

Q1

1. Network-Based IDS (NIDS):

- **Usage:** Monitors network traffic in real-time and identifies suspicious patterns or anomalies.
- **Circumstances:** Ideal for large-scale networks where monitoring at the perimeter is crucial.
- **Capabilities:** Analyzes packets and identifies malicious activity.

2. Host-Based IDS (HIDS):

- **Usage:** Monitors activities on individual computers or hosts, looking for suspicious behavior or changes.
- **Circumstances:** Suitable for environments where internal threats are a concern or for critical servers that require close monitoring.
- **Capabilities:** Analyzes system logs, file integrity, and other host-specific data to detect abnormal activities.

3. Signature-Based IDS/IPS:

- **Usage:** Matches known patterns or signatures of known threats.
- **Circumstances:** Effective against known attack types with well-defined signatures.
- **Capabilities:** Compares network traffic or system activity against a database of known attack signatures and triggers an alert or takes action if a match is found.

4. Anomaly-Based IDS/IPS:

- **Usage:** Identifies deviations from normal behavior patterns, rather than relying on predefined signatures.
- **Circumstances:** Useful for detecting previously unknown threats or variations in attack patterns.
- **Capabilities:** Learns what "normal" behavior looks like and raises an alert or takes action when significant deviations are detected.

Q2

1. Sniffer Mode:

In Sniffer mode, Snort functions as a passive network sniffer. It analyzes packets on the network without actively blocking or modifying them.

2. Packet Logger Mode:

Packet Logger mode extends the capabilities of Sniffer mode by logging captured packets to disk for later analysis. In this mode, Snort records packet data to log files, enabling administrators to review and investigate network activity over an extended period.

3. Network Intrusion Detection System (NIDS) Mode:

NIDS mode is the most active and widely used operational mode of Snort. In this mode, Snort actively analyzes network traffic, compares it against predefined rules or signatures, and generates alerts or takes actions when suspicious or malicious activity is detected. NIDS mode allows Snort to function as a real-time intrusion detection system.

Q3

1. Traditional Rules

Snort's intrusion detection and prevention system relies on the presence of Snort rules to protect networks, and those rules consist of two main sections:

The rule header defines the action to take upon any matching traffic, as well as the protocols, network addresses, port numbers, and direction of traffic that the rule should apply to.

The rule body section defines the message associated with a given rule, and most importantly the payload and non-payload criteria that need to be met in order for a rule to match. Although rule options are not required, they are essential for making sure a given rule targets the right traffic.

2. Service Rules

service rules let rule writers target a particular service regardless of the IP addresses or ports being used in a given network flow.

3. File Rules

allow rule writers to create rules to match a particular file regardless of the protocol, source IPs, destination IPs, ports, and service.

Snort is able to process files that are sent using any of the following application-layer protocols: HTTP, SMTP, POP3, IMAP, SMB, FTP.

4. File Identification Rules

File identification rules take advantage of Snort's detection engine to enable file type identification. These rules are basic Snort 3 rules, but instead of alerting on and/or blocking traffic, they identify files based on the contents of that file and then define a file type that can be used in subsequent rules

Q4

The distance keyword is similar to offset but is relative to a preceding content match instead of the start of the payload/buffer. It tells Snort to look skip X number of bytes after the last content match before looking for this one.

The within keyword is similar to depth but is relative to a preceding content match instead of the start of the payload. It tells Snort to look this content match within X number of bytes of the last one.

The replace rule option is used to overwrite prior matching content with the string provided to the option. This option should be used with the rewrite rule action, and it works for raw packets only.

The http_stat_code sticky buffer contains the status code field of an HTTP response status line. This includes values such as 200, 403, and 404.

The metadata option adds additional and arbitrary information to a rule in the form of key-value pairs

Part 1: drop nmap ping request

```
drop icmp any any -> 192.168.179.101 any (msg:"Ping request blocked";
sid:10000004;)
```

```
snort -Q -A console --daq nfg --daq-var device=enp0s3 --daq-var queue=1
```

```
iptables -t nat -I PREROUTING -j NFQUEUE --queue-num 1
```

```
iptables -I INPUT -j NFQUEUE --queue-num 1
```

The screenshot shows the Zenmap application window. At the top, there are tabs for "can", "Tools", "Profile", and "Help". Below these, there's a section for setting the target and profile. The target is set to "192.168.179.101" and the profile is empty. There are "Scan" and "Cancel" buttons. Below this, the command field contains "nmap -PM 192.168.179.101 --disable-arp-ping".

The main area has several tabs: "Hosts", "Services", "Nmap Output", "Ports / Hosts", "Topology", "Host Details", and "Scans". The "Hosts" tab is selected, showing a list of hosts: "S Host" and two entries for "192.168.179.101" and "192.168.179.102".

The "Nmap Output" tab is also visible, showing the command "nmap -PM 192.168.179.101 --disable-arp-ping" and its results:

```
nmap -PM 192.168.179.101 --disable-arp-ping
Starting Nmap 7.93 ( https://nmap.org ) at 2023-11-15 19:08 Iran Standard Time
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 2.66 seconds
```

Below the Zenmap window, a terminal window displays the raw nmap output, showing multiple "Ping request blocked" messages and warnings about preprocessors.

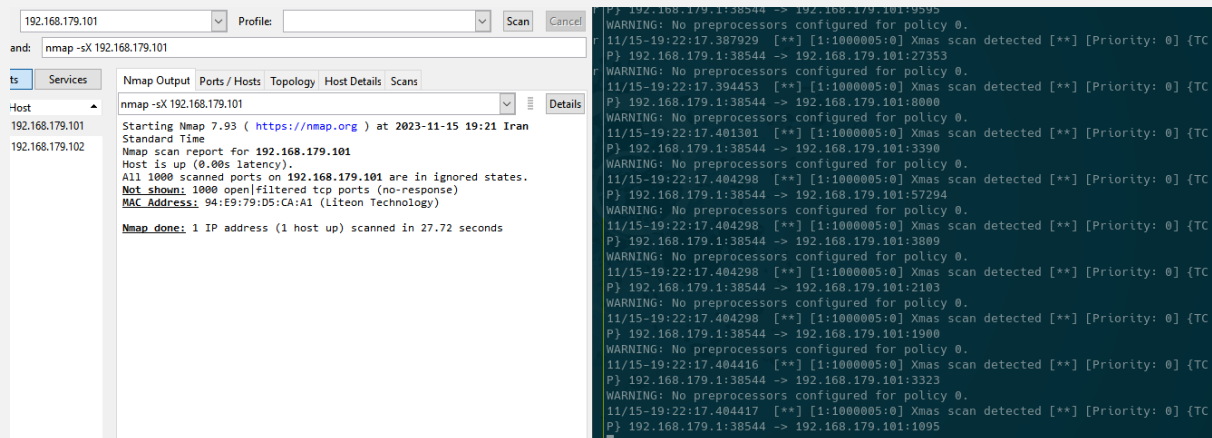
Bypass: we can use -Pn flag in nmap to bypass this rule

Part 2: Xmas Scan

Rule:

```
alert tcp any any -> 192.168.179.101 any (flags: FPU; msg:"Xmas scan detected";  
sid:1000005;)
```

Result:



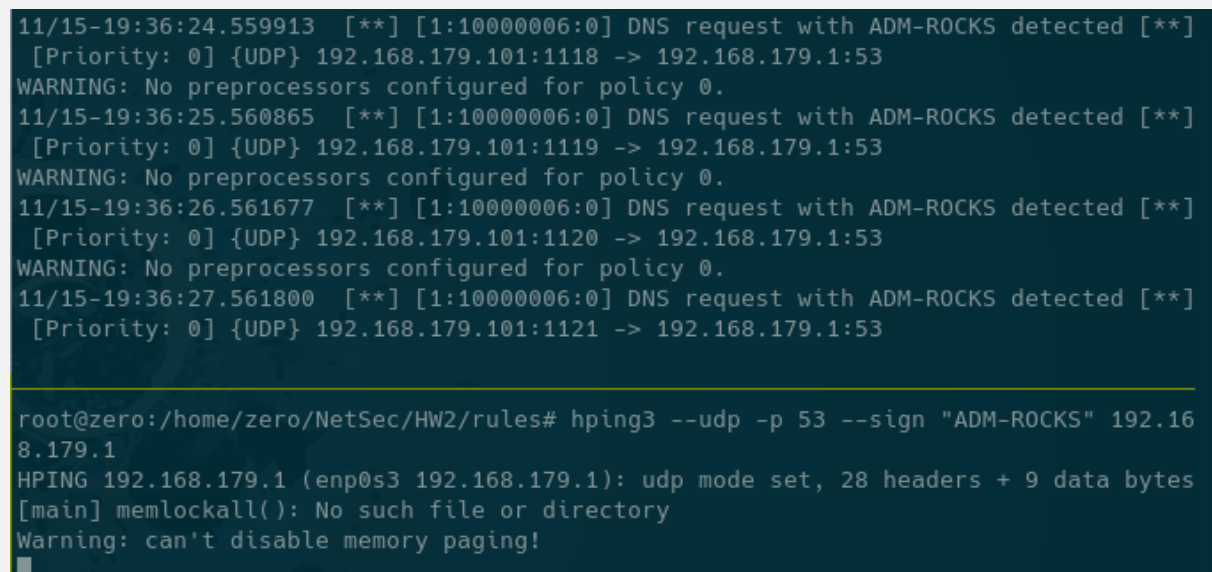
The screenshot shows the Nmap GUI interface. The target IP is 192.168.179.101. The scan command is nmap -sX 192.168.179.101. The output shows that the scan is complete and 1 IP address is up. On the right, a terminal window displays the raw scan results, showing multiple Xmas scan detections (flags: FPU) for various ports on the target IP.

Part 3: ADM-ROCKS DNS Request

Rule:

```
alert udp 192.168.179.101 any -> any 53 (content:"ADM-ROCKS"; msg:"DNS  
request with ADM-ROCKS detected"; sid:10000006;)
```

Result: (using tmux)



The screenshot shows a terminal window with multiple lines of output. Each line represents a DNS request with ADM-ROCKS detected, showing the timestamp, priority, and destination IP. Below the terminal output, a command is executed: hping3 --udp -p 53 --sign "ADM-ROCKS" 192.168.179.1. The output of this command shows that the UDP mode is set and the data bytes are 28 headers + 9 data bytes.

Part 4: nmap SSH scan

Rule:

```
alert tcp any any -> 192.168.179.101 22 (dsize:0; flags:S; msg:"TCP Port 22, SYN Flag Set, No Payload"; sid:1000007;)
```

Result:

Zenmap

Scan Tools Profile Help

Target: 192.168.179.101 Profile: Scan Cancel

Command: nmap -sV -p 22 192.168.179.101

Hosts Services

OS Host

192.168.179.101

192.168.179.102

Filter Hosts

Nmap Output Ports / Hosts Topology Host Details Scans

nmap -sV -p 22 192.168.179.101 Details

Starting Nmap 7.93 (<https://nmap.org>) at 2023-11-15 20:41 Iran Standard Time
NSOCK ERROR [0.0920s] ssl_init_helper(): OpenSSL legacy provider failed to load.

Nmap scan report for 192.168.179.101
Host is up (0.00s latency).

PORT	STATE	SERVICE	VERSION
22/tcp	filtered	ssh	

MAC Address: 94:E9:79:D5:CA:A1 (Liteon Technology)

Service detection performed. Please report any incorrect results at <https://nmap.org/submit/> .
Nmap done: 1 IP address (1 host up) scanned in 6.94 seconds

WARNING: No preprocessors configured for policy 0.
WARNING: No preprocessors configured for policy 0.
11/15-20:41:35.566089 [**] [1:1000007:0] TCP Port 22, SYN Flag Set [**] [Priority: 0] {TCP} 192.168.179.1:35554 -> 192.168.179.101:22
WARNING: No preprocessors configured for policy 0.
11/15-20:41:35.669407 [**] [1:1000007:0] TCP Port 22, SYN Flag Set [**] [Priority: 0] {TCP} 192.168.179.1:35556 -> 192.168.179.101:22

Part 5: HTTP DoS

Rule:

```
alert tcp any any -> 192.168.179.101 80 (msg:"Potential HTTP DoS Attack";  
threshold:type threshold, track by_src, count 5, seconds 10; flags:!S; sid:1000008;)
```

Test:

```
hping3 -c 60 -p 80 192.168.179.101
```

Result:

```
WARNING: No preprocessors configured for policy 0.  
11/15-21:11:39.892403  [**] [1:100008:0] Potential HTTP DoS Attack [**] [Priority: 0] {TCP} 192.168.179.104:1895 -> 192.168.179.101:80  
WARNING: No preprocessors configured for policy 0.
```

Part 6: SQL Injection

Rule:

```
alert tcp any any -> 192.168.179.101 any (msg:"SQL Injection Attempt";  
flow:established,to_server; content:"" or 1=1--"; sid:1000009;)
```

Test: (input.txt contains the payload string)

```
hping3 -c 3 -p 80 -S -d 40 -E input.txt 192.168.179.101
```

```
root@server1:/home/zero# hping3 -c 3 -p 80 -S -d 40 -E input.t  
xt 192.168.179.101  
HPING 192.168.179.101 (enp0s3 192.168.179.101): S set, 40 heade  
rs + 40 data bytes  
[main] memlockall(): Operation not supported  
Warning: can't disable memory paging!  
  
--- 192.168.179.101 hping statistic ---  
3 packets transmitted, 0 packets received, 100% packet loss  
round-trip min/avg/max = 0.0/0.0/0.0 ms  
root@server1:/home/zero#
```

Result:

```
11/15-21:47:08.788991  [**] [1:1000009:0] SQL Injection Attempt [**] [Priority: 0]  
{TCP} 192.168.179.104:2115 -> 192.168.179.101:80  
WARNING: No preprocessors configured for policy 0.  
WARNING: No preprocessors configured for policy 0.  
WARNING: No preprocessors configured for policy 0.  
11/15-21:47:09.789718  [**] [1:1000009:0] SQL Injection Attempt [**] [Priority: 0]  
{TCP} 192.168.179.104:2116 -> 192.168.179.101:80  
WARNING: No preprocessors configured for policy 0.  
11/15-21:47:10.790390  [**] [1:1000009:0] SQL Injection Attempt [**] [Priority: 0]  
{TCP} 192.168.179.104:2117 -> 192.168.179.101:80
```

Part 7: index.php access

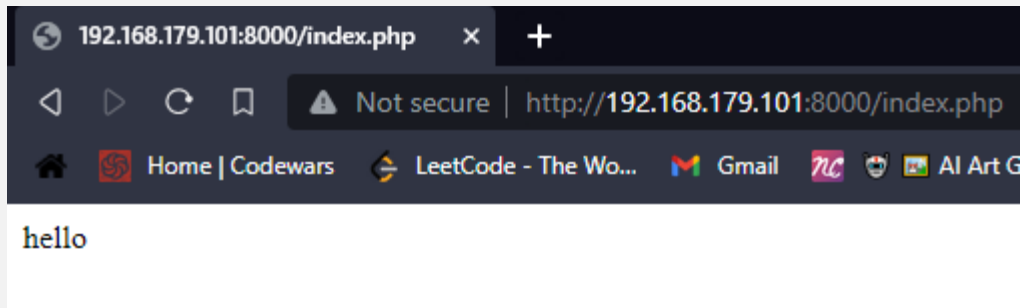
Rule:

```
alert tcp any any -> 192.168.179.101 8000 (msg:"Unauthorized Access to  
index.php";flow:to_server,established; content:"GET /index.php"; sid:1000010;)
```

I started a simple php server with a simple index file with this command:

```
php -S 0.0.0.0:8000
```

Browser:



Wireshark:

1	0.000000	192.168.179.1	192.168.179.101	TCP	66	8511 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.000277	192.168.179.101	192.168.179.1	TCP	66	8000 → 8511 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
3	0.000374	192.168.179.1	192.168.179.101	TCP	54	8511 → 8000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
4	0.000605	192.168.179.1	192.168.179.101	HTTP	506	GET /index.php HTTP/1.1
5	0.000746	192.168.179.101	192.168.179.1	TCP	60	8000 → 8511 [ACK] Seq=1 Ack=453 Win=64128 Len=0
6	0.001092	192.168.179.101	192.168.179.1	TCP	223	8000 → 8511 [PSH, ACK] Seq=1 Ack=453 Win=64128 Len=169 [TCP segment of a reassembled P
7	0.001206	192.168.179.101	192.168.179.1	HTTP	60	HTTP/1.1 200 OK (text/html)
8	0.001270	192.168.179.1	192.168.179.101	TCP	54	8511 → 8000 [ACK] Seq=453 Ack=177 Win=2102016 Len=0
9	0.002908	192.168.179.1	192.168.179.101	TCP	54	8511 → 8000 [FIN, ACK] Seq=453 Ack=177 Win=2102016 Len=0
10	0.003152	192.168.179.101	192.168.179.1	TCP	60	8000 → 8511 [ACK] Seq=177 Ack=454 Win=64128 Len=0
11	5.213874	LiteonTe_d5:ca:a1	0a:00:27:00:00:17	ARP	60	Who has 192.168.179.1? Tell 192.168.179.101
12	5.213891	0a:00:27:00:00:17	LiteonTe_d5:ca:a1	ARP	42	192.168.179.1 is at 0a:00:27:00:00:17

< >

Ethernet II, Src: 0a:00:27:00:00:17 (0a:00:27:00:00:17), Dst: LiteonTe_d5:ca:a1 (94:e9:79:d5:ca:a1)

Internet Protocol Version 4, Src: 192.168.179.1, Dst: 192.168.179.101

Transmission Control Protocol, Src Port: 8511, Dst Port: 8000, Seq: 1, Ack: 1, Len: 452

Source Port: 8511

Destination Port: 8000

[Stream index: 0]

[Conversation completeness: Complete, WITH_DATA (31)]

[TCP Segment Len: 452]

Sequence Number: 1 (relative sequence number)

Sequence Number (raw): 346577719

[Next Sequence Number: 453 (relative sequence number)]

Acknowledgment Number: 1 (relative ack number)

Acknowledgment number (raw): 18839336

0101 = Header Length: 20 bytes (5)

0020 b3 65 21 3f 1f 40 14 a8 5b 37 01 1f 77 28 50 18 e!? @.. [7-w(P-

0030 20 14 e3 34 00 00 47 45 54 20 7e 02 2e 25 03 72 44 GE T/index

0040 2e 70 68 70 20 48 54 54 50 2f 31 2e 31 0d 0a 48 php HTTP/1.1-H

0050 6f 73 74 3a 20 31 39 32 2e 31 36 38 2e 31 37 39 ost: 192.168.179

0060 2e 31 30 31 3a 38 30 30 30 0d 0a 43 6f 6e 6e 65 .101:8000 -Conne

0070 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 ction: k eep-aliv

0080 65 0d 0a 43 61 63 68 65 2d 43 6f 6e 74 72 6f 6c e-Cache -Control

Result:

```
WARNING: No preprocessors configured for policy 0.
WARNING: No preprocessors configured for policy 0.
WARNING: No preprocessors configured for policy 0.
11/15-22:33:46.580112  [*] [1:1000010:0] Unauthorized Access to index.php [*] [Pr
iority: 0] {TCP} 192.168.179.1:8511 -> 192.168.179.101:8000
WARNING: No preprocessors configured for policy 0.
WARNING: No preprocessors configured for policy 0.
```


Bonus 1: Suricata

Suricata is an open-source network threat detection engine developed by the Open Information Security Foundation (OISF). It is designed to monitor network traffic and detect potential security threats or intrusions. Suricata is often used as an intrusion detection system (IDS), intrusion prevention system (IPS), and network security monitoring (NSM) tool.

1. Multi-Threaded Architecture:

- Suricata is designed to take advantage of multi-core processors, allowing it to handle high network traffic loads efficiently.

2. Traffic Capture and Analysis:

- Suricata can capture and analyze network traffic in real-time using various capture methods, including PCAP files and AF_PACKET.

3. Rule-Based Detection:

- Similar to Snort, Suricata uses rule-based detection to identify and alert on malicious activity. It supports the Emerging Threats and VRT rule sets.

4. Protocol Support:

- Suricata supports a wide range of network protocols, including IPv4, IPv6, TCP, UDP, ICMP, HTTP, DNS, and more.

5. File Extraction:

- Suricata can extract files from network traffic for further analysis, aiding in the identification of malware or other malicious content.

6. Integration with Other Tools:

- Suricata can integrate with other security tools and platforms, such as intrusion prevention systems, SIEMs (Security Information and Event Management), and threat intelligence feeds.

7. Performance:

- The multi-threaded architecture and efficient handling of network traffic contribute to Suricata's performance, making it suitable for high-speed networks.

Bonus 2: Zeek

Zeek, formerly known as Bro, is an open-source network security monitoring (NSM) framework. Unlike traditional intrusion detection systems (IDS) that focus on signature-based detection (matching known patterns of malicious activity), Zeek is known as an anomaly-based IDS because it primarily relies on the detection of anomalous behavior in network traffic.

The term "anomaly-based IDS" refers to a detection approach that focuses on identifying deviations from normal behavior rather than relying on known attack signatures. In the context of Zeek, the framework is considered anomaly-based for several reasons:

1. Behavioral Analysis:

Zeek observes and analyzes the behavior of network traffic, looking for patterns that deviate from the expected or normal baseline. This includes identifying unusual communication patterns, sudden spikes in traffic, or deviations from established network behavior.

2. Customizable Policies:

Zeek allows users to define custom policies using its scripting language. This flexibility enables the creation of rules that are specific to the organization's network environment, making it adaptable to unique threats.

3. Context-Aware Detection:

By maintaining detailed connection tracking and extracting metadata, Zeek is able to provide context around network activities. This context is crucial for distinguishing between normal and anomalous behavior.

4. Dynamic Detection:

Anomaly-based detection is well-suited for identifying emerging threats and zero-day attacks, as it does not rely on pre-existing signatures. Instead, it adapts to changes in network behavior.