270  147]
 [ 101 2521]]

Classification report:
               precision    recall  f1-score   support

      anomaly       0.96      0.94      0.95      2417
       normal       0.94      0.96      0.95      2622

    accuracy                           0.95      5039
   macro avg       0.95      0.95      0.95      5039
weighted avg       0.95      0.95      0.95      5039
```

```
============================== Random Forest Classifier Model Test Results ==============================

Model Accuracy:
 0.9978170271879341

Confusion matrix:
 [[2408    9]
 [   2 2620]]

Classification report:
               precision    recall  f1-score   support

      anomaly       1.00      1.00      1.00      2417
       normal       1.00      1.00      1.00      2622

    accuracy                           1.00      5039
   macro avg       1.00      1.00      1.00      5039
weighted avg       1.00      1.00      1.00      5039
```

EXTRACTING TP FP TN FN:

```
for i, v in models:
    print("For model:", i)
    TP, FP, TN, FN = perf_measure(Y_test, v.predict(X_test))
    print ("TP:", TP, "\tFP:", FP, "\t\tTN:", TN, "\tFN:", FN)
```

```
For model: Naive Baye Classifier
TP: 2080        FP: 187                 TN: 2435        FN: 337
For model: KNeighborsClassifier
TP: 2399        FP: 17          TN: 2605        FN: 18
For model: LogisticRegression
TP: 2270        FP: 101                 TN: 2521        FN: 147
For model: Random Forest Classifier
TP: 2408        FP: 2           TN: 2620        FN: 9
```

CONSOLIDATION OF THE RESULTS:

Machine Learning Classifiers	Correctly Classified Instances	Incorrectly Classified Instances	Accuracy rate
Naive Baye Classifier	4515	524	89.601%
KNeighbors Classifier	5004	35	99.305%
Logistic Regression	4791	248	95.078%
Random Forest Classifier	5028	11	99.781%

Test Accuracy:

Machine Learning Classifiers	TP Rate	Precision
Naive Baye Classifier	0.860	0.917
KNeighbors Classifier	0.992	0.992
Logistic Regression	0.939	0.957
Random Forest Classifier	0.996	0.999

RESULTS:

In this project, we have designed and implemented real time Intrusion detection system with the help of integration of Snort. It generates alerts for the rules which are created manually. The log files from snort are downloaded which is next uploaded onto Wireshark where we get a clear analysis of our data. For anomaly based intrusion detection we used ML classification models, such as Naive Bayes, KNeighbors, Logistic regression and Random forest. Here Random Forest classifier gives better detection rate and less false positives in detecting the intrusions among the other techniques used. The detection accuracy of 99% is achieved along with the smallest amount of incorrectly classified instances (11) in the Random Forest algorithm.

REFERENCES

[1] A. Patcha and J.-M. Park, “An overview of anomaly detection techniques: Existing solutions and latest technological trends,” Computer Networks, vol. 51, no. 12, pp. 3448–3470, 2007.

[2] K. Ilgun, R. A. Kemmerer, and P. A. Porras, “State transition analysis: A rule-based intrusion detection approach,” IEEE Transactions on Software Engineering, vol. 21, no. 3, pp. 181–199, 1995.

[3] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, “Specification-based anomaly detection: A new approach for detecting network intrusions,” in Proceedings of the ACM Conference on Computer and Communications Security, 2002, pp. 265–274.

[4] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges,” Computers & Security, Elsevier, vol. 28, no. 1, pp. 18–28, 2009.

[5] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in Symposium on Security and Privacy (SP), 2010 IEEE, 2010, pp. 305–316.

[6] D. E. Denning. “An Intrusion-Detection Model”. IEEE transactions on software engineering, Volume : 13 Issue: 2, February 1987.

[7] Harley Kozushko, “Intrusion Detection: Host-Based and Network-Based Intrusion Detection Systems”, on September 11, 2003.

[8] S. Antonatos K.G. Anagnostakis and E. P. Markats. Generating realistic workloads for network intrusion detection systems. In Proceedings ACM Workshop on Software and Performance., 2004.

[9] Mike Fisk and George Varghese. Fast content-based packet handling for intrusion detection. Technical report, University of California at San Diego, 2001.

[10] D. E. Denning, "An intrusion-detection model." IEEE Transactions on Software Engineering, Vol.SE-13(No.2):222-232, Feb. 1987.