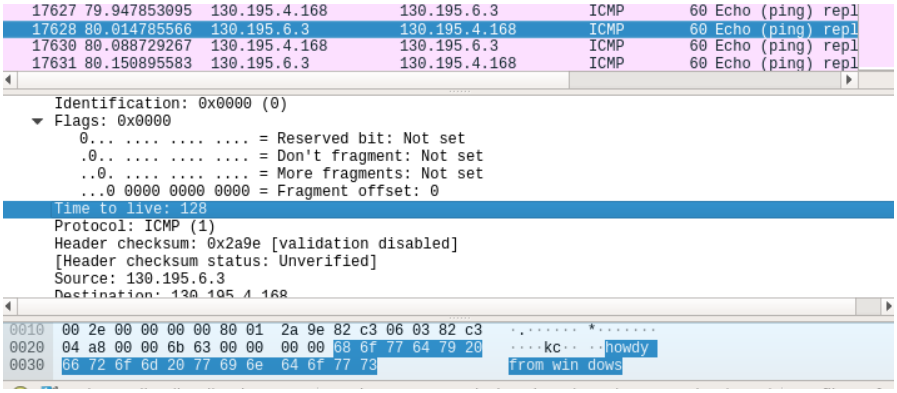
CYBR371
Assignment 2

Gayle Wootton
300578290

Part A: Network Attacks and Vulnerabilities

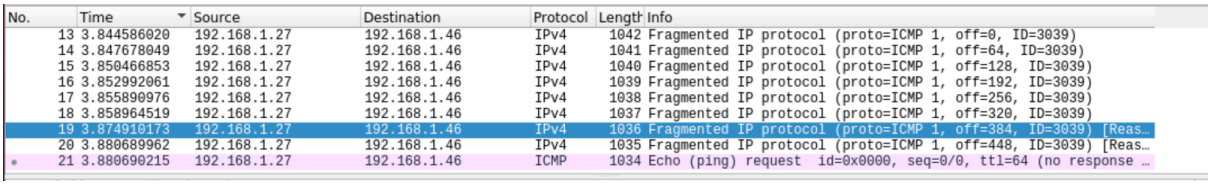
1. Script: windowsReply.py

This script first defines a function that is used as the callback function for processing captured ICMP packets. In the function, we check if the captured packet is an ICMP packet before saving that particular packet's source and destination IP address, ICMP ID and ICMP sequence. I then create a variable to store the payload message for the ICMP echo reply packet. To build the packets so they simulate a Windows host, the TTL value is set to 128. The Time-To-Live field specifies how many hops or network nodes a packet can traverse before it is discarded. 128 is a common TTL value associated with a Windows host. The IP identifier is then set to an ICMP echo reply by setting the value equal to 0. The Windows reply packet is then sent using Scapy's send function before executing the sniff function to capture ICMP packets. For each packet that matches the ICMP filter, the prn parameter with the spoof_icmp function is called.



We can review the captured packets using wireshark on the target VM. As illustrated above, the ICMP packets contain TTL values of 128, illustrating the success of the modified 'Windows' packet with the payload present.

2. Script: teardropAttack.py



I first defined the attack function to take my Ubuntu machine's IP address as the parameter. Variables are then initialised such as the size of the payload, the initial offset and a list of patterns that are used for each fragmented packet's payload. A loop is utilised to create exactly 8 fragmented packets. Every fragment is created using the IP function containing the Ubuntu target IP address, specific ID (to affiliate/connect every fragment during filtering), the offset multiplies by 8 and a flag ('MF') to indicate more fragments. The ICMP layer is added to each fragment before appending these fragments to the packets list. After the loop completes, the final fragment is created (similar to the other packets) but without the 'MF' flag set to indicate that this is the last fragment. The send function is then called to send all of the packets to the target IP. To run the program, the script first checks to see if it is being run as a main program and verifies that a target IP address is provided as a command-line argument.

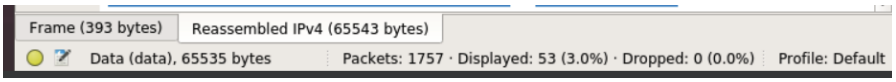
The ICMP teardrop attack takes advantage of the reassembly process of the packet fragments. In IP fragmentation- illustrated in the large payload from the above script, a large packet is divided into smaller fragments to be transmitted over a network. These fragments have a portion of the payload from the original packet. In order to reassemble the fragments, the 'fragment offset' field is used, located in the IP header. In relation to the attack illustrated above, we can see packet 19 has a payload/fragment size equal to 1036 bytes (a portion(fragment) of the original payload). The offset value specifies the position of packet 19 relative to the original packet. In IP fragmentation, the fragmented packets are reassembled

based on their offset values but in the case where the offset values overlap/are misaligned, this leads to a vulnerability known as the teardrop attack. In this case, packet 19 has an offset == 384 meaning the payload starts at byte 3072 (384*8) in the original payload. While packet 20 has an offset of 448, with a payload that starts at byte 3584 (448*8). The teardrop attack occurs in this case as there is an overlapping region between packets 19 and 20. The payload of packet 20 starts into a region that packet 19 had already covered. Packet 19 starts at byte 3072 (offset 384) and extends to 4108 (as packet 19 covers 1036 bytes, fragmented from the original packet). In a successful IP fragmentation reassembly, packet 20 would start right after packet 19, but in this case packet 20 starts at 3584 of the original packet- overlapping the range covered by packet 19. Packet 20 should have an offset of 4108, to ensure the fragments properly align and not overlap, but as the offset < 4108, we have successfully simulated the teardrop attack.

3. Script: dosAttack.py

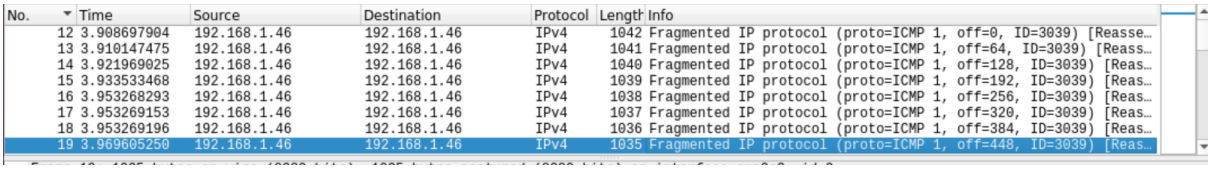
This script combined all 3 DoS attacks- teardrop fragmentation, IP spoofing and ping of death. We first implement the teardrop attack by creating fragmented ICMP packets with overlapping offsets. The attack function generates and sends the teardrop packets- first creating 8 fragmented packets and adding these to the list. I followed the same technique as prior, creating packets with the target machine's IP, setting the ID field to 12345 and adjusting the fragment offset and 'MF' flags to set for the first 8 packets. The list of packets are then sent using the send function. For the IP spoofing attack, I modified the source IP in the IP header in order to impersonate another host, in this case, itself. This address is used as the source IP within the IP header when sending the ICMP packets. The ping of death is the final DoS attack- its function stems from sending oversized ICMP echo request packets with the objective to crash or freeze the targetted machine. This attack was illustrated by sending ICMP packets with a large payload, with the target machine's IP before the packets are sent. The oversized packet simulates a ping of death payload so that when the target machine attempts to reassemble the fragmented packets, it should exceed the maximum packet size and disrupt the machine's reassembly.

Ping of Death:



The fragmented packets were reassembled, exceeding the MTU of 65536 bytes. Albeit, modern operating systems are not significantly impacted by this attack.

Teardrop Fragmentation and IP Spoofing:



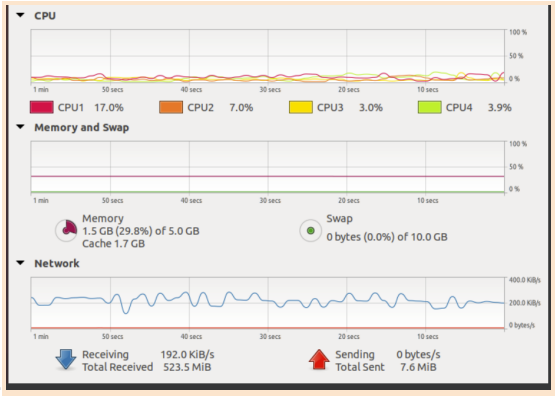
No.	Time	Source	Destination	Protocol	Length	Info
12	3.908697904	192.168.1.46	192.168.1.46	IPv4	1042	Fragmented IP protocol (proto=ICMP 1, off=0, ID=3039) [Reasse…
13	3.910147475	192.168.1.46	192.168.1.46	IPv4	1041	Fragmented IP protocol (proto=ICMP 1, off=64, ID=3039) [Reass…
14	3.921969025	192.168.1.46	192.168.1.46	IPv4	1040	Fragmented IP protocol (proto=ICMP 1, off=128, ID=3039) [Reas…
15	3.933533468	192.168.1.46	192.168.1.46	IPv4	1039	Fragmented IP protocol (proto=ICMP 1, off=192, ID=3039) [Reas…
16	3.953268293	192.168.1.46	192.168.1.46	IPv4	1038	Fragmented IP protocol (proto=ICMP 1, off=256, ID=3039) [Reas…
17	3.953269153	192.168.1.46	192.168.1.46	IPv4	1037	Fragmented IP protocol (proto=ICMP 1, off=320, ID=3039) [Reas…
18	3.953269196	192.168.1.46	192.168.1.46	IPv4	1036	Fragmented IP protocol (proto=ICMP 1, off=384, ID=3039) [Reas…
19	3.969605250	192.168.1.46	192.168.1.46	IPv4	1035	Fragmented IP protocol (proto=ICMP 1, off=448, ID=3039) [Reas…

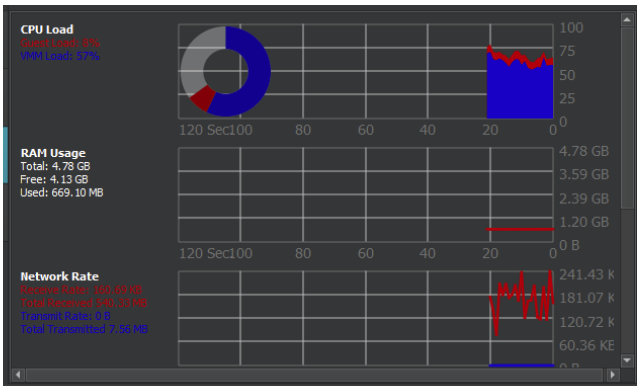
Reviewing packets 12-19, we can see the teardrop attack taking place. These are all fragmented packets with overlapping offsets to simulate the attack. Comparing the final fragment, packet 19, this has an offset of 448 indicating it overlaps with the initial portion and the previous fragments with offsets 64, 128, 192, 256, 320 and 384. These packets are then reassembled as one packet- packet 20, combining all fragmented and malformed packets into a single malformed packet.

4.

The objective of a DDoS attack is to flood the target system or network with so much traffic that it renders it unable to function properly. The attack in question 3 combines three techniques to simulate a DDoS attack. IP spoofing entails forging spoofed IP source addresses, making it appear as if the attack is coming from the same (target) machine. Spoofing the IP address can make the responses appear as if it is coming from the targetted machine, potentially simulating a reflection attack. This type of

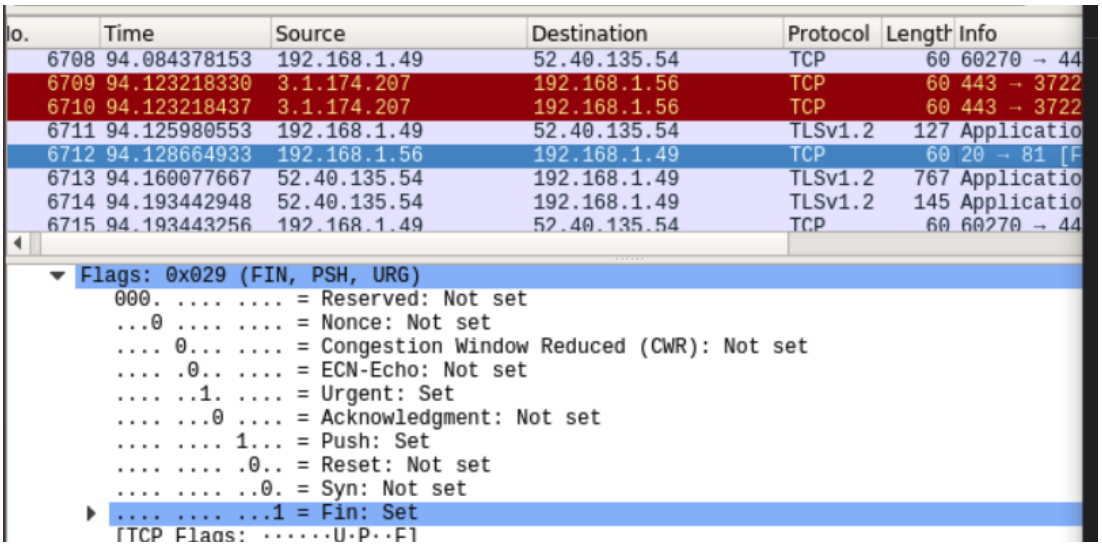
attack could potentially entail a DNS amplification attack where the spoofed IP address sends out requests to DNS servers and tricks them into sending large amounts of unsolicited traffic to the target machine. This causes it to become overwhelmed with traffic, facilitating a DDoS attack. I attempted to simulate a DDoS attack by launching two VM that would execute the script to the target VM at the same time. MY objective was to overwhelm the target machine with a large volume of traffic that it would hopefully crash. I monitored the target machines' resource consumption and performance metrics such as CPU usage, network bandwidth and memory usage to witness any indication of significant impact as a result of the DDoS attack:





There was an increase in network traffic during the attack, albeit not that significant, we can see that network resources were being consumed as illustrated by the unusual spike in traffic.

5. Script: xmasAttack.py



The Xmas tree attack is a type of port scan attack that exploits TCP packets with the purpose of gathering information about the target network, essentially identifying open ports to be potentially exploited. This is due to TCP packets containing a number of flags that are used to indicate (when set) certain characteristics of a packet. The FIN flag in particular indicates that the sender has finished sending the data and is closing the connection. The URG flag indicates that the packet is urgent and should be prioritised. The PSH flag tells the system to immediately send the data to the application layer, forcing the targetted system to process the data quickly. In a Xmas tree attack, the attacker sends a TCP packet that has flags such as FIN, URG and PSH set to 1. These flags denote that the packet is urgent, contains crucial data and is the last packet in the sequence. If a port is open, the targetted system responds with the reset packet (RST) which indicates that the connection has been reset. If a port is closed, the target responds with an ACK (acknowledgment) packet, meaning the packet has been received but the connection was not established. This attack is a stealthy port scan method that possesses the potential to bypass a few intrusion detection systems that monitor more common port scanning attacks. Albeit, this attack is unlikely to be effective due to firewalls and intrusion detection systems' ability to detect and block this attack nowadays.

The script performs a port scan utilising the Xmas Tree packets. This port scan aids in identifying the open ports on our target machine. During the script, the range of ports is defined (1001) before iterating over each port and crafting a Xmas tree packet. The ip destination is set to the target IP and the TCP destination port is set to the current port being scanned. The UPF flags are set to indicate a Xmas tree packet. The packet is then sent with the sr1 function to both send and receive the response. We set the timeout to -1 to indicate no timeout and verbose mode is disabled. To determine if a port is closed, the script checks if a response that was received contains a TCP layer with flags set to “RA” (reset acknowledgement). If neither, this implies that the port is open.

6.

Backscatter traffic is the result of unintentional and unsolicited replies that are produced by a network device in response to specific types of traffic like some distributed denial of service attacks. The primary purpose of DDoS attacks is to produce and send a large number of malicious packets to a targeted server or network with the objective of overloading the target and causing it to become unavailable. The traffic produced by these malicious packets possesses the ability to inadvertently make other network devices generate traffic such as error messages or notifications as a response. This type of traffic is referred to as backscatter traffic.

Only some DDoS attacks produce backscatter traffic as a large majority of these types of attacks frequently use spoofed IP addresses. Meaning the source IP address of the malicious packets is falsified which allows the packet to appear as if it is coming from a legitimate device. The target device then receives the malicious packets and generates a response to the spoof IP address source. Yet, since the source IP address is a spoofed address, this causes the response to be sent to third-party devices that have no affiliation with the attack which then results in the production of backscatter traffic as a response. DDoS attacks that generate backscatter traffic include reflection and amplification attacks. These attacks exploit misconfigured systems and network protocols to generate a large amount of response traffic. Backscatter traffic can be used to secure your network as it can monitor it for typical signs of a potential DDoS attack. If you analyse the backscatter traffic, you can see the source of the attack and implement measures to prevent it. These measures can include filtering malicious traffic, blocking the source IP addresses and deploying firewalls or intrusion detection systems. Furthermore, by analysing the backscatter traffic, one can identify vulnerabilities within the network that could potentially be exploited in a DDoS attack which aids in securing the network and making it increasingly less susceptible to attacks like those from occurring.

7.

1. Network Time Protocol (NTP) Servers: These servers are frequently utilised for time synchronisation in computer networks and can pose as an alternative set of servers for an amplification attack. Due to the large response generated by a small query, NTP servers are vulnerable to amplification attacks. Identifying misconfigured and vulnerable NTP servers can aid in attackers amplifying their attack traffic.
2. Simple Network Management Protocol (SNMP) Servers: These servers are utilised for network management and monitoring but similarly to DNS servers, SNMP servers are also susceptible to amplification attacks due to misconfiguration or weak security controls. To execute and generate large responses directed to the target host, an attacker can send crafted SNMP queries.
3. Memcached Servers: Memcached is an in-memory caching system used to speed up database-driven websites. Misconfigured Memcached servers can be exploited for amplification attacks. By sending a small query to a Memcached server, an attacker can cause it to generate a large response directed at the target host.
4. Lightweight Directory Access Protocol (LDAP) Servers: LDAP servers are used for accessing and managing directory information services. LDAP servers can be abused for amplification attacks if

they are misconfigured or allow anonymous queries. By sending a small LDAP query, an attacker can elicit a larger response from the server, which can overwhelm the target host.

Part B: Firewalls and Intrusion Detection Systems

1.

Firewalls, intrusions prevention systems and honeypots use certain properties from the TCP protocol such as window size and maximum segment size which slow the propagation of worms throughout the network- this is a technique known as TCP rate limiting. TCP rate limiting works by editing the TCP header field (window size and maximum segment size) on packets incoming, allowing the transmission of data to slow down. These TCP properties such as the window size field specify the amount of data that is able to be sent without receiving the ACK (acknowledgement). The maximum segment size field specifies the largest amount of data that is able to be sent in a single TCP segment.

Step 1 Monitoring: This step entails monitoring the network traffic passing through the firewall, intrusion prevention system or honeypot to identify any suspicious activity that may indicate that a worm is present.

Step 2 Identification: This next step is to identify the window size and maximum segment size by the worm which is done by analysing the TCP packets utilised by the worm.

Step 3 Reducing window size and maximum segment size: This step involves modifying and reducing the maximum segment size and window size used by the TCP connections bearing the worm traffic.

Step 4 Slow down propagation: Reducing the window size and maximum segment size, the honeypot, intrusion prevention system or firewall is able to successfully slow down the propagation of the worm across the network. This is owed to that reducing the window size and maximum segment size within the TCP header allows the rate of transmission in a TCP connection to be slowed down as the amount of data that can be transmitted over a TCP connection is reduced, in turn limiting the speed of the data transfer.

2A) Firewall Policy Table

No	Transport Protocol	Protocol	Source IP/Network	Dest. IP/Network	Source Port	Dest. Port	Action
1	TCP	HTTP	192.168.2.0/24	130.195.1.1/24	1024-65535	80	Allow
2	TCP	HTTP	192.168.3.0/24	130.195.1.1/24	1024-65535	80	Allow
3	TCP	HTTPS	192.168.2.0/24	130.195.1.1/24	1024-65535	443	Allow
4	TCP	HTTPS	192.168.3.0/24	130.195.1.1/24	1024-65535	443	Allow
5	TCP	HTTP	130.195.4.0/24	130.195.1.1/24	1024-65535	80	Allow
6	TCP	HTTPS	130.195.4.0/24	130.195.1.1/24	1024-65535	443	Allow
7	TCP	HTTP	ANY	130.195.1.1/24	0-1023	80	Drop
8	ICMP	-	130.195.4.0/24	130.195.1.1/24	-	-	Rate-limit (5/s)
9	UDP	-	ANY	130.195.1.1/24	-	-	Drop frags
10	TCP	SSH	130.195.4.1/24	130.195.1.1/24	1024-65535	22	Allow
11	TCP	SSH	ANY	130.195.1.1/24	-	22	Rate-limit (3/s)
12	TCP	-	130.195.4.0/24	130.195.1.1/24	0-65535	0-655	Drop (FIN,

						35	URG, PSH)
13	ANY	-	ANY	130.195.1.1/24	2002, 2004	-	Drop
14	ANY	-	ANY	ANY	-	2002, 2004	Drop
15	UDP	DNS	192.168.2.0/24	8.8.8.8	53	53	Allow
16	UDP	DNS	192.168.3.0/24	8.8.8.8	53	53	Allow
17	TCP	HTTP	130.195.1.1/24	130.195.4.100	1024-65535	80	Allow
18	ANY	-	130.195.1.1/24	ANY	-	-	Drop (except rule 17)
19	TCP/UDP	-	130.195.4.0/24	130.195.1.1/24	-	8088, 4044	Drop if matches Worm signature

2B) IPTABLES

 # Rule 1 Allow HTTP Connections from 192.168.2.0/24 to 130.195.1.1/24
iptables -A INPUT -p tcp -s 192.168.2.0/24 -d 130.195.1.1/24 --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p tcp -s 130.195.1.1/24 -d 192.168.2.0/24 --sport 80 -m state --state ESTABLISHED -j ACCEPT

 # Rule 2 Allow HTTP Connections from 192.168.3.0/24 to 130.195.1.1/24
iptables -A INPUT -p tcp -s 192.168.3.0/24 -d 130.195.1.1/24 --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p tcp -s 130.195.1.1/24 -d 192.168.3.0/24 --sport 80 -m state --state ESTABLISHED -j ACCEPT

 # Rule 3 Allow HTTPS Connections from 192.168.2.0/24 to 130.195.1.1/24
iptables -A INPUT -p tcp -s 192.168.2.0/24 -d 130.195.1.1/24 --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p tcp -s 130.195.1.1/24 -d 192.168.2.0/24 --sport 443 -m state --state ESTABLISHED -j ACCEPT

 # Rule 4 Allow HTTPS Connections from 192.168.3.0/24 to 130.195.1.1/24
iptables -A INPUT -p tcp -s 192.168.3.0/24 -d 130.195.1.1/24 --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p tcp -s 130.195.1.1/24 -d 192.168.3.0/24 --sport 443 -m state --state ESTABLISHED -j ACCEPT

 # Rule 5 Allow HTTP Connections from 130.195.4.0/24 to 130.195.1.1/24
iptables -A INPUT -p tcp -s 130.195.4.0/24 -d 130.195.1.1/24 --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p tcp -s 130.195.1.1/24 -d 130.195.4.0/24 --sport 80 -m state --state ESTABLISHED -j ACCEPT

 # Rule 6 Allow HTTPS Connections from 130.195.4.0/24 to 130.195.1.1/24

iptables -A INPUT -p tcp -s 130.195.4.0/24 -d 130.195.1.1/24 --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p tcp -s 130.195.1.1/24 -d 130.195.4.0/24 --sport 443 -m state --state ESTABLISHED -j ACCEPT

 # Rule 7 Drop HTTP Connections from ANY to 130.195.1.1/24 except for established connections
iptables -A INPUT -p tcp -d 130.195.1.1/24 --dport 80 -m state --state NEW -m recent --set --name HTTP
iptables -A INPUT -p tcp -d 130.195.1.1/24 --dport 80 -m state --state NEW -m recent --rcheck --seconds 10 --hitcount 3 --rttl --name HTTP -j DROP
iptables -A INPUT -p tcp -s 130.195.1.1/24 --sport 80 -m state --state ESTABLISHED -j ACCEPT

 # Rule 8 Rate-limit ICMP Traffic from 130.195.4.0/24 to 130.195.1.1/24 to 5/s
iptables -A INPUT -p icmp -s 130.195.4.0/24 -d 130.195.1.1/24 -m limit --limit 5/s -j ACCEPT

 # Rule 9 Drop Fragmented Packets from ANY to 130.195.1.1/24
iptables -A INPUT -f -d 130.195.1.1/24 -j DROP

 # Rule 10 Allow SSH Connections from 130.195.4.1/24 to 130.195.1.1/24
iptables -A INPUT -p tcp -s 130.195.4.1/24 -d 130.195.1.1/24 --dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p tcp -s 130.195.1.1/24 -d 130.195.4.1/24 --sport 22 -m state --state ESTABLISHED -j ACCEPT

 # Rule 11 Rate-limit SSH Connections from ANY to 130.195.1.1/24 to 3/s
iptables -A INPUT -p tcp -d 130.195.1.1/24 --dport 22 -m state --state NEW -m recent --set --name SSH
iptables -A INPUT -p tcp -d 130.195.1.1/24 --dport 22 -m state --state NEW -m recent --rcheck --seconds 10 --hitcount 3 --rttl --name SSH -j DROP

#Rule 12 Drop PACKETS with FIN, URG, and PSH flags set from 130.195.4.0/24 to 130.195.1.1/24
iptables -A INPUT -p tcp -s 130.195.4.0/24 -d 130.195.1.1/24 --tcp-flags FIN,URG,PSH FIN,URG,PSH -j DROP

 # Rule 13 Drop PACKETS from ANY to 130.195.1.1/24 with destination ports 2002 or 2004
iptables -A INPUT -p tcp -d 130.195.1.1/24 --dport 2002 -j DROP
iptables -A INPUT -p tcp -d 130.195.1.1/24 --dport 2004 -j DROP
iptables -A INPUT -p udp -d 130.195.1.1/24 --dport 2002 -j DROP
iptables -A INPUT -p udp -d 130.195.1.1/24 --dport 2004 -j DROP

 # Rule 14 Drop PACKETS from ANY to ANY with destination ports 2002 or 2004
iptables -A INPUT -p tcp --dport 2002 -j DROP
iptables -A INPUT -p tcp --dport 2004 -j DROP
iptables -A INPUT -p udp --dport 2002 -j DROP
iptables -A INPUT -p udp --dport 2004 -j DROP

 # Rule 15 Allow DNS Traffic from 192.168.2.0/24 to 8.8.8.8
iptables -A OUTPUT -p udp -s 192.168.2.0/24 -d 8.8.8.8 --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -p udp -s 8.8.8.8 -d 192.168.2.0/24 --sport 53 -m state --state ESTABLISHED -j ACCEPT

 # Rule 16 Allow DNS Traffic from 192.168.3.0/24 to 8.8.8.8
iptables -A OUTPUT -p udp -s 192.168.3.0/24 -d 8.8.8.8 --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -p udp -s 8.8.8.8 -d 192.168.3.0/24 --sport 53 -m state --state ESTABLISHED -j ACCEPT

 # Rule 17 Allow HTTP Connections from 130.195.1.1/24 to 130.195.4.100
iptables -A OUTPUT -p tcp -s 130.195.1.1/24 -d 130.195.4.100 --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -p tcp -s 130.195.4.100 -d 130.195.1.1/24 --sport 80 -m state --state ESTABLISHED -j ACCEPT

 # Rule 18 Drop ALL Traffic to 130.195.1.1/24 except for rule 17
iptables -A INPUT -d 130.195.1.1/24 -j DROP

 # Rule 19 Drop Worm signatures on TCP/UDP from 130.195.4.0/24 to 130.195.1.1/24 on ports 8088, 4044
iptables -A FORWARD -p tcp -s 130.195.4.0/24 -d 130.195.1.1/24 --dport 8088 -m string --hex-string "|03 0C FE BB A2|PASS : RECV" --algo bm --from 0 --to 40 -j DROP
iptables -A FORWARD -p udp -s 130.195.4.0/24 -d 130.195.1.1/24 --dport 4044 -m string --hex-string "|03 0C FE BB A2|PASS : RECV" --algo bm --from 0 --to 40 -j DROP