**Team –**

**a. Suraj Bhatia**
**b. Soumya Mohanty**

1. **The recorded MAC addresses.**

```
team@nslabu:~$ ifconfig tap0
tap0      Link encap:Ethernet  HWaddr 96:17:9e:e7:c5:4e
          inet addr:10.0.0.17  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::9417:9eff:fee7:c54e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5340 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5965 errors:0 dropped:1538 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:377675 (377.6 KB)  TX bytes:395363 (395.3 KB)
```

```
team@nslabu:~$ arp -a
chicago.nslab (10.0.0.124) at 00:50:56:9f:5a:33 [ether] on tap0
? (192.168.254.2) at 52:54:00:12:35:02 [ether] on eth0
knoxville.nslab (10.0.0.113) at 08:00:27:7b:11:4c [ether] on tap0
paiute.nslab (10.0.0.254) at 00:50:56:9f:3d:d9 [ether] on tap0
team@nslabu:~$
```

```
team@nslabu:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
chicago.nslab            ether   00:50:56:9f:5a:33   C                     tap0
192.168.254.2            ether   52:54:00:12:35:02   C                     eth0
knoxville.nslab          ether   08:00:27:7b:11:4c   C                     tap0
paiute.nslab             ether   00:50:56:9f:3d:d9   C                     tap0
team@nslabu:~$
```

2. **A snippet of the ARP data right after the ARP poisoning, showing at least one correct ARP reply and a few spoofed ARP replies.**

CORRECT ARP REPLIES

```
18:11:00.139674 ARP, Reply 10.0.0.124 is-at 00:50:56:9f:5a:33, length 28
18:11:00.479969 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
18:11:10.490577 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
18:11:20.501262 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
```

```
20:18:53.200690 ARP, Reply 10.0.0.113 is-at 08:00:27:7b:11:4c, length 46
20:18:53.200837 ARP, Reply 10.0.0.124 is-at 00:50:56:9f:5a:33, length 28
20:19:00.144944 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
20:19:10.155367 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
20:19:20.165884 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
```

ARP SPOOFED

```
19:53:00.318901 ARP, Request who-has 10.0.0.124 tell 10.0.0.113, length 4
19:53:00.320209 ARP, Reply 10.0.0.124 is-at 96:17:9e:e7:c5:4e, length 28
19:53:08.513886 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
```

```
20:23:00.394090 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
20:23:00.443256 ARP, Request who-has 10.0.0.124 tell 10.0.0.113, length 46
20:23:00.448796 ARP, Reply 10.0.0.124 is-at 96:17:9e:e7:c5:4e, length 28
20:23:10.404458 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
```

3. **A snippet of 10 or so lines of HTTP data between LIN and WIN.**

```
team@nslabu:~$ sudo tcpdump -n -i tap0 port 80 and host 10.0.0.124
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tap0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:08:59.960933 IP 10.0.0.124.46329 > 10.0.0.113.80: Flags [S], seq 900329400, win 5840, options [mss 1336,sackOK,TS val 1260
43 ecr 0,nop,wscale 4], length 0
18:08:59.985131 IP 10.0.0.124.46329 > 10.0.0.113.80: Flags [S], seq 900329400, win 5840, options [mss 1336,sackOK,TS val 1260
43 ecr 0,nop,wscale 4], length 0
18:11:00.077107 IP 10.0.0.124.46330 > 10.0.0.113.80: Flags [S], seq 3933024027, win 5840, options [mss 1336,sackOK,TS val 156
100 ecr 0,nop,wscale 4], length 0
18:11:00.080702 IP 10.0.0.124.46330 > 10.0.0.113.80: Flags [S], seq 3933024027, win 5840, options [mss 1336,sackOK,TS val 156
100 ecr 0,nop,wscale 4], length 0
18:11:00.125267 IP 10.0.0.113.80 > 10.0.0.124.46330: Flags [P.], seq 3859078012:3859078351, ack 3933024325, win 65238, option
s [nop,nop,TS val 4955 ecr 156126], length 339: HTTP: HTTP/1.1 200 OK
18:11:00.139709 IP 10.0.0.113.80 > 10.0.0.124.46330: Flags [P.], seq 0:339, ack 1, win 65238, options [nop,nop,TS val 4955 ec
r 156126], length 339: HTTP: HTTP/1.1 200 OK
18:11:00.150888 IP 10.0.0.113.80 > 10.0.0.124.46330: Flags [.], ack 2, win 65238, options [nop,nop,TS val 4956 ecr 156133], l
ength 0
18:11:00.151199 IP 10.0.0.113.80 > 10.0.0.124.46330: Flags [F.], seq 339, ack 2, win 65238, options [nop,nop,TS val 4956 ecr
156133], length 0
18:11:00.152625 IP 10.0.0.113.80 > 10.0.0.124.46330: Flags [.], ack 2, win 65238, options [nop,nop,TS val 4956 ecr 156133], l
ength 0
18:11:00.152691 IP 10.0.0.113.80 > 10.0.0.124.46330: Flags [F.], seq 339, ack 2, win 65238, options [nop,nop,TS val 4956 ecr
156133], length 0
18:13:00.193237 IP 10.0.0.124.46331 > 10.0.0.113.80: Flags [S], seq 1965479410, win 5840, options [mss 1336,sackOK,TS val 186
135 ecr 0,nop,wscale 4], length 0
18:13:00.201327 IP 10.0.0.124.46331 > 10.0.0.113.80: Flags [S], seq 1965479410, win 5840, options [mss 1336,sackOK,TS val 186
135 ecr 0,nop,wscale 4], length 0
18:15:00.308238 IP 10.0.0.124.46332 > 10.0.0.113.80: Flags [S], seq 1781062101, win 5840, options [mss 1336,sackOK,TS val 216
152 ecr 0,nop,wscale 4], length 0
18:15:00.312763 IP 10.0.0.124.46332 > 10.0.0.113.80: Flags [S], seq 1781062101, win 5840, options [mss 1336,sackOK,TS val 216
152 ecr 0,nop,wscale 4], length 0
18:17:00.427717 IP 10.0.0.124.46333 > 10.0.0.113.80: Flags [S], seq 727103849, win 5840, options [mss 1336,sackOK,TS val 2461
79 ecr 0,nop,wscale 4], length 0
18:17:00.432315 IP 10.0.0.124.46333 > 10.0.0.113.80: Flags [S], seq 727103849, win 5840, options [mss 1336,sackOK,TS val 2461
79 ecr 0,nop,wscale 4], length 0
18:19:00.541450 IP 10.0.0.124.46334 > 10.0.0.113.80: Flags [S], seq 1241604356, win 5840, options [mss 1336,sackOK,TS val 276
208 ecr 0,nop,wscale 4], length 0
18:19:00.544287 IP 10.0.0.124.46334 > 10.0.0.113.80: Flags [S], seq 1241604356, win 5840, options [mss 1336,sackOK,TS val 276
208 ecr 0,nop,wscale 4], length 0
```

4. **A password line from ettercap showing the HTTP username and password.**

```
Wed Oct 11 18:47:00 2017 [411463]
TCP  10.0.0.124:46348 --> 10.0.0.113:80 | AP (172)
GET /secure/secret.txt HTTP/1.0.
User-Agent: Wget/1.12 (linux-gnu).
Accept: */*.
Host: 10.0.0.113.
Connection: Keep-Alive.
Authorization: Basic bXJib3NzOlN0cjBuZ1BhNTU=.
.
HTTP : 10.0.0.113:80 -> USER: mrboss  PASS: Str0ngPa55  INFO: 10.0.0.113/secure/secret.txt
```

5. **A snippet of the ARP data right after the ARP poisoner is deactivated, showing the correct MAC address for the LIN machine.**

```
20:25:05.533344 ARP, Reply 10.0.0.124 is-at 00:50:56:9f:5a:33, length 28
20:25:10.531442 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
20:25:20.541791 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
20:25:30.552145 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
20:25:40.562528 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
20:25:50.572883 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
20:26:00.583270 ARP, Reply 10.0.0.113 is-at 96:17:9e:e7:c5:4e, length 28
20:26:03.862741 ARP, Request who-has 10.0.0.254 tell 10.0.0.124, length 28
```

```
20:32:32.908657 ARP, Request who-has 10.0.0.124 tell 10.0.0.17, length 28
20:32:32.938912 ARP, Reply 10.0.0.124 is-at 00:50:56:9f:5a:33, length 28
20:32:37.951149 ARP, Request who-has 10.0.0.17 tell 10.0.0.124, length 28
```

6. **What is Mallory's probability of success in the DNS cache poisoning attack against Alice?**

   In the given scenario Mallory who is the attacker knows that Alice's resolver will be requesting Bob's DNS server for the IP of example.org.

   Mallory knows the IP address of Alice's resolver and the IP address of Bob's server. Mallory also sends the forged response to Alice's resolver at the right time, that is just before Bob's response reaches Alice.

   Mallory's attack will only be successful if she knows the port number on which Alice's resolver is listening for responses and if she has the correct transaction id (TXID). If this is the case then the probability of success will be 100%.

   Else if Mallory does not have these pieces of information then the forged reply that she sent will be dropped by Alice's resolver as it won't have the correct transaction id (TXID) or it may be addressed to the wrong port. In that case Mallory will fail to poison the DNS cache and her probability of success is 0%.

7. **In the DNS cache poisoning section, you studied a specific flaw in some DNS software which could allow an attacker to easily poison it. State which flaw you studied and describe the *specific* scenario(s) in which an attacker could exploit it.**

   DNS requests mostly have two parts: a transaction id (TXID) and an IP with a port number paired together. Cache poisoning used to involve guessing the TXID and UDP combinations, but this was pretty difficult to guess in a limited time frame.

   But most DNS servers do not correctly validate DNS responses to ensure that they are from an authoritative source. Attackers can also send a series of bad requests of a certain type, which greatly increase the chances of the DNS server caching the IP address of a malicious website.

   First the attacker would start by trying to get the DNS server to connect to a maliciously controlled domain entry. If the attacker can make the DNS server look up an address from another DNS server that the attacker controls, he/she can obtain the port number that the targeted DNS server is using.

   Now the attacker would bombard the server with many requests for a non-existing subdomain at the domain for which the attacker wants to hijack the lookups.

   For example, the attacker may try to force the DNS server to lookup aaaaa.neu.edu and then as the attacker knows the UDP port of the target DNS server, he/she will send thousands of fake responses for that subdomain. If the neu.edu DNS server replies with the right TXID before the attacker then the attack is unsuccessful. But as the attacker is sending thousands of fake responses the chances of his response with the right TXID getting cached in the target DNS server is more. This leads to DNS cache poisoning.

   Even if aaaaa.neu.edu fails, the attacker can switch to aaaab.neu.edu. Eventually the attacker wins. The fake response will also contain other information about the same domain.

   This is known as additional record fields. The attacker will use that to replace the cached entry for the main domain or any other domain he/she wishes to change.

8. **Suppose it was your task to design a simple heuristic to detect ARP poisoning attacks. What kinds of abnormalities could a passive sniffer look for that would be strongly indicative of this kind of MitM attack?**

   A passive sniffer could be used to monitor the ARP traffic on a computer network and store a log of pairings of IP and MAC addresses. These logs could be periodically checked to see if any two MAC addresses have the same IP address. If this were the case then a Man In The Middle (MITM) attack is underway. This would also help the network administrator know who the attacker is.

9. **Submit a link to a tool that you can install on your Linux machine to detect ARP poisoning**.

   https://linux.die.net/man/8/arpwatch

10. **In the tcpdump output of HTTP packets, why are only packets from LIN to WIN shown?**

    One of the parameters used in the tcpdump command is "host" where it filters packets based on the "host" IP address. The host can be the source or the destination.

    Our host here is 10.0.0.124 i.e. is the Linux Host machine.

    When we run the Ettercap command, all traffic between the Linux and Windows machines is routed through our NAT router. Hence, only traffic between these two hosts is filtered and can be viewed in the tcpdump output.