UNIVERSITY SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

MAJOR PROJECT REPORT

DETECTION AND BLOCKING OF MALICIOUS TRAFFIC USING PYTHON AND WIRESHARK

Submitted by:
Naman Rajput
01616412816
B. Tech. ECE 8th sem

Naman Mittal
41216412816
B. Tech. ECE 8th sem

Submitted to:
Dr. M. Bala Krishna

DECLARATION OF ORIGINALITY

This is to certify that the dissertation entitled "DETECTION OF MALICIOUS TRAFFIC USING PYTHON AND WIRESHARK" done by Mr. Naman Rajput, Roll No. 01616412816 and Mr. Naman Mittal, Roll No. 41216412816 is an authentic work carried out by them at the University School of Information and Communication Technology under my guidance. The matter embodied in this Dissertation has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

ACKNOWLEDGEMENT

It gives us immense pleasure to take this opportunity to acknowledge our obligation to our Dissertation guide, Dr. M. Bala Krishna, Associate Professor, University School of Information and Communication Technology, GGSIPU, who has not only guided us throughout the Dissertation but also made a great effort in making the Dissertation a success. We are highly thankful to our guide for his keen interest, valuable guidance, technical acumen, round the clock encouragement, moral support & suggestions in the completion of the dissertation.

Naman Rajput
Roll No. 01616412816
B.Tech ECE 8th Sem
University School Of Information
Communication & Technology

Naman Mittal
Roll No. 41216412816
B.Tech ECE 8th Sem
University School Of Information
Communication & Technology

Certificate

The Major Project EC-454 titled as "Detection and Blocking of Malicious Traffic using Python and Wireshark" by Naman Rajput, Roll No. 01616412816 and Naman Mittal Roll. No. 41216412816 is a part of B.Tech (ECE) VIII semester.

Guide

(Dr. M. Bala Krishna)

CONTENTS

ABSTRACT

A network at any level needs a level of safety and security to work at an ideal pace. A user in an open network is prone to multiple unwanted traffic, due to which the user experiences a lag in the network. To ensure smooth sailing and uninterrupted safe surfing, our aim is to create a program which could detect potential attacking scenarios that we may encounter in our time surfing an open network. By detecting which users warrant a purpose for an action against them, we enable the user to block the malicious user or packets. This would enable the user to work in an unknown, possibly hostile environment, without worrying about invasion of privacy.

Our aim is to develop a software that could **detect & block malicious packets** using pyshark, which is a Python wrapper for tshark, allowing python packet parsing using wireshark. Wireshark, which is an open source packet analyzer, programmed to detect the packets in promiscuous mode and filter out the packets that would want to harm the network. The filtering of packets would be based on the protocols the packets belong to, number of packets the same ip send, port the packet wants to access, etc.

PROBLEM STATEMENT

When we start working in an open network we get prone to many antisocial elements trying to gain access to our data and exploit this information. To prevent this from happening, we want to build a software capable enough to block some of those attacks so that we can work securely in the work.

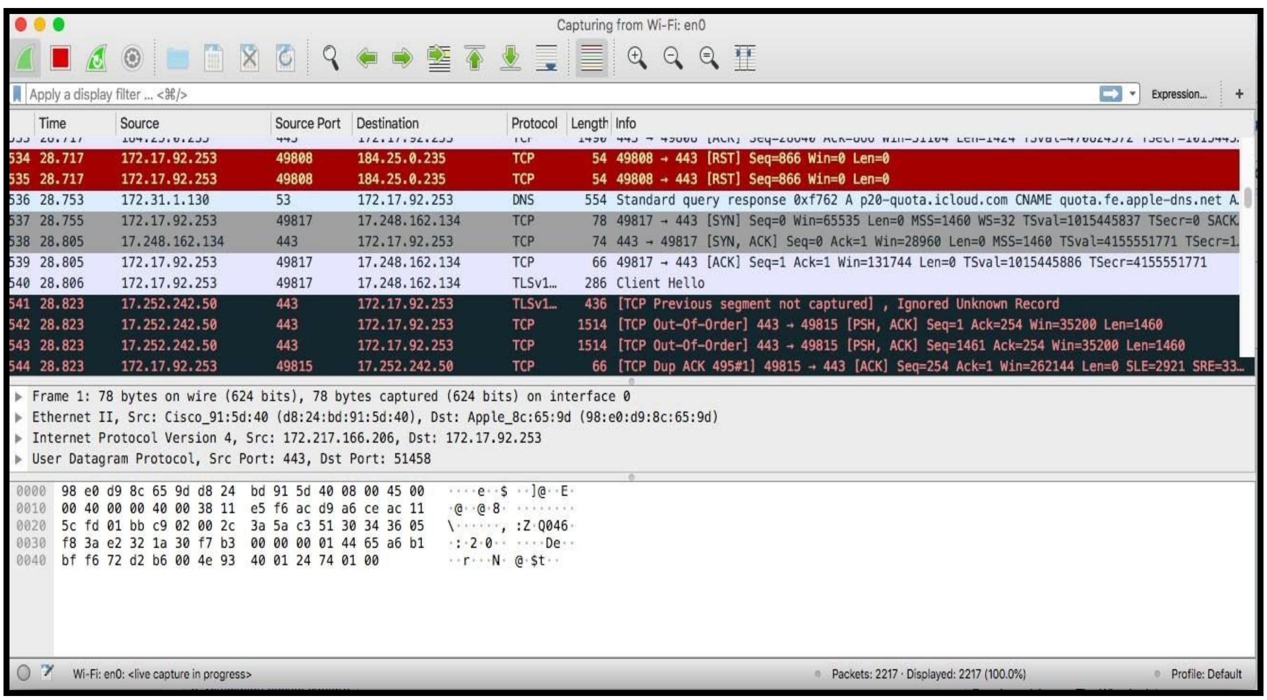
WORK DONE IN THE PREVIOUS PROJECT: We managed to build a program capable enough to detect malicious packets in a network belonging to some of the attacks, namely, Ping sweep attack, Null Scan attack, and ARP sweep attack. We successfully detected these attacks by making an open network and using one system to attack and another to detect.

DIFFERENCE IN PREVIOUS WORK DONE AND WORK TO BE DONE: We managed to detect three new attacks, SYN flooding, ARP poisoning, and IP protocol scanning and block the malicious IP address to mitigate the respective attacks.

Tools Used

WIRESHARK

The **wireshark** tool lets the user filter out the packets relevant using display filter and the capture filter, using this feature we'll first figure which packet might be suspicious enough to harm the network and then apply them in the filter through which we could easily identify which packets to warn the user about. **Wireshark,** an open-source packet analyzer helps in troubleshooting of such networks and analysing such packets[1].



Sample window of Wireshark Tool

NMAP: The Network Mapper

Nmap is a free and open software which is used for Network Management in an open network. Nmap uses multiple new ways to find out the ip address of a system in a network[8]. It can be used to monitor host and destination status in a network. Nmap is available as official binary packages on platforms like Windows, Linux, MacOs to use for network observation. Nmap has a classic command line structure but also has a GUI.

Following is the interface of NMAP network analyser:

```
gh0st@DedSec:~$ nmap --help
Nmap 7.60 ( https://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target specification}
TARGET SPECIFICATION:
  Can pass hostnames, IP addresses, networks, etc.
  Ex: scanme.nmap.org, microsoft.com/24, 192.168.0.1; 10.0.0-255.1-254
  -iL <inputfilename>: Input from list of hosts/networks
  -iR <num hosts>: Choose random targets
  --exclude <host1[,host2][,host3],...>: Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list from file
HOST DISCOVERY:
  -sL: List Scan - simply list targets to scan
  -sn: Ping Scan - disable port scan
  -Pn: Treat all hosts as online -- skip host discovery
  -PS/PA/PU/PY[portlist]: TCP SYN/ACK, UDP or SCTP discovery to given ports
  -PE/PP/PM: ICMP echo, timestamp, and netmask request discovery probes
  -PO[protocol list]: IP Protocol Ping
  -n/-R: Never do DNS resolution/Always resolve [default: sometimes]
  --dns-servers <serv1[,serv2],...>: Specify custom DNS servers
  --system-dns: Use OS's DNS resolver
  --traceroute: Trace hop path to each host
SCAN TECHNIQUES:
  -sS/sT/sA/sW/sM: TCP SYN/Connect()/ACK/Window/Maimon scans
  -sU: UDP Scan
  -sN/sF/sX: TCP Null, FIN, and Xmas scans
  --scanflags <flags>: Customize TCP scan flags
  -sI <zombie host[:probeport]>: Idle scan
  -sY/sZ: SCTP INIT/COOKIE-ECHO scans
  -sO: IP protocol scan
  -b <FTP relay host>: FTP bounce scan
PORT SPECIFICATION AND SCAN ORDER:
  -p <port ranges>: Only scan specified ports
    Ex: -p22; -p1-65535; -p U:53,111,137,T:21-25,80,139,8080,S:9
  --exclude-ports <port ranges>: Exclude the specified ports from scanning
  -F: Fast mode - Scan fewer ports than the default scan
  -r: Scan ports consecutively - don't randomize
  --top-ports <number>: Scan <number> most common ports
  --port-ratio <ratio>: Scan ports more common than <ratio>
```

Hping3 - Active Network Smashing Tool

Hping is a command-line oriented TCP/IP packet assembler/analyzer. The interface is inspired by the ping(8) unix command, but hping isn't only able to send ICMP echo requests. It supports TCP, UDP, ICMP and RAW-IP protocols, has a traceroute mode, the ability to send files between a covered channel, and many other features.

While hping was mainly used as a security tool in the past, it can be used in many ways by people that don't care about security to test networks and hosts. A subset of the stuff you can do using hping:

- Firewall testing
- Advanced port scanning
- Network testing, using different protocols, TOS, fragmentation
- Manual path MTU discovery
- Advanced traceroute, under all the supported protocols
- Remote OS fingerprinting
- Remote uptime guessing
- TCP/IP stacks auditing
- hping can also be useful to students that are learning TCP/IP.

```
root@kali:~# hping3 -h
usage: hping3 host [options]
  -h  --help      show this help
  -v  --version   show version
  -c  --count     packet count
  -i  --interval  wait (uX for X microseconds, for example -i u1000)
      --fast      alias for -i u10000 (10 packets for second)
      --faster    alias for -i u1000 (100 packets for second)
      --flood      sent packets as fast as possible. Don't show replies.
  -n  --numeric   numeric output
  -q  --quiet     quiet
  -I  --interface interface name (otherwise default routing interface)
  -V  --verbose   verbose mode
  -D  --debug     debugging info
  -z  --bind      bind ctrl+z to ttl           (default to dst port)
  -Z  --unbind    unbind ctrl+z
      --beep      beep for every matching packet received
```

Hping3 User Interface

Attacks Successfully Detected in previous work:

PING SWEEP ATTACK

Ping sweep scan is used in a network at many layers, to find out which IPs are currently active in the network. It can be performed using many layers: TCP or UDP or ICMP, but the most used one is ICMP Ping Sweep. In this, several ICMP type 8, ECHO requests are first sent after which, ICMP type 0, ECHO reply packets are used for extraction of IP[4].

ARP Sweep/ARP Scan Attack

While performing ARP scan, the attacker broadcasts ARP packets with destination MAC as 0xff:ff:ff:ff:ff:ff, for each and every possible IP destination in the selected subnet and an ARP response means that the host is active or live[4].

NULL Scan

A Null scan in a network is implemented in the TCP layer of the network. It is used to identify the ports of a network of a user, so that the user can be identified. A Null scan is the easiest scan to be performed to gain information of the user. It can be detected using a simple filter on the TCP layer of the network[7].

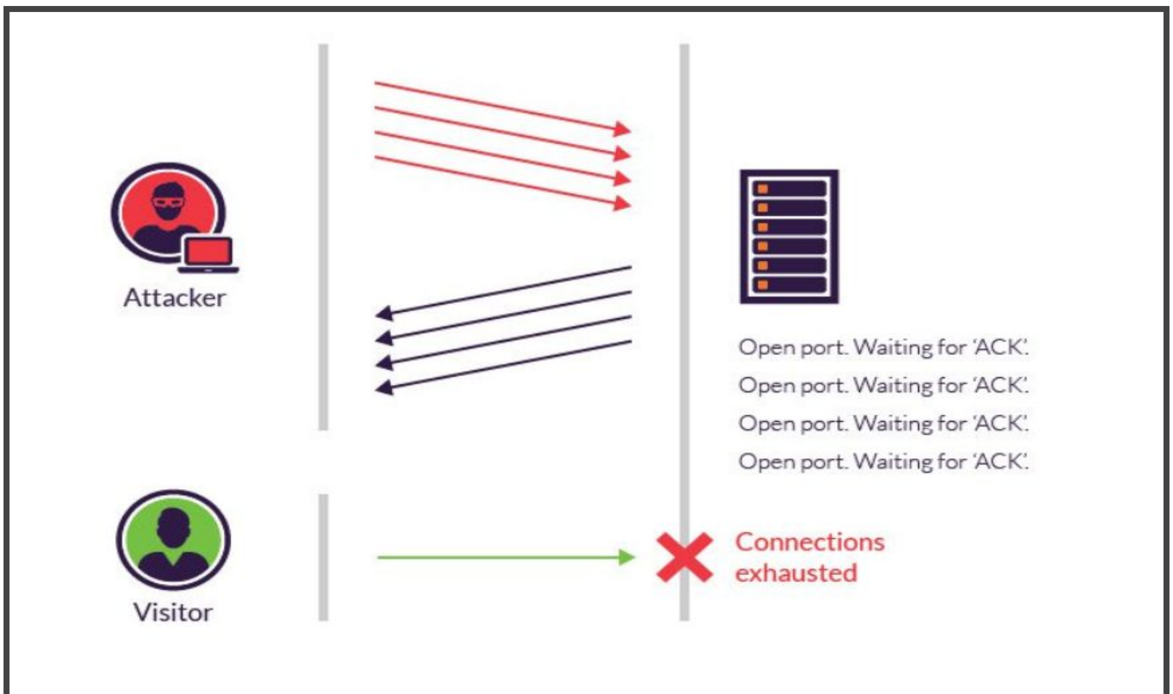
Current Work:

SYN Flooding Attack

A denial of service (DoS) attack where the user is attacked with a series of successive SYN requests. What the flooding of SYN requests does is it consumes a lot of servers' resources which in turn makes the system of the user unresponsive in the server to the authentic traffic in the network.

SYN flooding is experienced in the TCP layer of the network. In a legitimate three-way TCP handshake, the SYN request is acknowledged with SYN-ACK and then again the user responds with ACK. When a SYN flood attack is being executed, the attacker does not send the ACK packet code to the server network which makes the server send the SYN-ACK code to an unknown or wrong IP address. This creates half-open networks in the server of which the attacker takes advantage.

We'll block any IP address which seems suspicious of sending these kinds of SYN floods by first detecting the number of packets one IP address is sending over the network. If we get a large number of SYN packets, we'll detect the IP address of that system and block it from further use in the server[6].



Progression of a SYN flood.

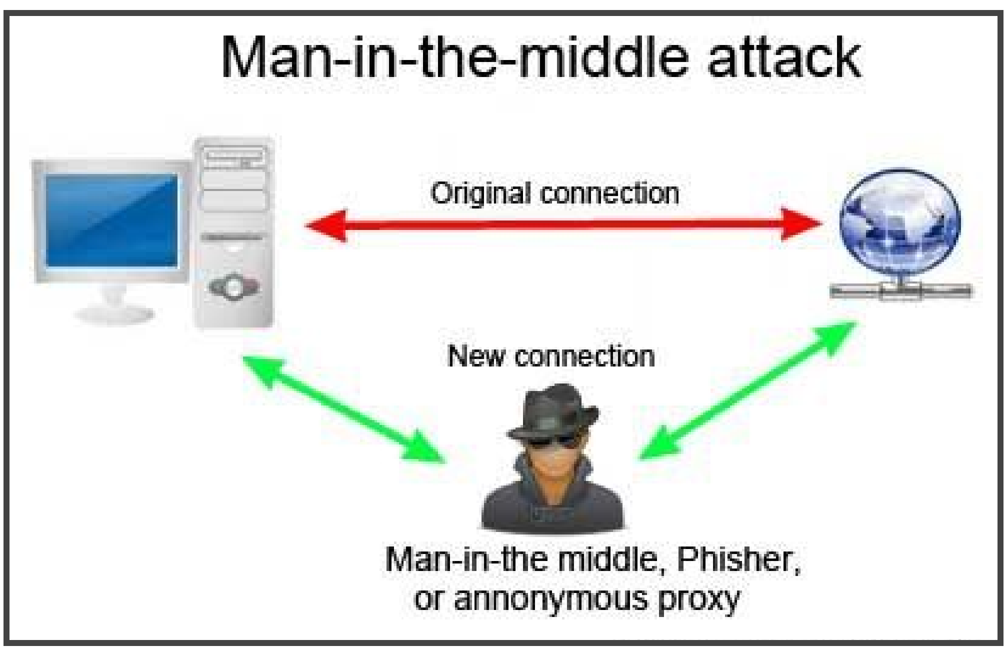
ARP Poisoning Attack

ARP poisoning attack is a type of man-in-the-middle attack done by exploiting the fact that a multiple numbers of ARP packets in a network can lead to the user responding to one of them. For an attacker to use this attack to intrude in the network, it would only need the access in the network and ARP poisoning can be carried out easily.

The attacker sends malicious packets of ARP type in the network's default gateway, it uses the fact that the ARP would change the IP into MAC address and that is what the ultimate goal of the attacker is.

The attacker sends an ARP false message to the gateway in the network asking for a system to connect its MAC address to the intruders IP address, or the other way round. By sending such a packet, the network broadcasts the changes that will need to be made in the destination users and this way the intruder can select to attack a specific system in the network.

For us to detect the ARP poisoning attack, we would identify if the user is sending ARP packets of packet type "1" or "2". By identifying such packets we declare them spammy and figure out the IP address using Wireshark to block such users[6].



IP Protocol Scan

IP protocol scanning is used to find out the supported protocols in the users' system. It surfs through the IP protocol numbers to find the protocols which are supported by the system unlike any other port scan that go through the port numbers.

UDP scan is the most similar to the IP protocol scan. It sends empty header IP packets in the network to the user and traverses through the IP protocol of 8-bit size. This scan will help the intruder in figuring out what sort of packets are being filtered in that user's network system.

Like the TCP scan there can be three scenarios possible, the host replies with a response and the connection is established, the intruder now knows that this port is open. Second, the user blocks the incoming packet and sends a response saying that connection has been denied. Third, there is no reply from the user.

To detect whether there is an intruder trying to perform an IP protocol scan or not, we can scan for an uncanny number of an empty header, ICMP protocol packets in a network with an empty header. By detecting these packets, we can say that there is an IP protocol scanning attack that is being undertaken. We can identify the IP address of the system and then block that IP from communicating further in the network[6].

```
# nmap -sO 62.233.173.90 para

Starting Nmap ( http://nmap.org )
Nmap scan report for ntwklan-62-233-173-90.devs.futuro.pl (62.233.173.90)
Not shown: 240 closed ports
PROTOCOL STATE         SERVICE
1        open          icmp
4        open|filtered ip
6        open          tcp
8        open|filtered egp
9        open|filtered igp
17       filtered      udp
47       open|filtered gre
53       filtered      swipe
54       open|filtered narp
55       filtered      mobile
77       filtered      sun-nd
80       open|filtered iso-ip
88       open|filtered eigrp
89       open|filtered ospfigp
94       open|filtered ipip
103      filtered      pim

Nmap scan report for para (192.168.10.191)
Not shown: 252 closed ports
PROTOCOL STATE         SERVICE
1        open          icmp
2        open|filtered igmp
6        open          tcp
17       filtered      udp
MAC Address: 00:60:1D:38:32:90 (Lucent Technologies)

Nmap done: 2 IP addresses (2 hosts up) scanned in 458.04 seconds
```

Methodology Adopted/ALGORITHM

To overcome our problem of invasion of privacy in a network, we will go through the following steps:

1. Observe the networks by exploring the methods of exploitations and bugs for accessing and using the data of a user.
2. By identifying how these exploitations work and how the data is accessed and identity compromised using this working.
3. Work on a possible way to try and identify these attacks which try and access our personal data.
4. Implement a code to detect a few of these attacks based on the exploitations, which would warn the user of the culprit.
5. Try and identify the source of the attack in the network, so that further action could be taken against the user by blocking the user from the network.
6. Device a method to block that specific IP address, which is attacking a system in the network, from using that network.

SOFTWARE & HARDWARE REQUIREMENTS

- Pyshark 0.4.2.9 (https://pypi.org/project/pyshark)
- Python 2.7.16 (https://www.python.org/downloads/release/python-2716)
- Wireshark 2.6.10 (https://www.wireshark.org/)
- NMAP: Network analyser tool (https://nmap.org)
- Hping3 (https://tools.kali.org/information-gathering/hping3)
- Iptables (http://ipset.netfilter.org/iptables.man.html)
- Packet data of the network on which attacks were performed
- A system with minimum i3 processor, 8 gb RAM & 1 gb of free space

Blocking the Intruder

To work in an open network we open ourselves to endless exploitations from all over the network. Cyber security is taking a leap in recent times and it has become one of the most important aspects of every user in a network. When we think of an open network we are hesitant enough not to log in to one or when we do log in to an open network, we do so restricting ourselves to a specific job and disconnect immediately.

By detecting a number of attacks in an open network that can be performed by an intruder, we open up the possibility of eliminating such attacks in a network. We managed to detect whether an attack is being carried out or not, and if the attack is being carried out, we can detect the IP address the attack is being performed from.

For blocking an intruder or an attacker who wants to exploit the network, we would need to have the access as a network administrator to make changes in the network. Being the administrator, we can block the IP address by running a script to block that IP address which is trying to attack the user.

In conclusion, we will make the open network a safer network by first identifying any attack that might be carried out on the network and then blocking the IP address that is trying to undertake that attack to comprise the safety and privacy of the network. This way a user can without any hesitation use an open network.

SIMULATION

1. SYN Flooding Attack

```
import pyshark
import operator
import os

def syn_flooding():
    capture = pyshark.FileCapture('./hping_syn.pcap', display_filter = 'tcp')
    capture.set_debug()
    under_attack = 0
    while True:
    ip_addresses = {}
    syn_count = 0
    for packet in capture:
    if (str(packet.tcp.flags) == "0x00000002"):
    src_ip = str(packet.ip.src)
    syn_count += 1
    if src_ip not in ip_addresses:
        ip_addresses[src_ip] = 1
    elif src_ip in ip_addresses:
        ip_addresses[src_ip] += 1
    if (syn_count > 20):
        ip = max(ip_addresses.items(), key=operator.itemgetter(1))[0]
        print("SYN FLOODING ATTACK DETECTED, SOURCE IP: %s" %ip)
        return ip
if __name__ ==  "__main__":
```

block = syn_flooding()

command = "iptables -A INPUT -s " + block + " -j DROP"

if (os.system(command) == 0):

print("%s BLOCKED SUCCESSFULLY" %block)

Output:

```
========= RESTART: /home/gh0st/Desktop/major_proj/syn_flood_block.py =========
[2020-05-16 16:23:03.535715] DEBUG: FileCapture: Creating TShark subprocess with
 parameters: /usr/bin/tshark -l -n -T pdml -Y tcp -r ./hping_syn.pcap
[2020-05-16 16:23:03.542559] DEBUG: FileCapture: %s subprocess created
SYN FLOODING ATTACK DETECTED, SOURCE IP: 192.168.75.137
Exception OSError: OSError(3, 'No such process') in <generator object _packets_f
rom_tshark_sync at 0x7fab90f65e60> ignored
192.168.75.137 BLOCKED SUCCESSFULLY
```

Blocking the IP address associated with undertaking the SYN Flooding attack in the network. First the attacker was detected and then blocked from accessing the network.

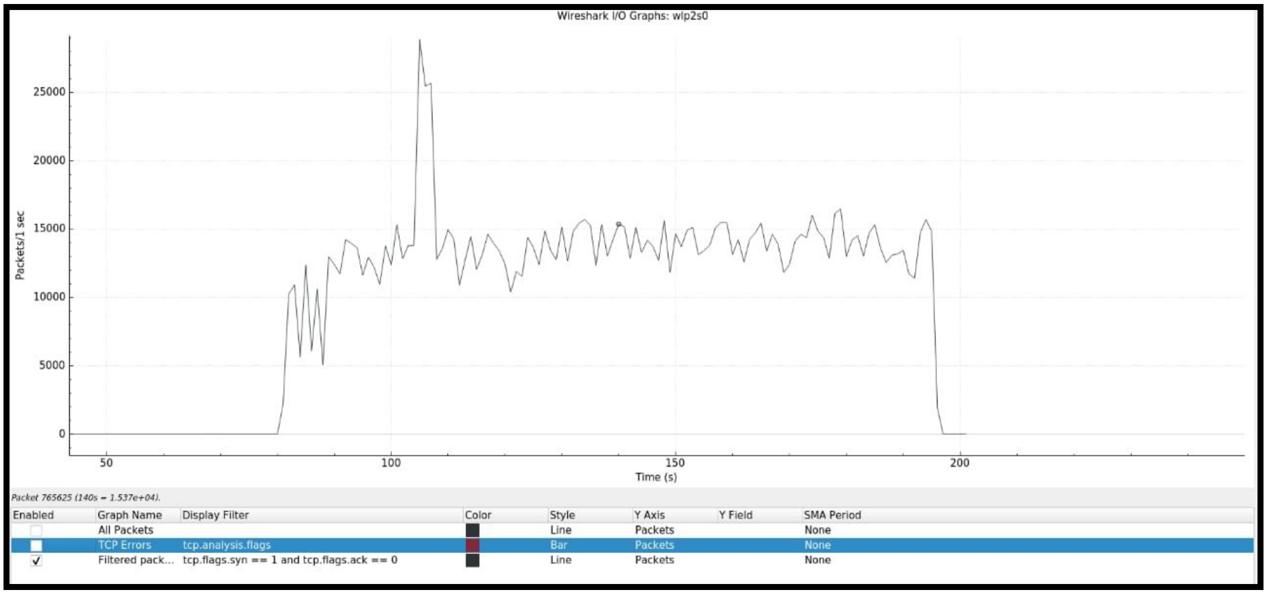
Network Data File Preview:

tcp.flags==0x002

No.	Time	Source	Destination	Protocol	Length	Info
8	7.035592	192.168.75.137	192.168.75.1	TCP	58	1152 → 0 [SYN] Seq=0 Win=512 Len=4
9	8.039004	192.168.75.137	192.168.75.1	TCP	58	1153 → 0 [SYN] Seq=0 Win=512 Len=4
10	9.042102	192.168.75.137	192.168.75.1	TCP	58	1154 → 0 [SYN] Seq=0 Win=512 Len=4
13	10.045123	192.168.75.137	192.168.75.1	TCP	58	1155 → 0 [SYN] Seq=0 Win=512 Len=4
14	11.048171	192.168.75.137	192.168.75.1	TCP	58	1156 → 0 [SYN] Seq=0 Win=512 Len=4
15	12.051256	192.168.75.137	192.168.75.1	TCP	58	1157 → 0 [SYN] Seq=0 Win=512 Len=4
16	13.054301	192.168.75.137	192.168.75.1	TCP	58	1158 → 0 [SYN] Seq=0 Win=512 Len=4
17	14.057601	192.168.75.137	192.168.75.1	TCP	58	1159 → 0 [SYN] Seq=0 Win=512 Len=4
18	15.060441	192.168.75.137	192.168.75.1	TCP	58	1160 → 0 [SYN] Seq=0 Win=512 Len=4
19	16.063476	192.168.75.137	192.168.75.1	TCP	58	1161 → 0 [SYN] Seq=0 Win=512 Len=4
20	17.066566	192.168.75.137	192.168.75.1	TCP	58	1162 → 0 [SYN] Seq=0 Win=512 Len=4
21	18.069651	192.168.75.137	192.168.75.1	TCP	58	1163 → 0 [SYN] Seq=0 Win=512 Len=4
22	19.075265	192.168.75.137	192.168.75.1	TCP	58	1164 → 0 [SYN] Seq=0 Win=512 Len=4
23	20.078671	192.168.75.137	192.168.75.1	TCP	58	1165 → 0 [SYN] Seq=0 Win=512 Len=4
24	21.083687	192.168.75.137	192.168.75.1	TCP	58	1166 → 0 [SYN] Seq=0 Win=512 Len=4
25	22.086834	192.168.75.137	192.168.75.1	TCP	58	1167 → 0 [SYN] Seq=0 Win=512 Len=4
26	23.089848	192.168.75.137	192.168.75.1	TCP	58	1168 → 0 [SYN] Seq=0 Win=512 Len=4
27	24.093009	192.168.75.137	192.168.75.1	TCP	58	1169 → 0 [SYN] Seq=0 Win=512 Len=4

```
  Acknowledgment number: 244884630
  Acknowledgment number (raw): 244884630
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x002 (SYN)
  Window size value: 512
  [Calculated window size: 512]
  Checksum: 0x5fd6 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  [SEQ/ACK analysis]
  [Timestamps]
  TCP payload (4 bytes)
Data (4 bytes)
```

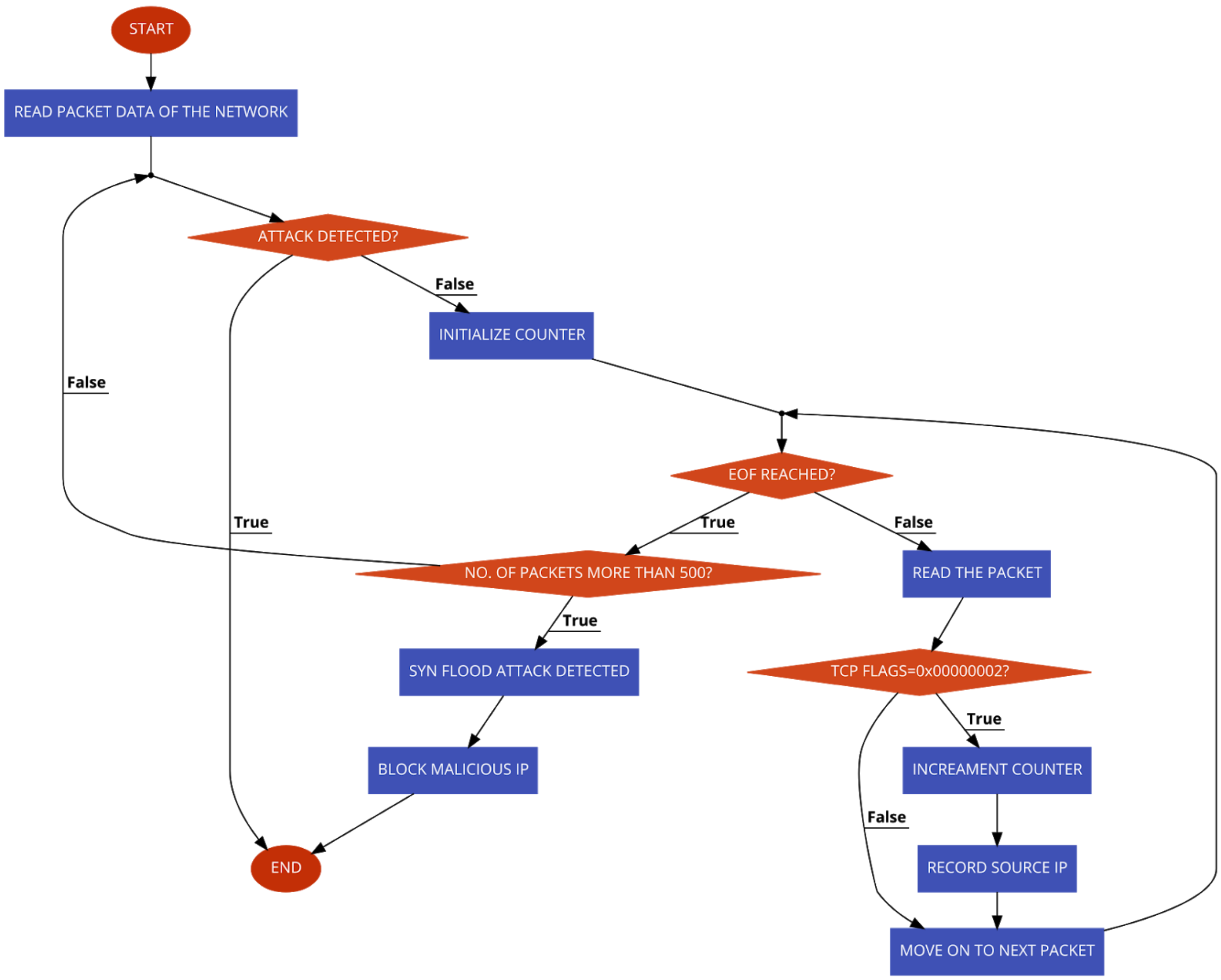
I/O Graph:



TCP Flag Details:

```
▾ Flags: 0x002 (SYN)
     000. .... .... = Reserved: Not set
     ...0 .... .... = Nonce: Not set
     .... 0... .... = Congestion Window Reduced (CWR): Not set
     .... .0.. .... = ECN-Echo: Not set
     .... ..0. .... = Urgent: Not set
     .... ...0 .... = Acknowledgment: Not set
     .... .... 0... = Push: Not set
     .... .... .0.. = Reset: Not set
  ▾  .... .... ..1. = Syn: Set
     ▾ [Expert Info (Chat/Sequence): Connection establish request (SYN): server port 80]
          [Connection establish request (SYN): server port 80]
          [Severity level: Chat]
          [Group: Sequence]
     .... .... ...0 = Fin: Not set
     [TCP Flags: ··········S·]
  Window size value: 64
  [Calculated window size: 64]
  Checksum: 0x7df2 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
```

Flowchart:


START
READ PACKET DATA OF THE NETWORK
ATTACK DETECTED?
False
INITIALIZE COUNTER
False
True
EOF REACHED?
True
False
NO. OF PACKETS MORE THAN 500?
READ THE PACKET
True
SYN FLOOD ATTACK DETECTED
TCP FLAGS=0x00000002?
True
BLOCK MALICIOUS IP
INCREAMENT COUNTER
False
END
RECORD SOURCE IP
MOVE ON TO NEXT PACKET

2. ARP Poisoning Attack

```
import pyshark
import operator
import os

def arp_poisoning():
    capture = pyshark.FileCapture('./arp_poison.pcap', display_filter='arp')
    capture.set_debug()
    mac_ip={}
    while True:
        for packet in capture:
            packetType = str(packet.arp.opcode)
            if packetType == '2' or packetType == '1':
                mac = str(packet.arp.src_hw_mac)
                ip = str(packet.arp.src_proto_ipv4)
                if mac in mac_ip:
                    if mac_ip[mac] != ip:
                        print("\nARP POISONING ATTACK DETECTED FROM IP: " + ip)
                        return ip
                else:
                    mac_ip[mac] = ip

if __name__ ==  "__main__":
    block = arp_poisoning()
    command = "iptables -A INPUT -s " + block + " -j DROP"
```

```
if (os.system(command) == 0):
    print("%s BLOCKED SUCCESSFULLY" %block)
```

Output:

```
========== RESTART: /home/gh0st/Desktop/major_proj/arp_poisoning.py ==========
[2020-05-16 16:20:57.889023] DEBUG: FileCapture: Creating TShark subprocess with
 parameters: /usr/bin/tshark -l -n -T pdml -Y arp -r ./arp_poison.pcap
[2020-05-16 16:20:57.895032] DEBUG: FileCapture: %s subprocess created

ARP POISONING ATTACK DETECTED FROM IP: 172.16.0.105
Exception OSError: OSError(3, 'No such process') in <generator object _packets_f
rom_tshark_sync at 0x7f104816be60> ignored
172.16.0.105 BLOCKED SUCCESSFULLY
>>> |
```

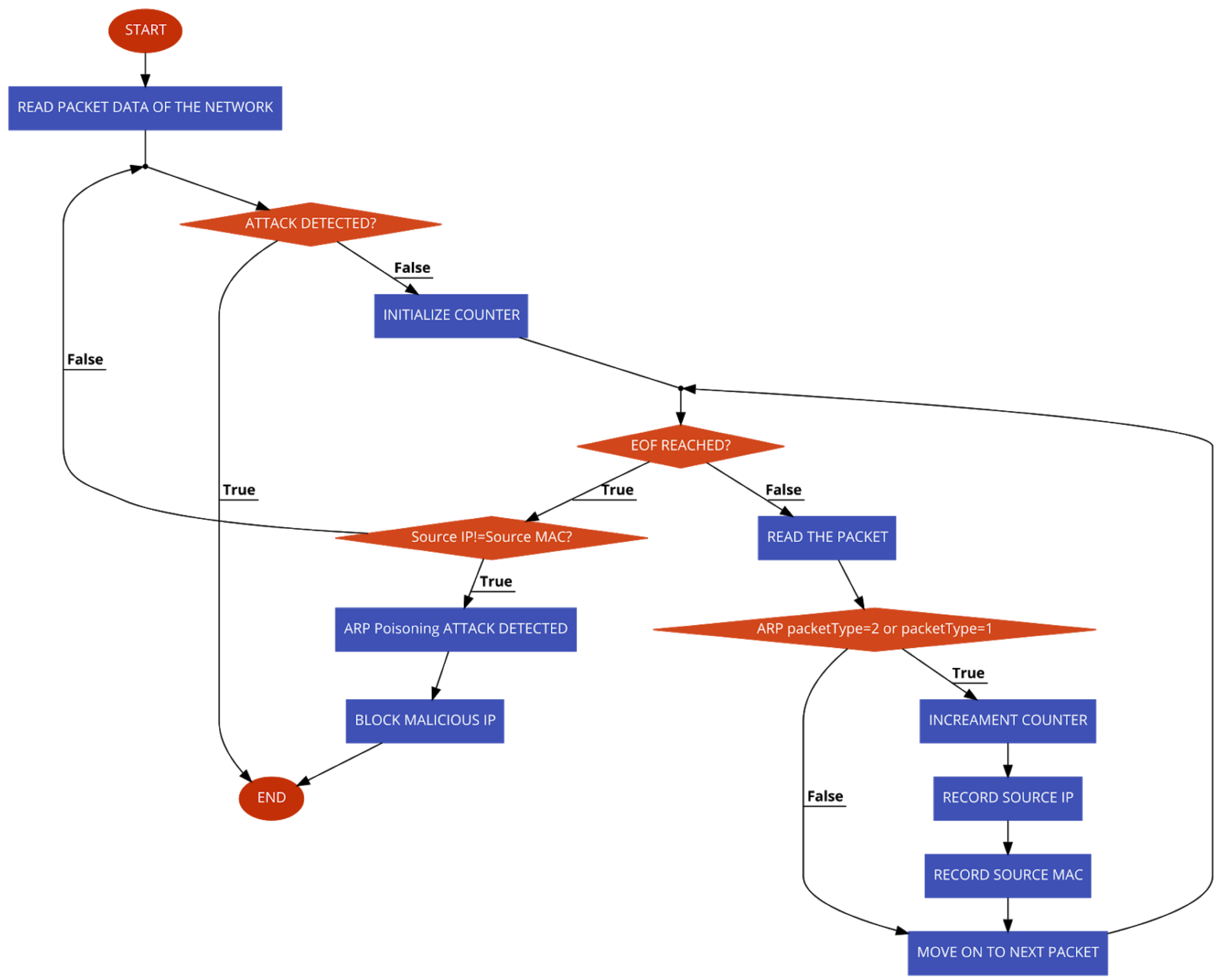
Blocking the IP address associated with undertaking the ARP Poisoning attack in the network. First the attacker was detected and then blocked from accessing the network.

Network Data File Preview:

```
arp
No.  Time       Source             Destination        Protocol Length Info
 54  4.646389   HewlettP_bf:91:ee  Dell_c0:56:f0      ARP      60 Who has 172.16.0.107? Tell 172.16.0.1
 55  4.646442   Dell_c0:56:f0      HewlettP_bf:91:ee  ARP      42 172.16.0.107 is at 00:21:70:c0:56:f0
 56  4.646455   HewlettP_bf:91:ee  Dell_c0:56:f0      ARP      60 172.16.0.1 is at 00:25:b3:bf:91:ee
165  14.392559  HewlettP_bf:91:ee  Broadcast          ARP      60 Who has 172.16.0.1? Tell 172.16.0.105

▸ Frame 55: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
▸ Ethernet II, Src: Dell_c0:56:f0 (00:21:70:c0:56:f0), Dst: HewlettP_bf:91:ee (00:25:b3:bf:91:ee)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: Dell_c0:56:f0 (00:21:70:c0:56:f0)
    Sender IP address: 172.16.0.107
    Target MAC address: HewlettP_bf:91:ee (00:25:b3:bf:91:ee)
    Target IP address: 172.16.0.1
```

Flowchart:


START
READ PACKET DATA OF THE NETWORK
ATTACK DETECTED?
False
INITIALIZE COUNTER
False
True
EOF REACHED?
True
False
Source IP!=Source MAC?
READ THE PACKET
True
ARP Poisoning ATTACK DETECTED
ARP packetType=2 or packetType=1
True
BLOCK MALICIOUS IP
INCREAMENT COUNTER
END
False
RECORD SOURCE IP
RECORD SOURCE MAC
MOVE ON TO NEXT PACKET

3. IP Protocol Scan Attack

```
import pyshark
import operator
import os

def ip_scan():
    capture = pyshark.FileCapture('./packet_ip_ub.pcap', display_filter = "icmp")
    capture.set_debug()
    while True:
        ip_add = {}
        for packet in capture:
            if(str(packet.icmp.type) == '3'):
                if str(packet.ip.src) not in str(ip_add):
                    ip_add[packet.ip.src] = 1
                else:
                    ip_add[packet.ip.src] += 1

        for ip, count in ip_add.items():
            if count > 5:
                print("IP PROTOCOL SCAN DETECTED, IP: ", ip)
                return ip

if __name__ ==  "__main__":
    block = ip_scan()
    command = "iptables -A INPUT -s " + block + " -j DROP"
```

```
if (os.system(command) == 0):
    print("%s BLOCKED SUCCESSFULLY" %block)
```

<u>Output:</u>

```
========== RESTART: /home/gh0st/Desktop/major_proj/ip_scan_block.py ==========
[2020-05-16 16:19:01.365925] DEBUG: FileCapture: Creating TShark subprocess with
 parameters: /usr/bin/tshark -l -n -T pdml -Y icmp -r ./packet_ip_ub.pcap
[2020-05-16 16:19:01.372145] DEBUG: FileCapture: %s subprocess created
[2020-05-16 16:19:01.624173] DEBUG: FileCapture: EOF reached (sync)
('IP PROTOCOL SCAN DETECTED, IP: ', '192.168.75.2')
192.168.75.2 BLOCKED SUCCESSFULLY
```

Blocking the IP address associated with undertaking the IP Protocol Scanning attack in the network. First the attacker was detected and then blocked from accessing the network.

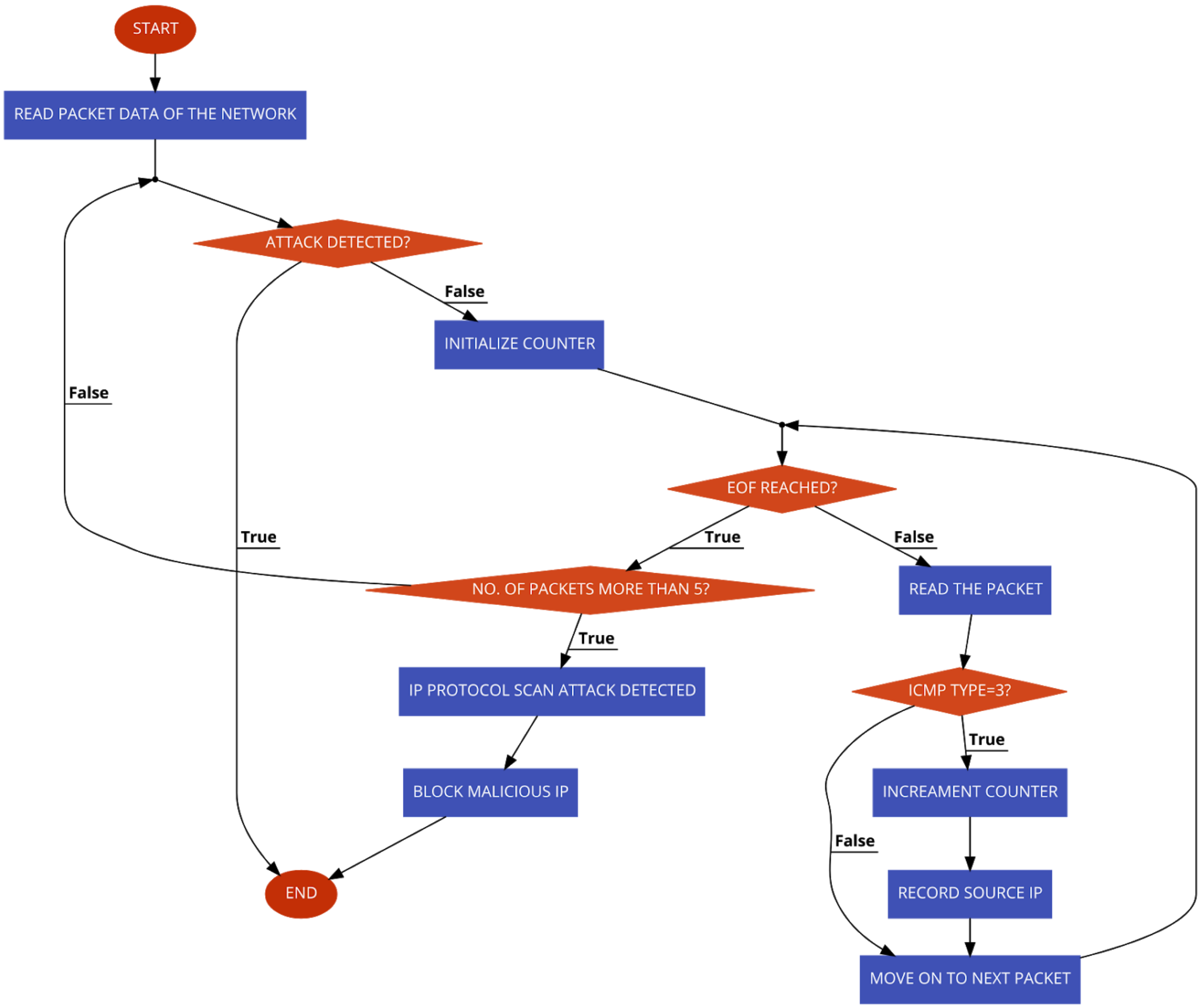
<u>Network Data File Preview:</u>

icmp.type ==3 and icmp.code==1

No.	Time	Source	Destination	Protocol	Length	Info
7	13.077923	192.168.75.2	192.168.75.132	ICMP	120	Destination unreachable (Host unreachable)
11	14.560662	192.168.75.2	192.168.75.132	ICMP	120	Destination unreachable (Host unreachable)
14	16.135750	192.168.75.2	192.168.75.132	ICMP	120	Destination unreachable (Host unreachable)
618	476.963186	192.168.75.2	192.168.75.132	ICMP	120	Destination unreachable (Host unreachable)
620	478.404276	192.168.75.2	192.168.75.132	ICMP	120	Destination unreachable (Host unreachable)
622	479.942357	192.168.75.2	192.168.75.132	ICMP	120	Destination unreachable (Host unreachable)

```
Frame 14: 120 bytes on wire (960 bits), 120 bytes captured (960 bits)
Ethernet II, Src: VMware_f5:2e:f3 (00:50:56:f5:2e:f3), Dst: VMware_0f:71:a3 (00:0c:29:0f:71:a3)
Internet Protocol Version 4, Src: 192.168.75.2, Dst: 192.168.75.132
Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 1 (Host unreachable)
    Checksum: 0x1522 [correct]
    [Checksum Status: Good]
    Unused: 00000000
  Internet Protocol Version 4, Src: 192.168.75.132, Dst: 192.168.75.2
  User Datagram Protocol, Src Port: 137, Dst Port: 137
  NetBIOS Name Service
```

Flowchart:


START
READ PACKET DATA OF THE NETWORK
ATTACK DETECTED?
False
INITIALIZE COUNTER
False
True
EOF REACHED?
True
False
NO. OF PACKETS MORE THAN 5?
READ THE PACKET
True
IP PROTOCOL SCAN ATTACK DETECTED
ICMP TYPE=3?
True
BLOCK MALICIOUS IP
INCREAMENT COUNTER
False
END
RECORD SOURCE IP
MOVE ON TO NEXT PACKET

4. Ping Sweep Attack

```
import pyshark
import operator
import os

def ping_sweep():
    capture = pyshark.FileCapture('./ping.pcap', display_filter="icmp")
    capture.set_debug()
    under_attack=0
    while True:
        ip_address={}
        count=0
        for packet in capture:
            typ=str(packet.icmp.type)
            if typ=='0':
                count+=1
                ip=str(packet.ip.src)
                if ip in ip_address:
                    ip_address[ip]+=1
                else:
                    ip_address[ip]=1
            elif typ=='8':
                count+=1
                ip=str(packet.ip.dst)
                if ip in ip_address:
                    ip_address[ip]+=1
```

```
            else:
                ip_address[ip]=1
        if count > 7:
            if under_attack == 0:
                IP = max(ip_address.items(), key=operator.itemgetter(1))[0]
                print('------------------------------------------------------------')
                print("EXCESSIVE ICMP TRAFFIC DETECTED, SOURCE IP: %s" %IP)
                return IP
            under_attack=1
        elif count < 7:
            under_attack=0

if __name__ ==  "__main__":
    block = ping_sweep()
    command = "iptables -A INPUT -s " + block + " -j DROP"

    if (os.system(command) == 0):
        print("%s BLOCKED SUCCESSFULLY" %block)
```

Output:

```
=========== RESTART: /home/gh0st/Desktop/major_proj/ping_sweep.py ===========
[2020-05-19 10:40:31.313446] DEBUG: FileCapture: Creating TShark subprocess with
 parameters: /usr/bin/tshark -l -n -T pdml -Y icmp -r ./ping.pcap
[2020-05-19 10:40:31.318791] DEBUG: FileCapture: %s subprocess created
[2020-05-19 10:40:31.477368] DEBUG: FileCapture: EOF reached (sync)
---------------------------------------------------------------
EXCESSIVE ICMP TRAFFIC DETECTED, SOURCE IP: 192.168.75.132
192.168.75.132 BLOCKED SUCCESSFULLY
=============================== RESTART: Shell ===============================
```

Blocking the IP address associated with undertaking the Ping Sweep attack in the network. First the attacker was detected and then blocked from accessing the network.

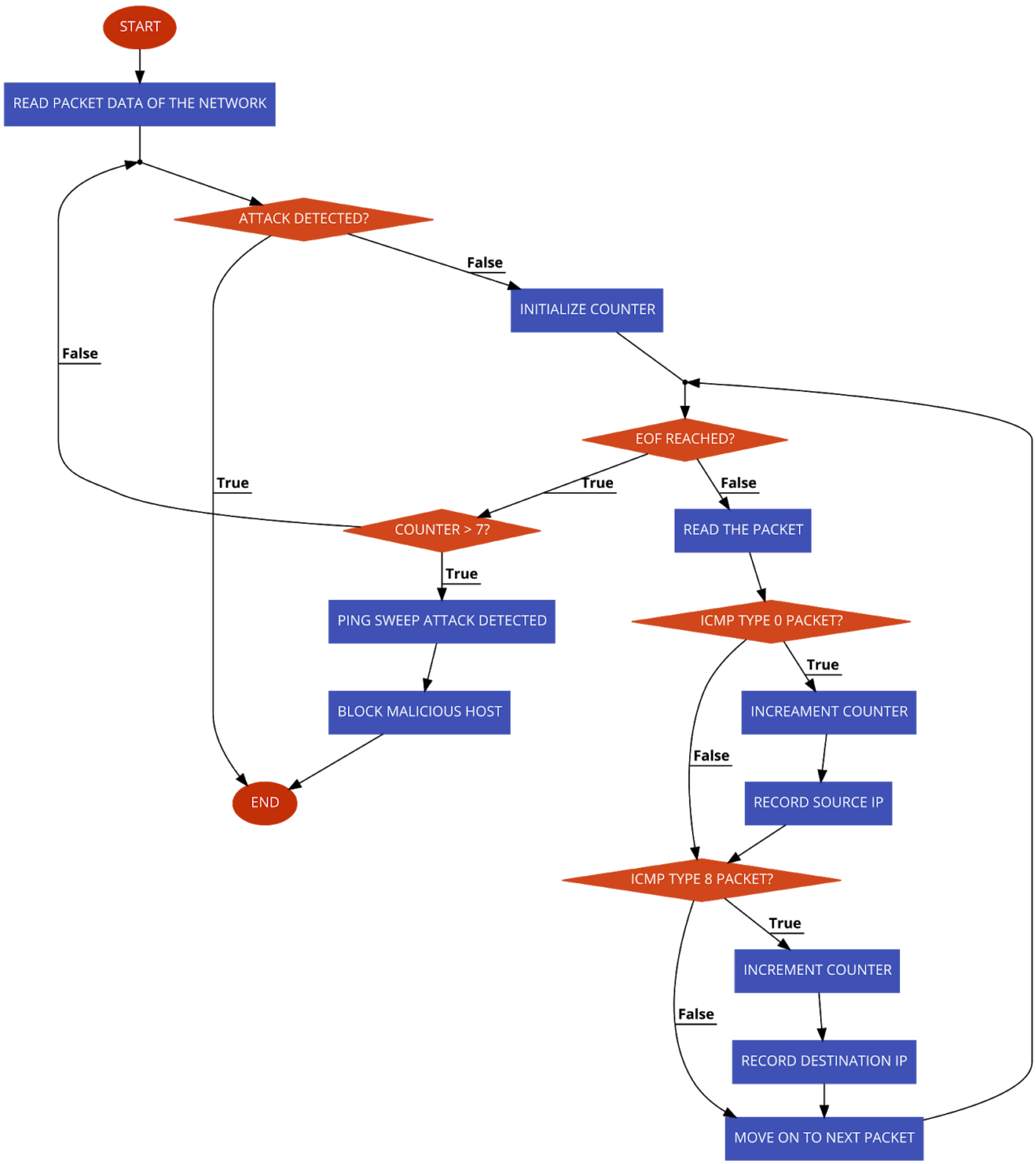
Network Data File Preview:

```
icmp.type==0 or icmp.type == 8
No.  Time       Source          Destination     Protocol Length Info
10 13.706916  192.168.75.1    192.168.75.132  ICMP  74 Echo (ping) request  id=0x0001, seq=17/4352, ttl=128 (reply in 11)
11 13.707279  192.168.75.132  192.168.75.1    ICMP  74 Echo (ping) reply    id=0x0001, seq=17/4352, ttl=128 (request in 10)
12 14.716736  192.168.75.1    192.168.75.132  ICMP  74 Echo (ping) request  id=0x0001, seq=18/4608, ttl=128 (reply in 13)
13 14.717049  192.168.75.132  192.168.75.1    ICMP  74 Echo (ping) reply    id=0x0001, seq=18/4608, ttl=128 (request in 12)
14 15.724823  192.168.75.1    192.168.75.132  ICMP  74 Echo (ping) request  id=0x0001, seq=19/4864, ttl=128 (reply in 15)
15 15.725175  192.168.75.132  192.168.75.1    ICMP  74 Echo (ping) reply    id=0x0001, seq=19/4864, ttl=128 (request in 14)
16 16.738845  192.168.75.1    192.168.75.132  ICMP  74 Echo (ping) request  id=0x0001, seq=20/5120, ttl=128 (reply in 17)
17 16.739219  192.168.75.132  192.168.75.1    ICMP  74 Echo (ping) reply    id=0x0001, seq=20/5120, ttl=128 (request in 16)

Frame 12: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
Ethernet II, Src: VMware_c0:00:08 (00:50:56:c0:00:08), Dst: VMware_0f:71:a3 (00:0c:29:0f:71:a3)
Internet Protocol Version 4, Src: 192.168.75.1, Dst: 192.168.75.132
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4d49 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 18 (0x0012)
    Sequence number (LE): 4608 (0x1200)
    [Response frame: 13]
  Data (32 bytes)
```

Wireshark detecting the incoming packets in a network. By filtering out the ICMP packets, we can check whether there is a possibility of a Ping Sweep attack or not.

Flowchart:


START
READ PACKET DATA OF THE NETWORK
ATTACK DETECTED?
False
INITIALIZE COUNTER
False
True
EOF REACHED?
True
False
COUNTER > 7?
READ THE PACKET
True
PING SWEEP ATTACK DETECTED
ICMP TYPE 0 PACKET?
True
BLOCK MALICIOUS HOST
INCREAMENT COUNTER
False
END
RECORD SOURCE IP
ICMP TYPE 8 PACKET?
True
INCREMENT COUNTER
False
RECORD DESTINATION IP
MOVE ON TO NEXT PACKET

5. ARP Sweep Attack

```
import pyshark
import operator
import os


def arp_sweep():
    capture = pyshark.FileCapture('./arp_scan.pcap', display_filter = "arp")
    capture.set_debug()
    while True:
        mac_addresses={}
        arp_count=0
        under_attack = 0
        for packet in capture:
            dest_mac = packet.eth.dst
            packet_type =  str(packet.arp.opcode)
            if packet_type == '1':
                arp_count+=1
                src_mac = packet.eth.src
                if src_mac not in mac_addresses:
                    mac_addresses[src_mac]=1;
                elif src_mac in mac_addresses:
                    mac_addresses[src_mac]+=1
        if arp_count > 100:
            if under_attack == 0:
                mac = max(mac_addresses.items(), key=operator.itemgetter(1))[0]
                print('-----------------------------------------------------')
```

```
            print("EXCESSIVE ARP TRAFFIC DETECTED, SOURCE MAC: %s" %mac )
            return mac
        elif arp_count<100:
          under_attack=0


if __name__ ==  "__main__":
    block = arp_sweep()
    command = "iptables -A INPUT -j DROP -m mac --mac-source " + block


    if (os.system(command) == 0):
        print("%s BLOCKED SUCCESSFULLY" %block)
```

Network Data File Preview:

No.	Time	Source	Destination	Protocol	Length	Info
12	0.001623	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.13? Tell 192.168.47.171
13	0.001684	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.14? Tell 192.168.47.171
14	0.198543	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.4? Tell 192.168.47.171
15	0.198634	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.5? Tell 192.168.47.171
16	0.198696	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.6? Tell 192.168.47.171
17	0.198759	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.7? Tell 192.168.47.171
18	0.198819	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.8? Tell 192.168.47.171
19	0.198926	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.9? Tell 192.168.47.171
20	0.199019	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.10? Tell 192.168.47.171
21	0.199083	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.11? Tell 192.168.47.171
22	0.199142	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.12? Tell 192.168.47.171
23	0.199202	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.13? Tell 192.168.47.171
24	0.199263	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.14? Tell 192.168.47.171
25	0.401780	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.19? Tell 192.168.47.171
26	0.402007	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.20? Tell 192.168.47.171
27	0.402161	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.21? Tell 192.168.47.171
28	0.402309	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.22? Tell 192.168.47.171
29	0.402487	VMware_1d:b3:b1	Broadcast	ARP	42	Who has 192.168.47.23? Tell 192.168.47.171

```
Frame 10: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
Ethernet II, Src: VMware_1d:b3:b1 (00:0c:29:1d:b3:b1), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: VMware_1d:b3:b1 (00:0c:29:1d:b3:b1)
    Sender IP address: 192.168.47.171
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 192.168.47.11
```

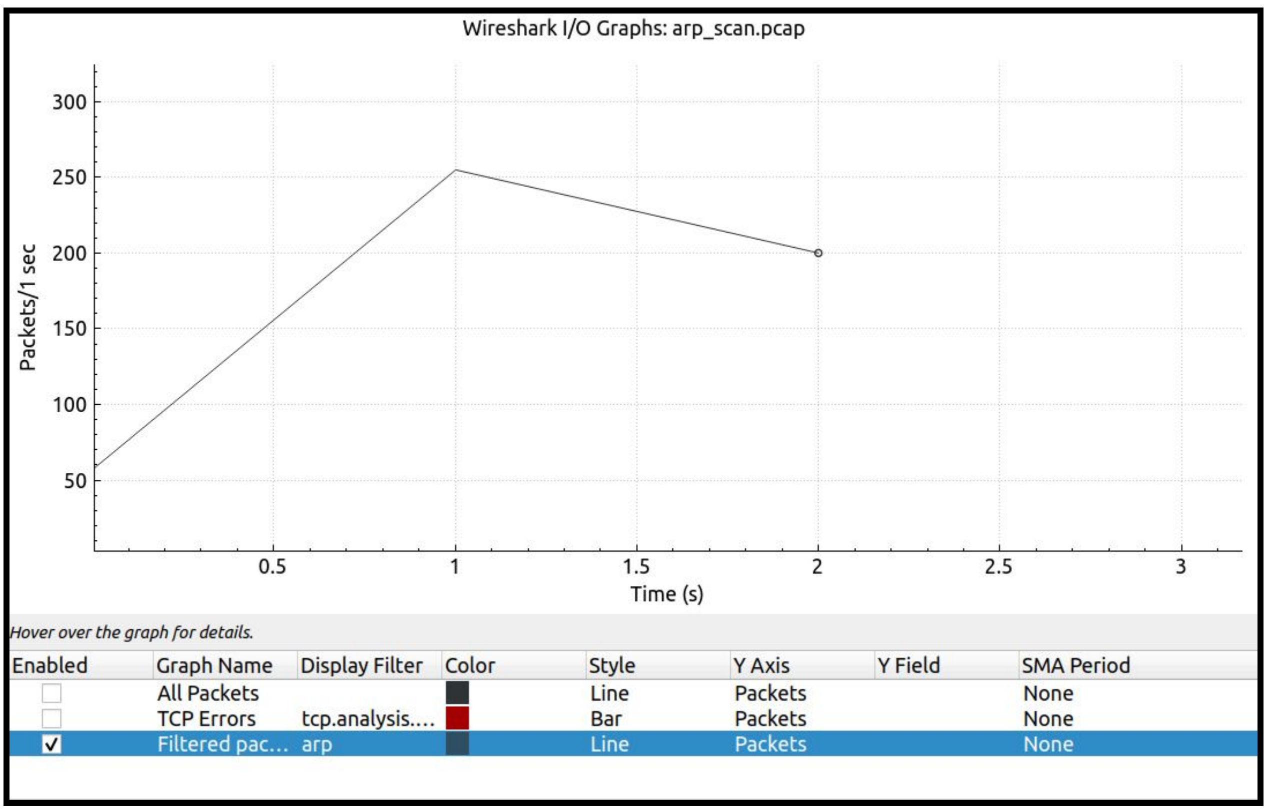
Wireshark detecting the incoming packets in a network. By filtering out the ARP packets, we can check whether there is a possibility of an ARP Sweep attack or not.

Output:

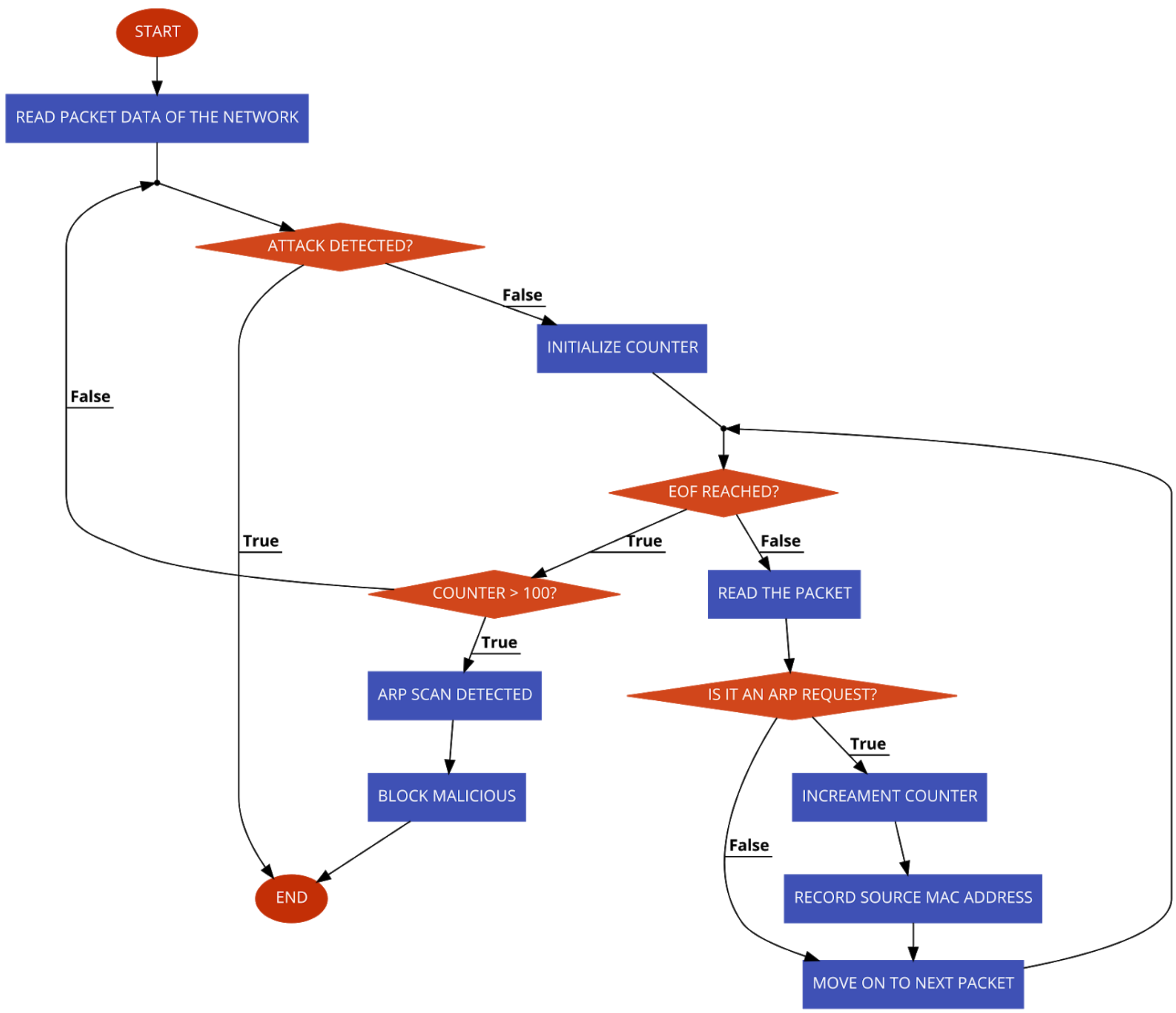
```
============ RESTART: /home/gh0st/Desktop/major_proj/arp_sweep.py ============
[2020-05-19 10:57:27.848685] DEBUG: FileCapture: Creating TShark subprocess with
 parameters: /usr/bin/tshark -l -n -T pdml -Y arp -r ./arp_scan.pcap
[2020-05-19 10:57:27.853462] DEBUG: FileCapture: %s subprocess created
[2020-05-19 10:57:28.289111] DEBUG: FileCapture: EOF reached (sync)
------------------------------------------------------
EXCESSIVE ARP TRAFFIC DETECTED, SOURCE MAC: 00:0c:29:1d:b3:b1
00:0c:29:1d:b3:b1 BLOCKED SUCCESSFULLY
=============================== RESTART: Shell ===============================
```

Blocking the IP address associated with undertaking the ARP Sweep attack in the network. First the attacker was detected and then blocked from accessing the network.

I/O Graph:



Flowchart:


START
READ PACKET DATA OF THE NETWORK
ATTACK DETECTED?
False
INITIALIZE COUNTER
False
EOF REACHED?
True
True
False
COUNTER > 100?
READ THE PACKET
True
ARP SCAN DETECTED
IS IT AN ARP REQUEST?
True
BLOCK MALICIOUS
INCREAMENT COUNTER
False
END
RECORD SOURCE MAC ADDRESS
MOVE ON TO NEXT PACKET

6. Null Scan Attack

```
import pyshark
import operator
import os


def null_scan():
    cap=pyshark.FileCapture('./nmap_null.pcap', display_filter='tcp')
    cap.set_debug()
    null_scan_ip_add={}
    count=0
    while True:
        for pkt in cap:
            if (str(pkt.tcp.flags)=="0x00000000"):
                count+=1
                if str(pkt.ip.src) not in null_scan_ip_add:
                    null_scan_ip_add[pkt.ip.src]=1
                else:
                    null_scan_ip_add[pkt.ip.src]+=1
        if count>100:
            ip = max(null_scan_ip_add.items(), key=operator.itemgetter(1))[0]
            print("--------------------------------------------------------")
            print("NULL SCAN DETECTED, SOURCE IP: %s" %ip)
            return ip


if __name__ ==  "__main__":
    block = null_scan()
    command = "iptables -A INPUT -s " + block + " -j DROP"
```

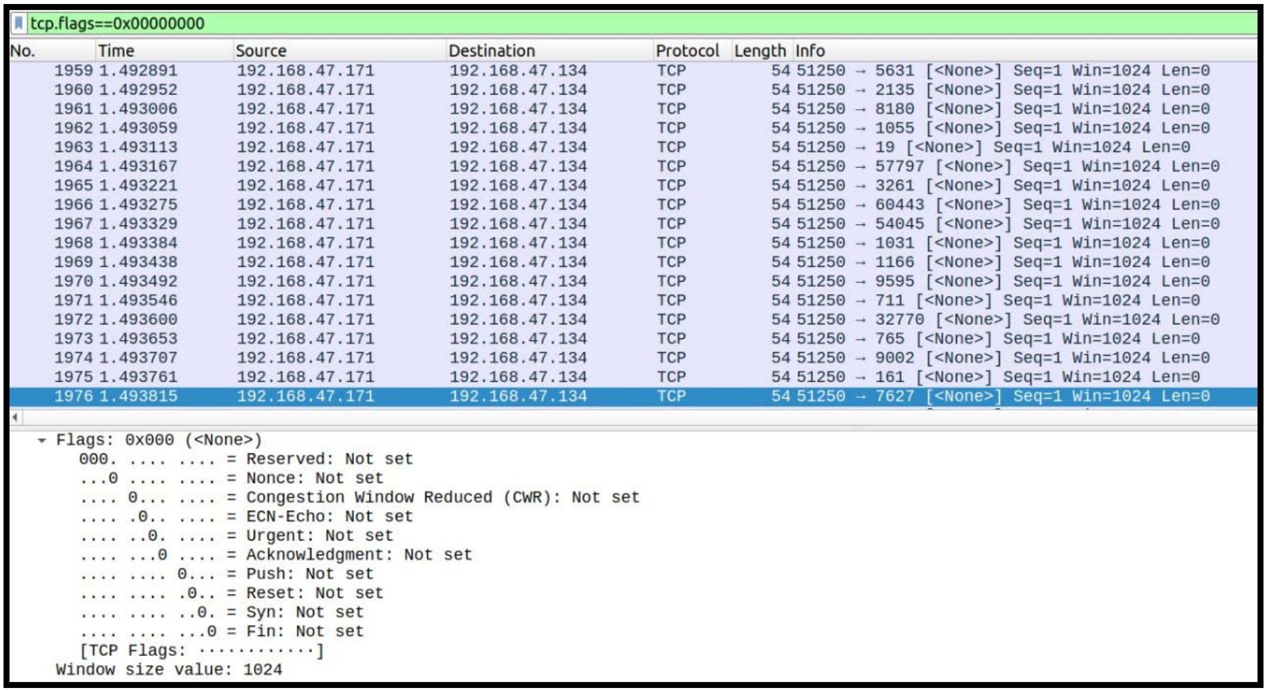
```
if (os.system(command) == 0):
    print("%s BLOCKED SUCCESSFULLY" %block)
```

Output:

```
============ RESTART: /home/gh0st/Desktop/major_proj/null_scan.py ============
[2020-05-19 11:11:54.533442] DEBUG: FileCapture: Creating TShark subprocess with
 parameters: /usr/bin/tshark -l -n -T pdml -Y tcp -r ./nmap_null.pcap
[2020-05-19 11:11:54.539430] DEBUG: FileCapture: %s subprocess created
[2020-05-19 11:11:56.788324] DEBUG: FileCapture: EOF reached (sync)
----------------------------------------------------------
NULL SCAN DETECTED, SOURCE IP: 192.168.47.171
192.168.47.171 BLOCKED SUCCESSFULLY
=============================== RESTART: Shell ================================
```

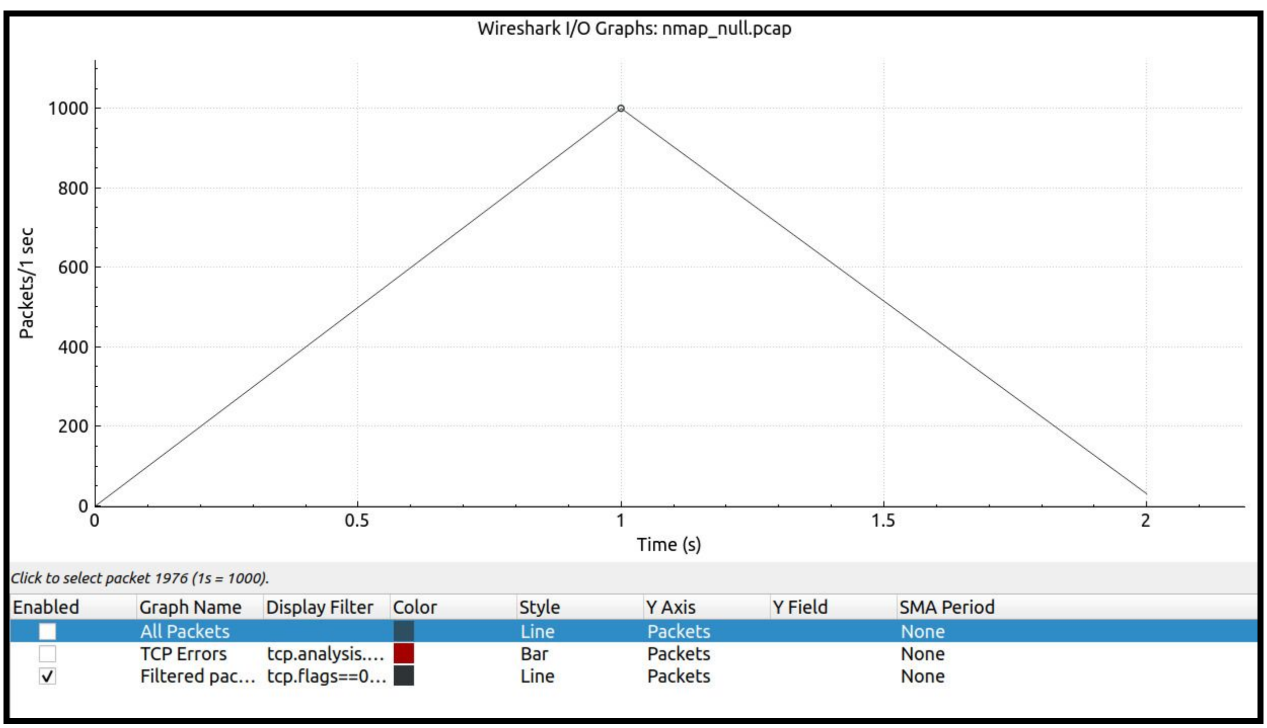
Blocking the IP address associated with undertaking the Null Scan attack in the network. First the attacker was detected and then blocked from accessing the network.

Network Data File Preview:

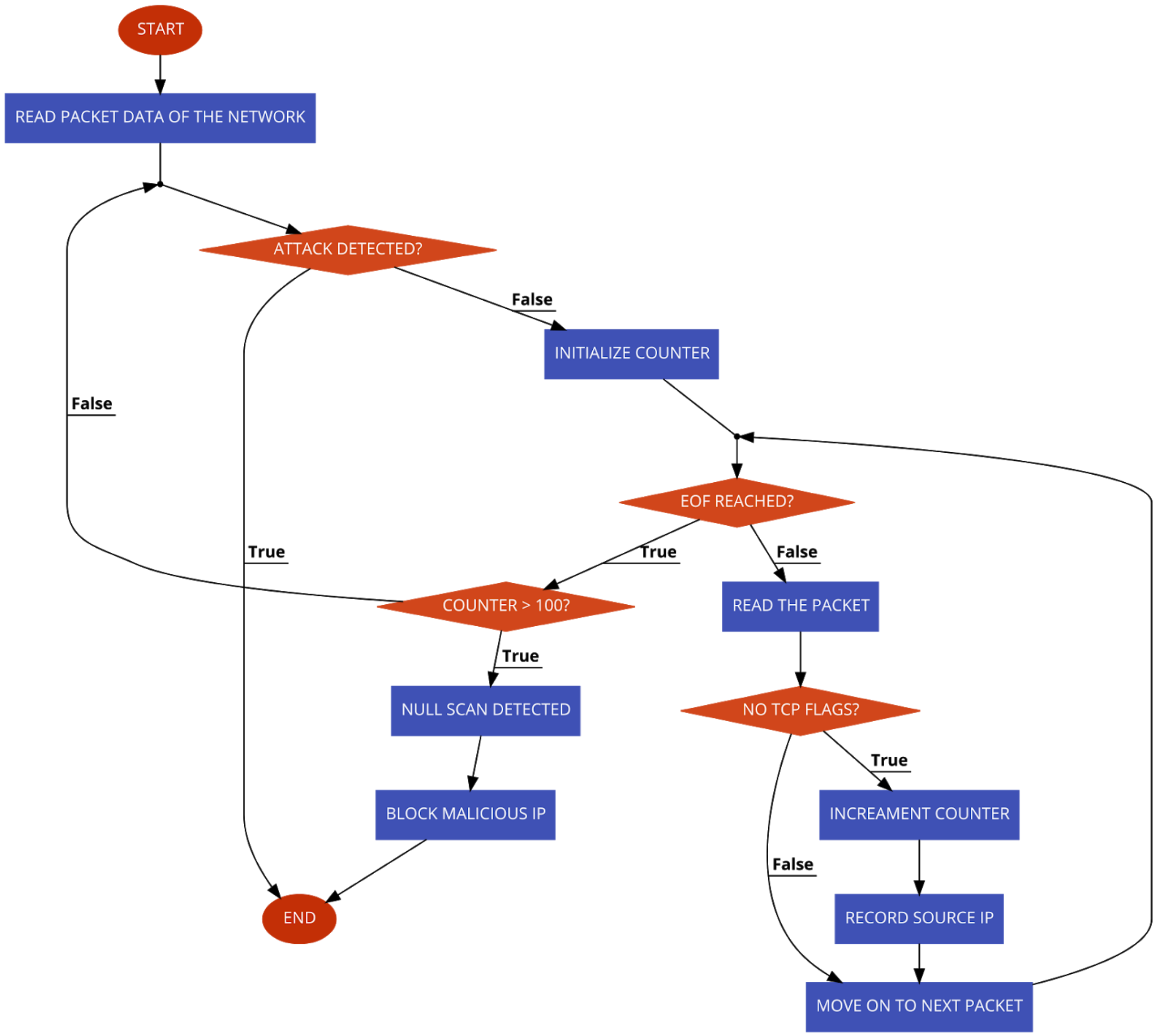


Wireshark detecting the incoming packets in a network. By filtering out the TCP packets, we can check whether there is a possibility of a Null Scan attack or not.

I/O Graph:



Flowchart:


START
READ PACKET DATA OF THE NETWORK
ATTACK DETECTED?
False
INITIALIZE COUNTER
False
EOF REACHED?
True
True
False
COUNTER > 100?
READ THE PACKET
True
NULL SCAN DETECTED
NO TCP FLAGS?
True
BLOCK MALICIOUS IP
INCREAMENT COUNTER
False
END
RECORD SOURCE IP
MOVE ON TO NEXT PACKET

CONCLUSIONS

After observing numerous open networks and traffic in those networks, we came to many conclusions:

First, the network is open to a number of loopholes which can be and are exploited by few notorious users trying to invade our privacy.

Second, the IP address of any system in an open network is obtained by any user in the network by running the simplests of scripts, which can be found anywhere nowadays.

Third, without proper security to overcome these unfriendly users, we open ourselves and our data to be used by these users and even our identity through our IP and MAC addresses.

Finally, we can make an open network safe to use by identifying intruders and blocking their access to the network.

REFERENCES/RESOURCES

[1] thePacketGeek - https://thepacketgeek.com/pyshark-using-the-capture-object/
[2] Network's packet data - https://asecuritysite.com/forensics/pcap?infile=ping.pcap
[3] Analyzing network reconnaissance attempts (by Packt)
[4] Detect/Analyze Scanning Traffic Using Wireshark - PenTest
[5] Python 2.7 Documentation - https://docs.python.org/2.7/
[6] Pyshark Documentation - https://github.com/KimiNewt/pyshark/
[7] CAPEC-304: TCP Null Scan - https://capec.mitre.org/data/definitions/304.html
[8] Nmap Documentation - https://nmap.org/book/scan-methods-null-fin-xmas-scan.html