ARP Cache Poisoning Attack Lab

Contents

Lab Setup

Please download the Labsetup.zip file from the below link to your VM, unzip it, enter the Labsetup folder, and use the docker-compose.yml file to set up the lab environment.

https://seedsecuritylabs.org/Labs_20.04/Files/ARP_Attack/Labsetup.zip

In this lab, we need to have at least three machines. We use containers to set up the lab environment.

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.

Students can also use three virtual machines for this lab, but it will be much more convenient to use containers.

Note: When we use the attacker container to launch attacks, we need to put the attacking code inside the attacker container. Code editing is more convenient inside the VM than in containers, because we can use our favorite editors. Hence it is advisable for you to place your respective codes in the "volumes" folder directly (using gedit for example).

Lab 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 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 defence 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

Attacker (Host M) - 10.9.0.105
Host A - 10.9.0.5
Host B - 10.9.0.6

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 the 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/python3
from scapy.all import *
E = Ether()
A = ARP()
pkt = E/A
pkt.show()
sendp(pkt)
```

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.

There are many ways to conduct the ARP cache poisoning attack. Students need to try the following three methods and report whether each method works or not.

Points & tips to be Noted:

- All the **python** code in this lab needs to be run on the Attacker Machine.
- To view the ARP cache table, you need to use the command **arp**
- 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**

Task 1.A: Using ARP request

On host M, construct an ARP request packet and send it to host A. Check whether M's MAC address is mapped to B's IP address in A's ARP cache

- <u>Without Ether</u>

- View the arp table and then run tcpdump on the Hosts before executing the below code.

To view the arp table run the following on both Host's A and B

Command:

On Host A and B

```
# arp
```

Then run the following on both A and B to sniff packets going through each container -

Command:

On Host A and B

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

Finally, execute the below command on the attacker machine M -

Command:

On Attacker M

```
# python3 task1A.py
```

Show your observations by providing screenshots of your terminal - Packets Captured using tcpdump and the ARP cache on Host A and Host B, **before and after** running the attack.

Close the tcpdump and run the following to view the updated arp cache, take a screenshot of the same -

Command:

On Host A and B

```
# arp
```

Take a screenshot of the attacker terminal after the attack as well.

Now delete the ARP Cache entries of the attacker and Host B by executing the below commands on Host A. Provide a screenshot for the same -

Command:

On Host A

```
# arp -d 10.9.0.6
# arp -d 10.9.0.105
```

- With Ether

- Perform the same steps as mentioned previously, but this time we provide parameters for Ether

- View the arp table and then run tcpdump before executing the below code.

To view the arp table run the following on both Hosts A and B

Command:

On Host A and B

```
# arp
```

Then run the following to sniff packets going through each container -

Command:

On Host A and B

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

Now execute the below command on the attacker machine M -

Command:

On Attacker M

```
# python3 task11A.py
```

Show your observations by providing screenshots of your terminal - Packets Captured using tcpdump and the ARP cache on Host A and Host B, **before and after** running the attack. Also, show a screenshot of the attacker's terminal after the attack.

Delete ARP cache entries of Attacker and Host B and provide a screenshot.

Command:

```
# arp -d 10.9.0.6
# arp -d 10.9.0.105
```

Questions:

1. What does the 'op' in the screenshot of the attacker machine signify? What is its default value?
2. What was the difference between the ARP cache results in the above 2 approaches? Why did you observe this difference?

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.

<u>For Scenario 1</u>

In order to place B's IP in A's cache -Execute the following on the Attacker Machine M -

Command:

On Attacker M

```
# python3 task11A.py
```

Take a screenshot of the arp cache. Then we run tcpdump on Host A to sniff packets -

Command:

On Host A

# tcpdump -i eth0 -n

Finally, we run the following on the Attacker M

Command:

On Attacker M

# python3 task1B.py

Show your observations by providing screenshots of your terminal - Packets Captured using tcpdump and the ARP cache on Host A, **before and after** running the attack.
Also, show a screenshot of the attacker's terminal after the attack.

For Scenario 2

Delete the ARP Cache entry of Host B in A

Command:

On Host A

# arp -d 10.9.0.6

# arp -d 10.9.0.105

Check the arp table to confirm the same.

Now run tcpdump to sniff the packets on Host A

Command:

On Host A

# tcpdump -i eth0 -n

Now run the following on the Attacker Machine M

Command:

On Attacker M

```
# python3 task1B.py
```

Question:

1. What does op=2 mean?

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 the ARP header and Ethernet header are the broadcast MAC address (ff:ff:ff:ff:ff:ff).
- No reply is expected.

For Scenario 1

Execute the following on the Attacker Machine M -

Command:

On Attacker M

```
# python3 task1A.py
```

Take a screenshot of the **arp cache**. Then we run tcpdump on **Host A and Host B** to sniff packets

Command:

On Host A and Host B

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

Finally, we run the following on the Attacker M

Command:

Attacker M

```
# python3 task1C.py
```

Show your observations by providing screenshots of all the terminals (A,B and M) - Packets Captured using tcpdump and the ARP cache on Host A and B, **before and after** running the attack.

Also, show a screenshot of the attacker's terminal after the attack.

For Scenario 2

Now we delete the ARP Cache entries on both Host A and Host B

Command:

On Host A and B

```
# arp -d 10.9.0.6
# arp -d 10.9.0.105
```

Check the arp table to confirm the same.

Now run tcpdump to sniff the packets on Host A and Host B

Command:

On Host A and B

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

Now run the following on the Attacker Machine M

Command:

On Attacker M

```
# python3 task1C.py
```

Show your observations by providing screenshots of all terminals - Packets Captured using tcpdump and the ARP cache on Host A and B, **before and after** running the attack.
Also, show a screenshot of the attacker's terminal after the attack.

Questions:

1. Why does VM B's ARP cache remain unchanged in this approach even though the packet was broadcasted on the network?

Make sure to delete your ARP cache entries of Host A, and Host B before proceeding to the next Task.

Command:

On Host A and B

```
# arp -d 10.9.0.6
# arp -d 10.9.0.105
```

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 1. We have already created an account called "seed" inside the container, the password is "dees". You can telnet into this account.

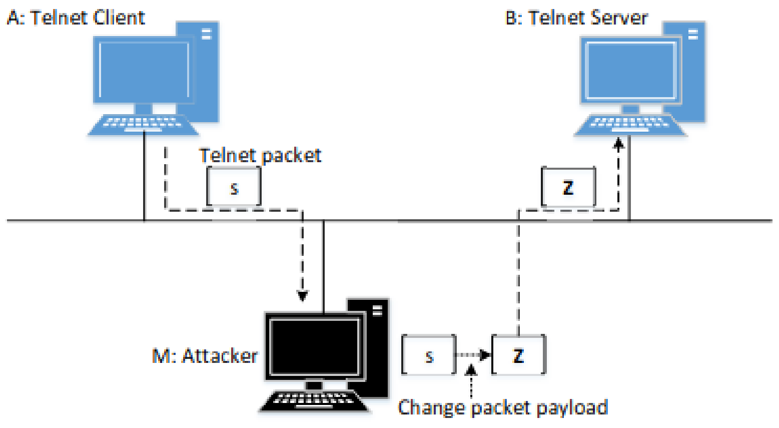

Figure 1

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.**

First, check the ARP caches of Host A and Host B

Command:

On Host A and B

# arp

Now for this step, we execute the code and commands as discussed in Task 1A (with Ether) mapping B's IP address to M's MAC address in A's ARP Cache.

Command:

```
# python3 taskl1A.py
```

Then execute the below code to map A's IP address to M's MAC address in B's ARP Cache

Command:

```
# python3 task2.py
```

Finally check the updated ARP caches of Host A and Host B

Command:

On Host A and B

```
#arp
```

Show your observations by providing screenshots of your terminal -ARP cache on Host A and B **before and after** running the attack. Also, show a screenshot of the attacker terminal M after the attack.

Please Note: From now on, at times you won't be getting the desired output, this is due to the fact that the ARP caches are being made redundant on Both Host Machines (A and B), so you will have to execute task11A.py and task2.py in order to update the ARP entries.

Step 2 - Testing

You will need Wireshark from now - open the container interface ‘br-’ in order to capture the required packets.

On Attacker M, disable IP forwarding by executing the following

Command:

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

Update the ARP Caches

Command:

On Attacker M

```
# python3 task11A.py
# python3 task2.py
```

Then we ping from Host A to Host B using the following command:

Command:

On Host A

```
# ping 10.9.0.6
```

Please provide screenshots of Host A and the Attacker, with the packets captured on Wireshark.

Question:

1. What do you observe? Explain

In case the desired output (ping does not work) does not occur, then you will have to update the ARP Cache by executing task11A.py and task2.py on Attacker M.

Step 3 - Turn on IP Forwarding

Now we turn on the IP forwarding on Host M so that it will forward the packets between A and B.

Command -

On Attacker M

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

Now ping Host B from Host A -

Command:

On Host A

```
# ping 10.9.0.6
```

Please provide screenshots (Terminals and Wireshark) and describe your observation

Question

1. Compare the results between the above two steps.

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 keystroke 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. We will write a sniff-and-spoof program to accomplish this goal. In particular, we would like to do the following:

Reminder - Make sure to execute Step 1 to update the ARP tables and open Wireshark on the given interface.

On Host M

Command :

```
# python3 task11A.py
# python3 task2.py
```

We first keep the IP forwarding on, so we can successfully **create a Telnet connection between A to B.**

On Host M

Command :

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

To establish a Telnet connection between Host A and B

On Host A

Command:

# telnet 10.9.0.6

Once the connection is established, we turn off the IP forwarding

Back On Host M

Command:

# sysctl net.ipv4.ip_forward=0

Please type something on Host A's Telnet window, and **see the packets captured on Wireshark, take a screenshot of the same.**

Now to perform the <u>Man in the Middle Attack</u>, we start over and repeat the above steps - for establishing the Telnet connection. (Wireshark Required)

On Host M

- **Command :**

  # python3 task11A.py

  # python3 task2.py

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.

On Host M

Command :

# sysctl net.ipv4.ip_forward=1

On Host A

Command:

# telnet 10.9.0.6

Back On Host M

Command:

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

Now on Host M, we run the following to accomplish our Attack

Command:

```
# python3 task11A.py
# python3 task2.py
# python3 mitm.py
```

Now type anything on the Telnet Window (Host A), only 'Z' should be displayed.

Show your observations of your terminals with the manipulated data, the telnet connection, Wireshark capture and the attacker terminal.

The 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 be 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

Task 3: MITM Attack on Netcat using ARP Cache Poisoning

This task is similar to Task 2, except that Hosts A and B are communicating using netcat, instead of telnet. Host M wants to intercept their communication, so it can make changes to the data sent between A and B. Once the connection is made, you can type messages on A. Each line of messages will be put into a TCP packet sent to B, which simply displays the message. Your task is to replace every occurrence of your first name in the message with a sequence of A's.

Commands:

The sequence of commands to be run:

On Attacker M -

```
# python3 task11A.py
# python3 task2.py
# sysctl net.ipv4.ip_forward=1
```

You can use the following commands to establish a netcat TCP connection between A and B

On Host B -

```
# nc -lp 9090
```

On Host A -

```
# nc 10.9.0.6 9090
```

To launch the Attack

On Attacker M -

```
# python3 task11A.py
# python3 task2.py
# sysctl net.ipv4.ip_forward=0
# python3 mitm1.py
```

Type a 6-character sequence (maximum) or word on Host A's netcat connection, preferably the first 6 characters of your name.

The length of the sequence should be 6, or you will mess up the TCP sequence number, and hence the entire TCP connection.

The output on Host B should be a sequence of 6 'A's' confirming our attack has worked.

Show screenshots of all the Hosts and explain your observations.

Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to explain the observations that are interesting or surprising. Please also list the important code snippets followed by an explanation. Simply attaching code without any explanation will not receive credits.