

ARP Cache Poisoning Lab

CMSC 426 - Computer Security

April 13, 2021

1 Lab Overview

The Address Resolution Protocol (ARP) is used to discover a link-layer address such as a MAC address given an IP address. The protocol does not implement any security measures. In particular, as discussed in class, it is fairly simple to poison the ARP cache of a host. By poisoning the ARP caches on two hosts, an attacker may force communication between the victims to pass through the attacker's machine, allowing the attacker to monitor or modify the communications.

In this lab, you will use a virtual lab environment to poison the ARP caches of two victims, ultimately launching a successful man-in-the-middle attack against the victims. In the process of carrying out the attack, you will gain experience with packet spoofing using the Python Scapy library and basic packet sniffing using `tcpdump`.

2 Lab Tasks

2.1 Initial Setup

To complete the lab, you will need to run the `SEEDUbuntu20.04` virtual machine in Virtual Box or a cloud provider (e.g. Amazon AWS or Microsoft Azure). The VM and set-up instructions are available from the SEED website (<https://seedsecuritylabs.org/labsetup.html>). Docker virtualization software is pre-installed on the `SEEDUbuntu20.04` VM, and you will be running multiple Docker containers within the VM.

Note: Throughout the lab, “VM” will refer to the `SEEDUbuntu20.04` VM and “container” will refer to a Docker container.

To carry out the ARP cache poisoning attack you will use three containers connected to a virtual local-area network, as depicted in Figure 1. The attacker machine, Host M, will be used to launch attacks against Host A and Host B.

Once you have the VM running, download the `Labsetup.zip` (https://seedsecuritylabs.org/Labs_20.04/Files/ARP_Attack/Labsetup.zip), unzip it, and `cd` to the `Labsetup` folder.

The lab containers and virtual network are created using Docker Compose, a tool for defining multi-container Docker applications. The configuration is specified in a YAML file, `docker-compose.yml`. It is not necessary that you understand all the details of the YAML file; it is sufficient to understand that it creates three containers (A, B, and M) and connects them through a virtual LAN. However, if you would like to learn more, see the *SEED Manual for Containers* (<https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/SEEDManual-Container.md>).

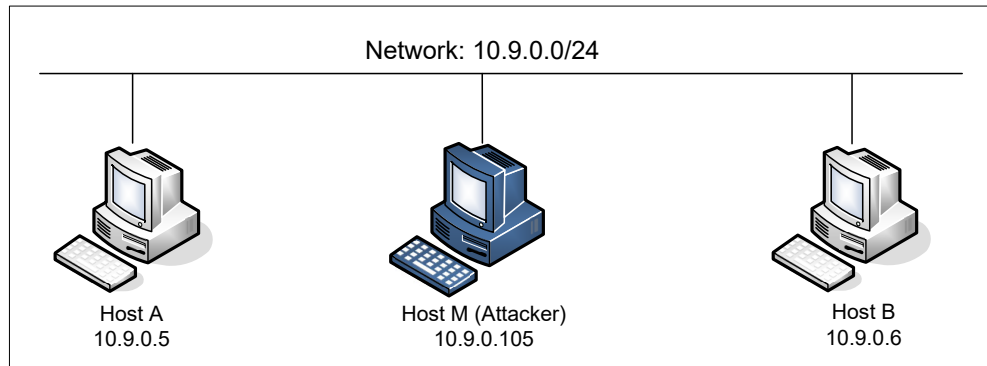


Figure 1: Lab Environment

To build and start the lab containers, you will use the `docker-compose build` and `docker-compose up` commands. Be sure you are in the `Labsetup` directory and run the commands:

```
$ docker-compose build
$ docker-compose up
```

You should see something like the following

```
cmarron — seed@SEEDUbuntu20: ~/Labsetup — ssh -i .ssh/SEED-VM_key.pem seed@20.84.66.200 — 80x14
seed@SEEDUbuntu20:~/Labsetup$ docker-compose build
HostA uses an image, skipping
HostB uses an image, skipping
HostM uses an image, skipping
seed@SEEDUbuntu20:~/Labsetup$ docker-compose up
B-10.9.0.6 is up-to-date
A-10.9.0.5 is up-to-date
M-10.9.0.105 is up-to-date
Attaching to B-10.9.0.6, A-10.9.0.5, M-10.9.0.105
A-10.9.0.5 | * Starting internet superserver inetd          [ OK ]
B-10.9.0.6 | * Starting internet superserver inetd          [ OK ]
```

Figure 2: Docker Startup

At this point, in order to interact with the containers, you will need to open another terminal window to the VM. To stop the containers, change to the `Labsetup` directory on a second terminal and enter the command:

```
$ docker-compose down
```

It may take a few seconds, but eventually the containers will be shut down, as shown in Figure 3.

Since these commands will be needed frequently, they are summarized here, along with aliases defined in the `.bashrc` file in the VM.

```
$ docker-compose build # Build the container image
```

A terminal window titled 'cmarron — seed@SEEDUbuntu20: ~/Labsetup — ssh -i .ssh/SEED-VM_key.pem seed@20.84.66.200 — 80x14'. The prompt is 'seed@SEEDUbuntu20:~/Labsetup\$'. The command 'docker-compose down' has been executed, resulting in the following output: 'Stopping B-10.9.0.6 ... done', 'Stopping M-10.9.0.105 ... done', 'Stopping A-10.9.0.5 ... done', 'Removing B-10.9.0.6 ... done', 'Removing M-10.9.0.105 ... done', 'Removing A-10.9.0.5 ... done', and 'Removing network net-10.9.0.0'. The prompt is now 'seed@SEEDUbuntu20:~/Labsetup\$'.

Figure 3: Docker Shutdown

```
$ docker-compose up      # Start the container
$ docker-compose down    # Shut down the container

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

To open a shell on one of the virtualized host containers, you will first need to find the container ID using the "docker ps" command. Then a shell may be opened using "docker exec". Useful aliases for these commands are provided:

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

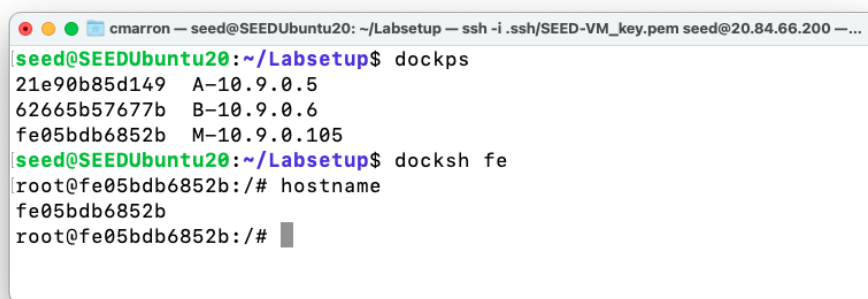
Figure 4 demonstrates accessing a shell on Host M. Note that if a docker command requires a container ID, you do not need to type the entire ID string; the first few characters will be sufficient, as long as they are unique among all the running containers.

If you encounter problems when setting up the lab environment, please see the “Common Problems” section of the SEED Docker Manual (<https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/SEEDManual-Container.md>).

2.2 Attacker Container

Although it is possible to use the VM to launch the attack, it is preferable that you use Host M as the attacker machine. Host M has been specially configured to allow it to function as the attacker:

- *Shared Folder:* To use Host M to launch the attacks, you will need to move the attack code to the container’s file system. In order for the SEEDUbuntu20.04 VM and Host M container to share files, a shared folder has been created using Docker volumes. Within the Labsetup folder, you will find a folder named `volumes`; this same folder is shared to Host M, where it is mounted as `/volumes`. Sharing files between the two filesystems is as simple as copying them to the shared folder.



```
cmarron — seed@SEEDUbuntu20: ~/Labsetup — ssh -i .ssh/SEED-VM_key.pem seed@20.84.66.200 —...
seed@SEEDUbuntu20:~/Labsetup$ dockps
21e90b85d149  A-10.9.0.5
62665b57677b  B-10.9.0.6
fe05bdb6852b  M-10.9.0.105
seed@SEEDUbuntu20:~/Labsetup$ docksh fe
root@fe05bdb6852b:/# hostname
fe05bdb6852b
root@fe05bdb6852b:/#
```

Figure 4: Shell on a Docker Container

- *Privileged Mode.* Host M is configured to run in privileged mode so that kernel parameters may be modified at runtime (using `sysctl`). This will be needed to enable and disable IP forwarding.

2.3 Packet Sniffing

It is important that you be able to “sniff packets” for this lab, especially if your attack does not appear to work: packet sniffing will allow you to see precisely what packets are being sent, how they are addressed, and what responses are generated.

- The simplest method for packet sniffing is to run `tcpdump` on the VM. If you run `tcpdump` on the VM, you can sniff all the packets sent among the containers. However, 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. As an example:

```
# tcpdump -i br-dabc5193956c -n
```

If you are unsure of the interface name, you can use the `ifconfig` command to list the names of all network interfaces on the VM.

- Another option is to run `tcpdump` on one of the containers. However, due to the way containers are isolated by Docker, only packets to or from the container on which `tcpdump` is run will be visible: you won’t be able to sniff packets sent between other containers. Suppose we want to sniff the `eth0` interface; this would be done with the following command:

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

If you are not sure of the interface names, use `ifconfig` to view the available interfaces (but it should usually be `eth0`).

- If you are running the Linux GUI on the VM, either locally in VirtualBox, or via VNC from a cloud-based VM, sniffing can also be done using Wireshark. However, many students will prefer the command-line `tcpdump` for performance reasons.

3 Task 1: ARP Cache Poisoning

The objective of this task is to use packet spoofing to launch an ARP cache poisoning attack against two target machines so that the victims' communications can be monitored or modified by the attacker, resulting in a complete "Man-in-the-Middle" attack.

You will use the Python Scapy library to send spoofed packets. The following code skeleton demonstrates how to construct and send an ARP packet using Scapy. You will need to replace the interface name `br-05f0c56e8085` with the correct interface name for your setup.

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

E = Ether()
A = ARP()

pkt = E/A
sendp(pkt, iface='br-05f0c56e8085')
```

In order to poison the cache of Hosts A and B, you will need to set certain attributes of the ARP packet. Part of your challenge is to determine what these attributes should be set to; this may require some research. You can use `ls(ARP)` in Python to see the attribute names of the ARP class. If you do not explicitly set an attribute, a default value will be used (specified in the third column of the output):

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

First, you must poison Hosts A's ARP cache so that Host B's IP address is mapped to Host M's MAC address. You can check the contents of a container's ARP cache using the `arp -n` command. If you want to view the ARP cache entries associated with a specific, 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
```

You must attempt each of the following three approaches to poisoning the ARP cache:

- *Task 1.A: Using ARP Request.* On host M, construct an ARP request packet and send it to host A. Check A's ARP cache, and see whether M's MAC address is mapped to B's IP address.
- *Task 1.B: Using ARP Reply.* On host M, construct an ARP reply packet and send to host A. Check A's ARP cache, and see whether M's MAC address is mapped to B's IP address. Try the attack for two different scenarios:

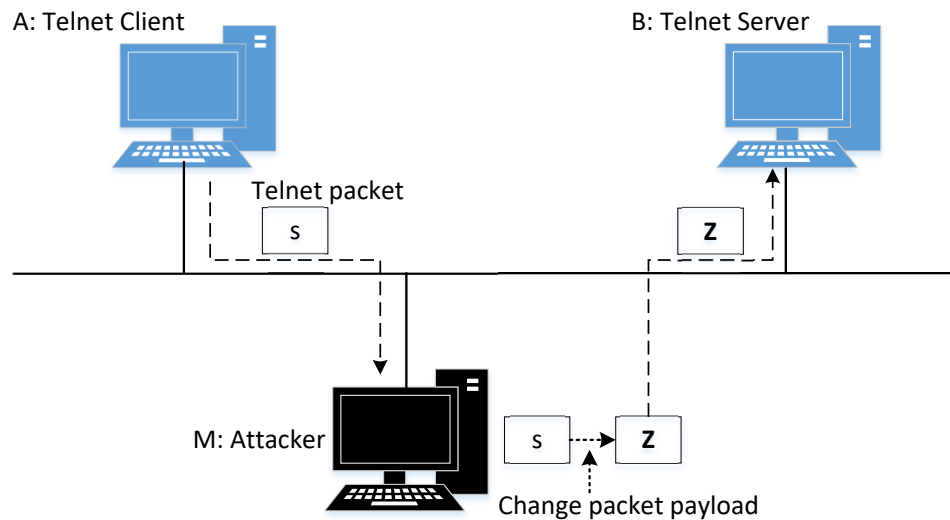


Figure 5: Man-In-The-Middle Attack against telnet

- Scenario 1: B's IP is already in A's cache.
- Scenario 2: B's IP is not in A's cache.
- *Task 1C: Using ARP Gratuitous Message.* On host M, construct an ARP gratuitous packet, and use it to map M's MAC address to B's IP address. Launch the attack under the same two scenarios as those described in Task 1.B.

An ARP gratuitous packet is a special ARP request packet. It is used when a host machine needs to update outdated information in all other machine's ARP caches. 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 the ARP header and the Ethernet header are the broadcast MAC address (`ff:ff:ff:ff:ff:ff`).
- No reply is expected.

4 Task 2: Man-in-the-Middle Attack on Telnet

Hosts A and B will communicate using Telnet; Host M will be used to intercept and modify their communications. The setup is depicted in Figure 5. A username and password are needed to initiate a Telnet session, so the user `seed` with password `dees` has been pre-configured on the host containers.

Step 1: Launch the ARP cache poisoning attack). First, Host M conducts an ARP cache poisoning attack on both A and B so 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. Use the ARP cache poisoning attack from Task 1 to achieve this goal. Spoofed packets should be sent constantly (e.g. every 5 seconds) to ensure that the fake cache entries are not replaced with valid entries.

Step 2: Testing. First, turn off IP forwarding on Host M:

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

Now, attempt to ping Host B from Host A, and vicer versa. Describe your observations and include a screenshot of `tcpdump` or Wireshark output.

Step 3: Turn on IP forwarding. Now turn on the IP forwarding on Host M so that it will forward the packets between A and B:

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

Now repeat step 2. Describe your observations and include a `tcpdump` or Wireshark screenshot.

Step 4: Launch the MITM attack. You are ready to modify 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 will be generated and sent to B. You must intercept the TCP packets, and replace each typed character with a fixed character, say "Z". Then, no matter what the user types on A, Telnet will display "Z".

In the previous steps, you were able to redirect the TCP packets to Host M. Now, rather than forward the packets, you must replace them with spoofed packets. You will write a sniff-and-spoof program to accomplish this goal. A skeleton sniff-and-spoof program is provided below to help you get started on that portion of the lab. The completed program must do the following:

- Start with IP forwarding on so that a Telnet connection can be created between A and B. Once the connection is established, turn off IP forwarding using the command provided previously. Type something on A's Telnet window, and report your observation.
- Run your sniff-and-spoof program on Host M so that the captured packets sent from A to B are replaced with new, spoofed packets. Do not modify packets from B to A (Telnet response).

The skeletong sniff-and-spoof program captures all the TCP packets and modifies packets from A to B (the modification portion is not provided; that is part of your 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)

```

Note that the skeleton code captures all TCP packets, including packets generated by the program itself. You must modify the “filter” portion of the program so that it does not capture its own packets.

Behavior of Telnet. In Telnet, typically, every character that is typed is sent in a separate TCP packet, but if you type very quickly, some characters may be sent together in a single packet. The character sent to the server will be echoed back to the client, 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. Therefore, modifying the typed character to “Z” when it is sent from A to B will result in “Z” being displayed on the Telnet client window, even though that is not what was typed.

5 Task 3: MITM Attack on Netcat

This task is similar to Task 2, except that Hosts A and B will communicate using `netcat`. Host M will intercept and modify data sent between A and B. You can use the following commands to establish a `netcat` TCP connection between A and B:

```

On Host B (server, IP address is 10.9.0.6), run the following:
# nc -lp 9090

```

```

On Host A (client), run the following:

```



```
# nc 10.9.0.6 9090
```

Once the connection is made, you can type messages on A. Each line will be sent to B in a TCP packet, and B will display the line of text. Your task is to replace every occurrence of your first name in the message with a sequence of “A”s. The length of the sequence should be the same as that of your first name, otherwise the TCP sequence number may be incorrect. If the lab is completed in a group, choose the first name of one member of the the group to use for the demonstration. Include screenshots and any observations in your lab report.

6 Submission

You must submit a detailed lab report, with ample screenshots, describing what you have done and what you have observed. Provide an explanation of any observations that are interesting or surprising. Include your completed sniff-and-spoof program, any important code snippets, and explanations of your code. Submitting code without explanation is not sufficient.

Copyright © 2019 by Wenliang Du.

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. If you remix, transform, or build upon the material, this copyright notice must be left intact, or reproduced in a way that is reasonable to the medium in which the work is being re-published.