CYBR371: System and Network Security
2024– T1

Arman Khouzani, Mohammad Nekooei

Lab 2 (4%): Sniffing, Spoofing, and ARP Poisoning

Submission Deadline: ***23:59:00 (NZST) on Sunday, 31 March 2024***

EXAMPLE SOLUTIONS + GRADING RUBRIC

Instructions: Skim through this document quickly (to get an idea what the questions are about). Carry out the following labs on netlab through nuku (under modules):

- Lab: Investigating ARP poisoning
- Lab: Capturing Network Traffic
- Lab: Packet Crafting with Scapy

Answer the questions in the order they appear in this document. Submit your answer document as a single pdf using the ecs submission link (https://apps.ecs.vuw.ac.nz/submit/CYBR371) for Lab2.

1 ARP Poisoning & Network Traffic Capturing

Complete the following labs on netlab (access via Nuku/Modules):

- Lab: Investigating ARP poisoning
- Lab: Capturing Network Traffic

Q 1.1 [18 points] Explain the utility of the following settings/commands used in these labs. Provide clear, concise answers with one example scenario highlighting their significance (60 words max per each answer.) [3 points each]

(a) Promiscuous mode

Solution: This mode allows a network adaptor to pass all received traffic to the upper layers of the TCP/IP protocol, irrespective of the adaptor the traffic is addressed to. Example ref: support.citrix.com/article/CTX219263

(b) IP forward field

Solution: IP forwarding is a feature of the Linux Kernel. It enables the operating system to accept incoming network packets and pass it on to another network after recognising that it is not meant for itself. This field should be enabled when we want a system to act as a router; this allows the system to forward the IP packets through the designated interface(s).

Example reference: unix.stackexchange.com/questions/14056/what-is-kernel-ip-forwarding — Answer by LawrenceC

(c) **`arpwatch`**

Solution: Arpwatch keeps track for ethernet/ip address pairings. It syslogs arp activity and reports certain changes via email. This can help in identifying ARP poisoning or spoofing attacks, where an attacker attempts to deceive the network by associating their MAC address with the IP address of another host (e.g. the gateway router).
Example references: en.wikipedia.org/wiki/Arpwatch, linux.die.net/man/8/arpwatch.

(d) **`urlsnarf`**

Solution: **`urlsnarf`** is a tool that captures all URL links on sniffed HTTP traffic and is used as a web analysis tool. It logs all HTTP GET and POST requests in Common Log Format. (Example reference: linux.die.net/man/8/urlsnarf)

(e) **`tcpdump`**

Solution: **`tcpdump`** is a network sniffing utility that can capture raw network traffic, similar to **`wireshark`** but command line based.

(f) **`netstat`**

Solution: a network utility on Linux systems providing details on the network status, connections, sockets (IP+Port numbers) and state of the connections (established, `SYN_RCVD` etc), routing tables and provides statistics.

2 Packet Crafting with Scapy

Go to Nuku of the course and under Modules, complete the following lab:

- Lab: Packet Crafting with Scapy

Then answer the following questions.

Q 2.1 [6 points] Explain the utility of the following settings/commands used in this lab concisely – 100 words max per each entry (3 points each).

(a) TTL

Solution: TTL stands for Time to Live. It is a field in IP protocol which specifies a limit on the number of hops a packet is allowed before being discarded. Each router that receives the packet subtracts 1 from the TTL field. If the count remains greater than 0, the router forwards the packet, otherwise it discards the packet and sends an Internet Control Message Protocol (ICMP) message back to the host, indicating that the packet has been dropped.

Rubric: 3 points.

(b) the / Operator (Provide an example)

Solution: / operator is used in Scapy to stack layers. Example:

```
ip= IP(dst='94.180.216.34')
udp = UDP(dport=9100)
data = 'Hello ! \n'
pkt = ip/udp/data
```

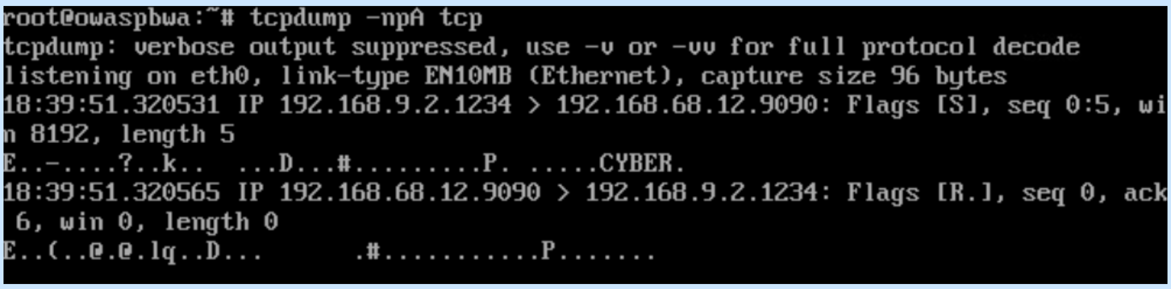
Rubric: 3 points: 2 for explanation, 1 for example.

Q 2.2 [16 points] Write the commands in Scapy for the following. Please make sure your Scapy code matches the requirements in each task. Partially correct answers are not accepted. (4 points each).

(a) Create and send a TCP packet with the payload "CYBER" with source port of 1234 from the Kali host, and to OWASP BWA host on destination port of 9090. Take a screenshot of the **tcpdump** capture on the destination, confirming your packet was delivered.

Solution:

```
send(IP(src="192.168.9.2", dst="192.168.68.12")
    /TCP(sport=1234, dport=9090)
    /Raw(load="CYBR371"))
```



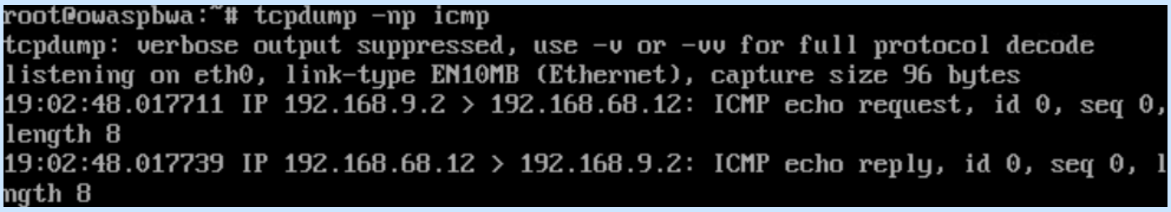
```
root@owaspbwa:~# tcpdump -npA tcp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
18:39:51.320531 IP 192.168.9.2.1234 > 192.168.68.12.9090: Flags [S], seq 0:5, wi
n 8192, length 5
E..-....?..k..  ...D...#.........P. .....CYBER.
18:39:51.320565 IP 192.168.68.12.9090 > 192.168.9.2.1234: Flags [R.], seq 0, ack
 6, win 0, length 0
E..(..@.@.lq..D...      .#..........P.......
```

Rubric: 2 points for code, 2 points for screenshot (any screenshot proving delivery).

(b) Create and send an "ICMP echo" packet from the host (Kali) to the destination OWASP BWA. Take a screenshot of the tcpdump capture on the destination confirming your ICMP packet was delivered.

Solution:

```
send(IP(src="192.168.9.2", dst="192.168.68.12")/ICMP())
```



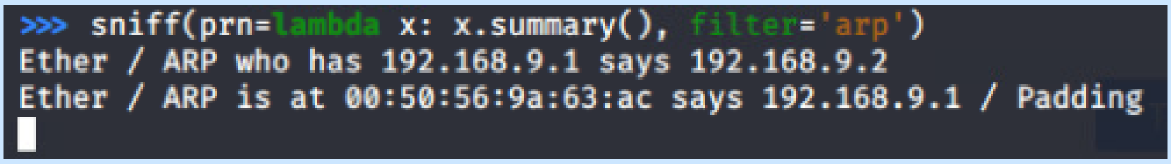
```
root@owaspbwa:~# tcpdump -np icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
19:02:48.017711 IP 192.168.9.2 > 192.168.68.12: ICMP echo request, id 0, seq 0,
length 8
19:02:48.017739 IP 192.168.68.12 > 192.168.9.2: ICMP echo reply, id 0, seq 0, l
ngth 8
```

Rubric: 2 points for code, 2 points for screenshot.

(c) Sniff (i.e. capture) ARP traffic on all the interfaces on the host machine (i.e. Kali). Provide a snippet of the screenshot of its output showing it runs as expected.

Solution: You can either enter the list of interfaces as an array in the iface field, or not specify any interface, because the default is `None` which is interpreted as all interfaces. We need to set the filter field to `arp`, we should also use `prn` to tell `sniff` what to do with the captured packets, here, display a summary of them:

```
sniff(prn=lambda x:x.summary(), filter='arp')
```



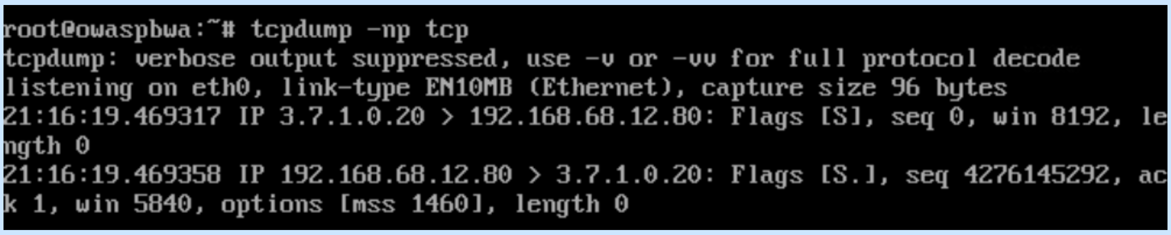
```
>>> sniff(prn=lambda x: x.summary(), filter='arp')
Ether / ARP who has 192.168.9.1 says 192.168.9.2
Ether / ARP is at 00:50:56:9a:63:ac says 192.168.9.1 / Padding
```

Rubric: 2 points for code, 2 points for screenshot. The screenshot should show some arp traffic.

(d) Create and send a spoofed TCP/IP packet (with forged source IP address) from the host (Kali) to the destination machine, OWASP BWA with forged source IP address of **`3.1.7.0`**. Take a screenshot of the **`tcpdump`** capture on the destination proving your message was delivered. Briefly explain why the message was not dropped in transmission even though the source IP address does not exist.

Solution:

```
send(IP(src="3.1.7.0", dst="192.168.68.12")/TCP())
```



```
root@owaspbwa:~# tcpdump -np tcp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
21:16:19.469317 IP 3.7.1.0.20 > 192.168.68.12.80: Flags [S], seq 0, win 8192, le
ngth 0
21:16:19.469358 IP 192.168.68.12.80 > 3.7.1.0.20: Flags [S.], seq 4276145292, ac
k 1, win 5840, options [mss 1460], length 0
```

Because network transmissions are by default destination based, they check the destination address and not the source address.

Rubric: 1 point for the code, 1 point for the screenshot, 2 points for the explanation.