Comp 434/534 - Spring 2023
Project #4
Due date - 23:59 19/05/2023

---

1 Overview

The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as the MAC address, given an IP address. The ARP protocol is a very simple protocol, and it does not implement any security measure. The ARP cache poisoning attack is a common attack against the ARP protocol. Using such an attack, attackers can fool the victim into accepting forged IP-to-MAC mappings. This can cause the victim's packets to be redirected to the computer with the forged MAC address, leading to potential man-in-the-middle attacks.

The objective of this lab is for students to gain the first-hand experience on the ARP cache poisoning attack, and learn what damages can be caused by such an attack. In particular, students will use the ARP attack to launch a man-in-the-middle attack, where the attacker can intercept and modify the packets between the two victims A and B. Another objective of this lab is for students to practice packet sniffing and spoofing skills, as these are essential skills in network security, and they are the building blocks for many network attack and defense tools. Students will use Scapy to conduct lab tasks. This lab covers the following topics:

- The ARP protocol
- The ARP cache poisoning attack
- Man-in-the-middle attack
- Scapy programming

Videos. Detailed coverage of the ARP protocol and attacks can be found in the following:

- Section 3 of the SEED Lecture at Udemy, *Internet Security: A Hands-on Approach*, by Wenliang Du. See details at `https://www.handsonsecurity.net/video.html`.

Lab environment. This lab has been tested on the SEED Ubuntu 20.04 VM. You can download a pre-built image from the SEED website, and run the SEED VM on your own computer. However, most of the SEED labs can be conducted on the cloud, and you can follow our instruction to create a SEED VM on the cloud.

2 Environment Setup using Container

In this lab, we need three machines. We use containers to set up the lab environment, which is depicted in Figure 1. In this setup, we have an attacker machine (Host M), which is used to launch attacks against the other two machines, Host A and Host B. These three machines must be on the same LAN, because the ARP cache poisoning attack is limited to LAN. We use containers to set up the lab environment.

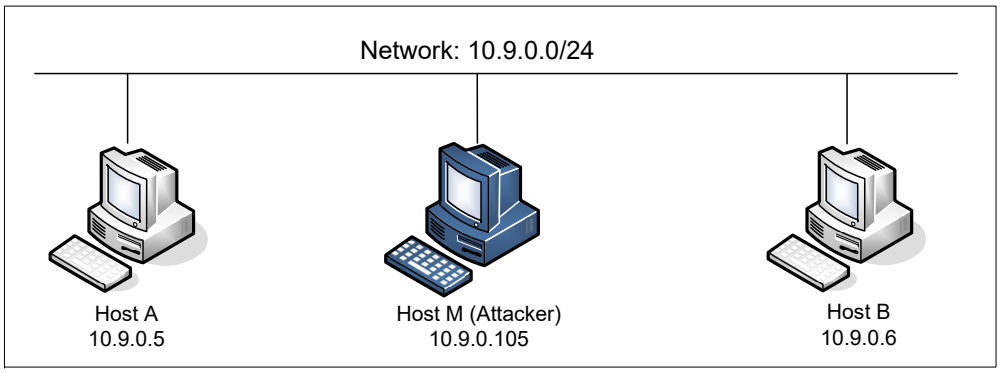

Figure 1: Lab environment setup

2.1 Container Setup and Commands

Please download the `Labsetup.zip` file to your VM from the lab's website, unzip it, enter the `Labsetup` folder, and use the `docker-compose.yml` file to set up the lab environment. Detailed explanation of the content in this file and all the involved `Dockerfile` can be found from the user manual, which is linked to the website of this lab. If this is the first time you set up a SEED lab environment using containers, it is very important that you read the user manual.

In the following, we list some of the commonly used commands related to Docker and Compose. Since we are going to use these commands very frequently, we have created aliases for them in the `.bashrc` file (in our provided SEEDUbuntu 20.04 VM).

```
$ docker-compose build  # Build the container images
$ docker-compose up     # Start the containers
$ docker-compose down   # Shut down the containers

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the `"docker ps"` command to find out the ID of the container, and then use `"docker exec"` to start a shell on that container. We have created aliases for them in the `.bashrc` file.

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#
```

```
// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

2.2 About the Attacker Container

In this lab, we can either use the VM or the attacker container as the attacker machine. If you look at the Docker Compose file, you will see that the attacker container is configured differently from the other containers. Here are the differences:

- *Shared folder.* When we use the attacker container to launch attacks, we need to put the attacking code inside the container. Code editing is more convenient inside the VM than in containers, because we can use our favorite editors. In order for the VM and container to share files, we have created a shared folder between the VM and the container using the Docker `volumes`. If you look at the Docker Compose file, you will find out that we have added the following entry to some of the containers. It indicates mounting the `./volumes` folder on the host machine (i.e., the VM) to the `/volumes` folder inside the container. We will write our code in the `./volumes` folder (on the VM), so they can be used inside the containers.

  ```
  volumes:
        - ./volumes:/volumes
  ```

- *Privileged mode.* To be able to modify kernel parameters at runtime (using `sysctl`), such as enabling IP forwarding, a container needs to be privileged. This is achieved by including the following entry in the Docker Compose file for the container.

  ```
  privileged: true
  ```

2.3 Packet Sniffing

Being able to sniff packets is very important in this lab, because if things do not go as expected, being able to look at where packets go can help us identify the problems. There are several different ways to do packet sniffing:

- Running `tcpdump` on containers. We have already installed `tcpdump` on each container. To sniff the packets going through a particular interface, we just need to find out the interface name, and then do the following (assume that the interface name is `eth0`):

  ```
  # tcpdump -i eth0 -n
  ```

  It should be noted that inside containers, due to the isolation created by Docker, when we run `tcpdump` inside a container, we can only sniff the packets going in and out of this container. We won't be able to sniff the packets between other containers. However, if a container uses the `host` mode in its network setup, it can sniff other containers' packets.

- Running `tcpdump` on the VM. If we run `tcpdump` on the VM, we do not have the restriction on the containers, and we can sniff all the packets going among containers. The interface name for a network is different on the VM than on the container. On containers, each interface name usually starts with

`eth`; on the VM, the interface name for the network created by Docker starts with `br-`, followed by the ID of the network. You can always use the `ip address` command to get the interface name on the VM and containers.

- We can also run Wireshark on the VM to sniff packets. Similar to `tcpdump`, we need to select what interface we want Wireshark to sniff on.

3 Task 1: ARP Cache Poisoning

The objective of this task is to use packet spoofing to launch an ARP cache poisoning attack on a target, such that when two victim machines A and B try to communicate with each other, their packets will be intercepted by the attacker, who can make changes to the packets, and can thus become the man in the middle between A and B. This is called Man-In-The-Middle (MITM) attack. In this task, we focus on the ARP cache poisoning part. The following code skeleton shows how to construct an ARP packet using Scapy.

```
#!/usr/bin/env python3
from scapy.all import *

E = Ether()
A = ARP()
A.op = 1     # 1 for ARP request; 2 for ARP reply

pkt = E/A
sendp(pkt)
```

The above program constructs and sends an ARP packet. Please set necessary attribute names/values to define your own ARP packet. We can use `ls(ARP)` and `ls(Ether)` to see the attribute names of the `ARP` and `Ether` classes. If a field is not set, a default value will be used (see the third column of the output):

```
$ python3
>>> from scapy.all import *

>>> ls(Ether)
dst        : DestMACField                    = (None)
src        : SourceMACField                  = (None)
type       : XShortEnumField                 = (36864)

>>> ls(ARP)
hwtype     : XShortField                     = (1)
ptype      : XShortEnumField                 = (2048)
hwlen      : ByteField                       = (6)
plen       : ByteField                       = (4)
op         : ShortEnumField                  = (1)
hwsrc      : ARPSourceMACField               = (None)
psrc       : SourceIPField                   = (None)
hwdst      : MACField                        = ('00:00:00:00:00:00')
pdst       : IPField                         = ('0.0.0.0')
```

In this task, we have three machines (containers), A, B, and M. We use M as the attacker machine. We would like to cause A to add a fake entry to its ARP cache, such that B's IP address is mapped to M's MAC address. We can check a computer's ARP cache using the following command. If you want to look at the ARP cache associated with a specific interface, you can use the `-i` option.

```
$ arp -n
Address     HWtype  HWaddress           Flags Mask  Iface
10.0.2.1    ether   52:54:00:12:35:00   C           enp0s3
10.0.2.3    ether   08:00:27:48:f4:0b   C           enp0s3
```

There are many ways to conduct ARP cache poisoning attack. Students need to try the following three methods, and report whether each method works or not. Include your scripts and explanations in the report. In order to correct the ARP cache between attacks, students can use the ping command.

- **Task 1.A (using ARP request).** On host M, construct an ARP request packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not.
- **Task 1.B (using ARP reply).** On host M, construct an ARP reply packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not. Try the attack under the following two scenarios, and report the results of your attack:
  - Scenario 1: B's IP is already in A's cache.
  - Scenario 2: B's IP is not in A's cache. You can use the command "`arp -d a.b.c.d`" to remove the ARP cache entry for the IP address `a.b.c.d`.
- **Task 1.C (using ARP gratuitous message).** On host M, construct an ARP gratuitous packet, and use it to map B's IP address to M's MAC address. Please launch the attack under the same two scenarios as those described in Task 1.B.

  ARP gratuitous packet is a special ARP request packet. It is used when a host machine needs to update outdated information on all the other machine's ARP cache. The gratuitous ARP packet has the following characteristics:
  - The source and destination IP addresses are the same, and they are the IP address of the host issuing the gratuitous ARP.
  - The destination MAC addresses in both ARP header and Ethernet header are the broadcast MAC address (`ff:ff:ff:ff:ff:ff`).
  - No reply is expected.

4 Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Hosts A and B are communicating using Telnet, and Host M wants to intercept their communication, so it can make changes to the data sent between A and B. The setup is depicted in Figure 2. We have already created an account called "`seed`" inside the container, the password is "`dees`". You can telnet into this account.

Step 1 (Launch the ARP cache poisoning attack). First, Host M conducts an ARP cache poisoning attack on both A and B, such that in A's ARP cache, B's IP address maps to M's MAC address, and in B's ARP cache, A's IP address also maps to M's MAC address. After this step, packets sent between A and B will all be sent to M. We will use the ARP cache poisoning attack from Task 1 to achieve this goal. It is better that you send out the spoofed packets constantly (e.g. every 5 seconds); otherwise, the fake entries may be replaced by the real ones.

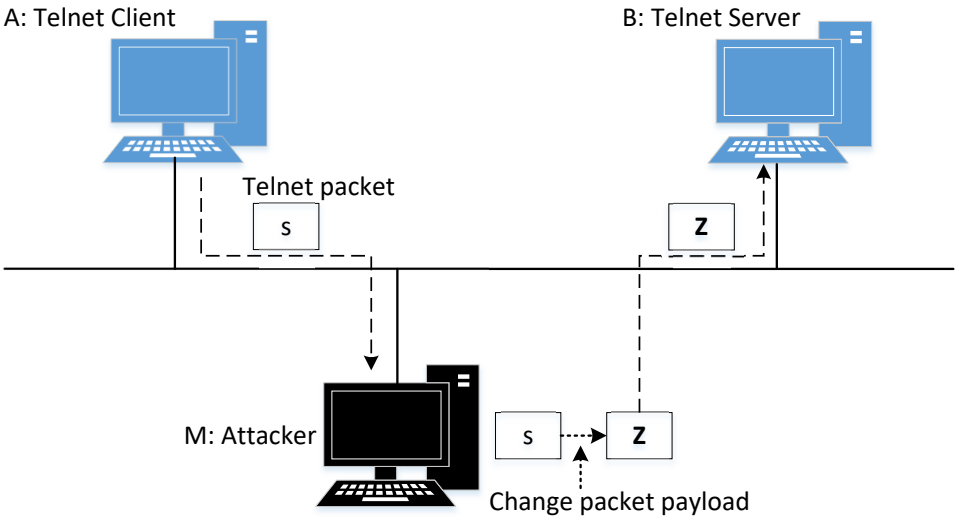

Figure 2: Man-In-The-Middle Attack against telnet

Step 2 (Testing). After the attack is successful, please try to ping each other between Hosts A and B, and report your observation. Please show Wireshark results in your report. Before doing this step, please make sure that the IP forwarding on Host M is turned off. You can do that with the following command:

```
# sysctl net.ipv4.ip_forward=0
```

Step 3 (Turn on IP forwarding). Now we turn on the IP forwarding on Host M, so it will forward the packets between A and B. Please run the following command and repeat Step 2. Please describe your observation using Wireshark results.

```
# sysctl net.ipv4.ip_forward=1
```

Step 4 (Launch the MITM attack). We are ready to make changes to the Telnet data between A and B. Assume that A is the Telnet client and B is the Telnet server. After A has connected to the Telnet server on B, for every key stroke typed in A's Telnet window, a TCP packet is generated and sent to B. We would like to intercept the TCP packet, and replace each typed character with a fixed character (say Z). This way, it does not matter what the user types on A, Telnet will always display Z.

From the previous steps, we are able to redirect the TCP packets to Host M, but instead of forwarding them, we would like to replace them with a spoofed packet. Write a sniff-and-spoof program to accomplish this goal. Include the program code and necessary explanations in your report. In particular, we would like to do the following:

- We first keep the IP forwarding on, so we can successfully create a Telnet connection between A to B. Once the connection is established, we turn off the IP forwarding using the following command. Please type something on A's Telnet window, and report your observation:

  ```
  # sysctl net.ipv4.ip_forward=0
  ```

- We run our sniff-and-spoof program on Host M, such that for the captured packets sent from A to B, we spoof a packet but with TCP different data. For packets from B to A (Telnet response), we do not make any change, so the spoofed packet is exactly the same as the original one. After running your program, try typing letters through A's Telnet window, report your observations.

To help students get started, we provide a skeleton sniff-and-spoof program below. The program captures all the TCP packets, and then for packets from A to B, it makes some changes (the modification part is not included, because that is part of the task). For packets from B to A, the program does not make any change.

```
#!/usr/bin/env python3
from scapy.all import *

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"


def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
         # Create a new packet based on the captured one.
         # 1) We need to delete the checksum in the IP & TCP headers,
         #    because our modification will make them invalid.
         #    Scapy will recalculate them if these fields are missing.
         # 2) We also delete the original TCP payload.

         newpkt = IP(bytes(pkt[IP]))
         del(newpkt.chksum)
         del(newpkt[TCP].payload)
         del(newpkt[TCP].chksum)

         ################################################################
         # Construct the new payload based on the old payload.
         # Students need to implement this part.

         if pkt[TCP].payload:
             data = pkt[TCP].payload.load  # The original payload data
             newdata = data   # No change is made in this sample code

             send(newpkt/newdata)
         else:
             send(newpkt)
         ################################################################

    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
         # Create new packet based on the captured one
         # Do not make any change

         newpkt = IP(bytes(pkt[IP]))
         del(newpkt.chksum)
         del(newpkt[TCP].chksum)
         send(newpkt)
```

```
f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

It should be noted that the code above captures all the TCP packets, including the one generated by the program itself. That is undesirable, as it will affect the performance. Students need to change the filter, so it does not capture its own packets.

Behavior of Telnet. In Telnet, typically, every character we type in the Telnet window triggers an individual TCP packet, but if you type very fast, some characters may be sent together in the same packet. That is why in a typical Telnet packet from client to server, the payload only contains one character. The character sent to the server will be echoed back by the server, and the client will then display the character in its window. Therefore, what we see in the client window is not the direct result of the typing; whatever we type in the client window takes a round trip before it is displayed. If the network is disconnected, whatever we typed on the client window will not displayed, until the network is recovered. Similarly, if attackers change the character to Z during the round trip, Z will be displayed at the Telnet client window, even though that is not what you have typed.

5 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. At the beginning of your report, you are required to include the honor pledge. You can find it in the course syllabus. If you have any question about the assignment, send an email to efehanguner21@ku.edu.tr.

6 Acknowledgement

This project is adapted from ”ARP Cache Poisoning Attack Lab” from SEED Labs.