**ARP Cache Poisoning Attack Lab Report**

This report details the steps I followed to complete the ARP Cache Poisoning Attack Lab as outlined in the provided documentation. The lab aimed to give hands-on experience with ARP cache poisoning attacks, leading to man-in-the-middle (MITM) scenarios where communication between two victim machines could be intercepted and modified by an attacker. I also explored various packet sniffing techniques and executed MITM attacks on both **Telnet** and **Netcat** communications.

**1. Environment Setup**

I started by setting up the lab environment using **Docker** containers as virtual machines. The setup consisted of three machines on a local area network (LAN):

* **Host A (Victim A)** with IP 10.9.0.5
* **Host B (Victim B)** with IP 10.9.0.6
* **Host M (Attacker)** with IP 10.9.0.105

These machines communicated within the LAN network 10.9.0.0/24, and the attacker aimed to intercept and manipulate traffic between Host A and Host B.

**Commands Used:**

1. Downloaded the necessary setup files (Labsetup.zip), unzipped, and used **Docker Compose** to configure the environment.

$ docker-compose build # Build the container images

$ docker-compose up # Start the containers

$ docker-compose down # Stop the containers

1. Accessed the individual containers (Hosts A, B, and M) via a shell using:

$ dockps # List container IDs and names

$ docksh <container\_id> # Open shell within a container

**2. Task 1: ARP Cache Poisoning**

The objective of this task was to inject a fake entry into the ARP cache of Host A so that it would associate Host B's IP address with Host M’s MAC address. This redirection allowed the attacker to intercept communication between Hosts A and B.

**Steps and Commands:**

1. **Viewing the ARP Cache**: To view Host A’s ARP cache, I used the following command:

bash

Copy code

$ arp -n

1. **ARP Request Attack (Task 1.A)**: I constructed an ARP request to map Host B’s IP (10.9.0.6) to Host M’s MAC address, sending this request to Host A.
   * Python script using **Scapy**:

python

Copy code

from scapy.all import \*

E = Ether()

A = ARP(op=1, hwsrc="attacker\_mac", psrc="victimB\_ip", hwdst="victimA\_mac", pdst="victimA\_ip")

sendp(E/A)

1. **ARP Reply Attack (Task 1.B)**: I created an ARP reply packet and sent it to Host A under two scenarios:
   * **Scenario 1**: B’s IP is already in A’s cache.
   * **Scenario 2**: B’s IP is not in A’s cache (removed using arp -d).
2. **ARP Gratuitous Message (Task 1.C)**: I launched the attack by sending an ARP gratuitous message, which informed Host A that B’s IP address should be associated with M’s MAC address.
   * Characteristics of the gratuitous ARP:
     + Source and destination IP are both B’s IP.
     + Destination MAC is the broadcast address (ff:ff:ff:ff:ff:ff).
     + No reply is expected.

**Results:**

* After each attack method, I verified the success by checking Host A’s ARP cache. If successful, the cache showed B’s IP associated with M’s MAC, meaning Host A’s traffic was being redirected to the attacker.

**3. Task 2: MITM Attack on Telnet using ARP Cache Poisoning**

This task involved executing a MITM attack by using ARP cache poisoning to intercept Telnet communication between Hosts A and B. The attacker, Host M, altered the data exchanged between the two hosts.

**Steps and Commands:**

1. **Enabling IP Forwarding**: I first conducted the ARP cache poisoning attack and then enabled IP forwarding on Host M to forward traffic between Hosts A and B:

bash

Copy code

$ sysctl net.ipv4.ip\_forward=1

1. **Packet Sniffing**: I monitored network traffic using **tcpdump** and **Wireshark**:
   * **tcpdump** on Host M’s interface:

bash

Copy code

$ tcpdump -i eth0 -n

* + **Wireshark** on the VM to capture traffic across all containers.

1. **MITM Attack Implementation**: After successfully intercepting traffic, I manipulated Telnet communication by writing a **Scapy** script to modify packets from Host A before forwarding them to Host B.
   * Python script:

python

Copy code

def spoof\_pkt(pkt):

if pkt[IP].src == "A\_ip" and pkt[IP].dst == "B\_ip":

new\_pkt = IP(pkt[IP])/TCP(pkt[TCP])

new\_pkt[TCP].payload = "Z" \* len(pkt[TCP].payload) # Replace with 'Z'

send(new\_pkt)

**Results:**

* Host A’s keystrokes were replaced with 'Z' characters in Telnet communication, demonstrating successful packet interception and modification.

**4. Task 3: MITM Attack on Netcat using ARP Cache Poisoning**

Similar to Task 2, this task required intercepting Netcat communication between Hosts A and B using ARP cache poisoning, then altering specific parts of the transmitted data.

**Steps and Commands:**

1. **Netcat Connection**:
   * On Host B (server):

bash

Copy code

$ nc -lp 9090

* + On Host A (client):

bash

Copy code

$ nc 10.9.0.6 9090

1. **Packet Modification**: I modified the intercepted Netcat packets, replacing every occurrence of my first name with a string of 'A's of equal length. This was crucial to avoid disrupting the TCP sequence numbers and causing connection issues.
   * Python script using **Scapy**:

python

Copy code

def modify\_data(pkt):

if "myname" in pkt[TCP].payload.decode():

pkt[TCP].payload = pkt[TCP].payload.replace("myname", "AAAAA")

send(pkt)

**Results:**

* The altered Netcat packets successfully replaced occurrences of my first name with 'A's, maintaining the connection's stability.

**5. Packet Sniffing Techniques**

Throughout the tasks, I used packet sniffing to verify traffic behavior and troubleshoot any issues. The following methods were applied:

1. **tcpdump**: Used for capturing packets on container interfaces. I identified packet flow issues and verified the success of ARP cache poisoning attacks.
   * Example command:

bash

Copy code

$ tcpdump -i eth0 -n

1. **Wireshark**: Run on the VM to capture and analyze packets across the entire virtual network. It helped visualize the packet alterations and identify any errors.

**6. Submission**

I followed the submission guidelines to prepare a detailed report. Key components of my submission included:

* **Code Snippets**: Relevant Python scripts using **Scapy** for ARP spoofing, packet interception, and modification.
* **Screenshots**: I included screenshots from **Wireshark** and **tcpdump** demonstrating the captured traffic and successful MITM attacks.
* **Explanations**: Each task was described in detail, with observations of network behavior and outcomes of the ARP cache poisoning and MITM attacks.

**Conclusion**

This lab provided a comprehensive understanding of the vulnerabilities in the ARP protocol and how ARP cache poisoning can be leveraged to execute MITM attacks. By successfully completing ARP poisoning and intercepting traffic in Telnet and Netcat, I gained valuable insights into packet sniffing, spoofing, and network security.