Sniffing and Spoofing Lab

Table of Contents:

Lab Environment Setup

Please download the Labsetup.zip file from the link given below :
https://seedsecuritylabs.org/Labs_20.04/Networking/Sniffing_Spoofing/

Follow the instructions in the **lab setup document** to set up the lab environment.

In this lab, we will use three machines that are connected to the same LAN. We can either use three VMs or three containers. Figure 1 depicts the lab environment setup using containers. We will do all the attacks on the attacker container, while using the other containers as the user machines.

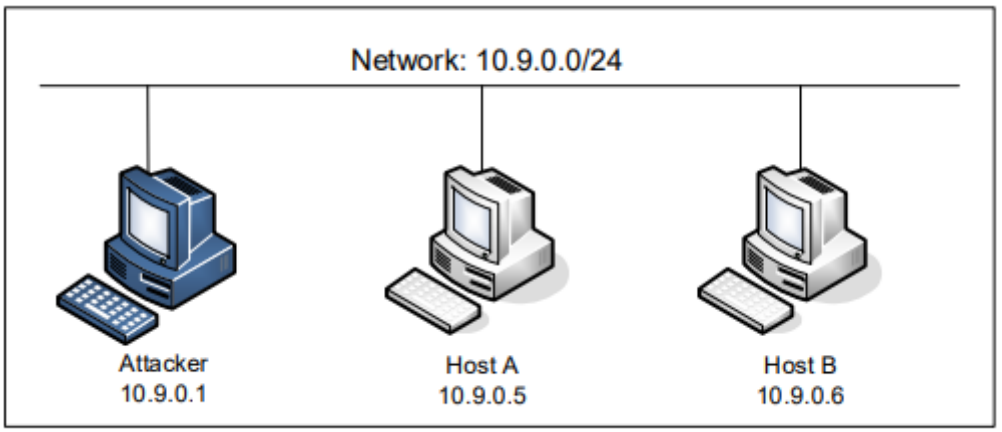

Figure 1 : Lab environment setup

Lab Task Set-1: Using Tools to Sniff and Spoof Packets using Scapy

Task 1.1 : Sniffing Packets

The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs.

Task 1.1 A : Sniff IP packets using Scapy

The program ,for each captured packet, the callback function print pkt() will be invoked; this function will print out some of the information about the packet. Run the program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations.

NOTE

Check the **Lab setup instructions document** for detailed instructions on how to find out the interface of your attacker machine. **Replace the interface in the code wherever required**.

On the Attacker terminal run the command:

 # python3 Task1.1A.py

Explain on which VM you ran this command and why? Provide a screenshot of your observations.

From the **host A** machine's terminal ping a random IP address(8.8.8.8)

On the Host A terminal run the command:

 # ping 8.8.8.8

Now, we run the same program without root privileges. Do you find any issues? If so, why?

Provide a screenshot of your observations.

On the Attacker terminal run the command:

 # su seed

$ python3 Task1.1A.py

Task 1.1 B : Capturing ICMP, TCP packet and Subnet

Usually, when we sniff packets, we are only interested in certain types of packets. We can do that by setting filters in sniffing. Scapy's filter uses the BPF (Berkeley Packet Filter) syntax; you can find the BPF manual from the Internet. Please set the following filters and demonstrate your sniffer program again (each filter should be set separately):

- Capture only ICMP packets.
- Capture any TCP packet that comes from a particular IP and with a destination port number 23.
- Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

Capture only the ICMP packet

The ICMP packets are captured by the Scapy sniffer program. Hence, when some machine on the same network sends ping requests, the packets get captured by the sniffer.

Fill in the interface of the attacker machine in the given program and run the code.

On the Attacker terminal run the command:

 # python3 Task1.1B-ICMP.py

Provide a screenshot of your observations

From the **host A** machine's terminal ping a random IP address(8.8.8.8)

On the Host A terminal run the command:

 # ping 8.8.8.8

The ICMP packets are captured by the sniffer program. Provide a screenshot of your observations.

Capture any TCP packet that comes from a particular IP and with a destination port number 23

The program must capture the TCP packets being sent from the specified IP address on the port 23.

Fill in the interface of the attacker machine in the given program and run the code.

On the Attacker terminal run the command:

 # python3 Task1.1B-TCP.py

Provide a screenshot of your observations

From the **host A** machine's terminal telnet to a random IP address.

On the Host A terminal run the command::

 # telnet 10.9.0.1

Provide screenshots of your observations.

Capture packets that come from or go to a particular subnet

You can pick any subnet, such as 192.168.254.0/24; you should not pick the subnet that your VM is attached to.

Show that on sending ICMP packets to 192.168.254.1, the sniffer program captures the packets sent out from 192.168.254.1 .

Fill in the interface of the attacker machine in the given program and run the code.

On the Attacker terminal run the command:

 # python3 Task1.1B-Subnet.py

Provide a screenshot of your observations

From the **host A** machine's terminal, ping a random IP address on the chosen subnet.

On the Host A terminal run the command::

 # ping 192.168.254.1

Provide screenshots of your observations.

Task 1.2 : Spoofing

The objective of this task is to spoof IP packets with an arbitrary source IP address. We will spoof **ICMP echo request packets** and send them to another VM on the same network. We will use Wireshark to observe whether our request will be accepted by the receiver. If it is accepted, an echo reply packet will be sent to the spoofed IP address. Below shows the code to create the ICMP packet. The spoofed request is formed by creating our own packet with the header specifications.

Similarly, we fill the IP header with source IP address of any machine within the local network and destination IP address of any remote machine on the internet which is alive.

Please keep wireshire open before you execute the program.Show that Wireshark captures the live machine sending back an ICMP response.

On the Attacker terminal run the command:

 # python Task1.2A.py

Provide a screenshot of your observations.

Demonstrate that you can spoof an ICMP echo request packet with an **arbitrary source IP address**. Open Wireshark and observe the ICMP packets as they are being captured.

On the Attacker terminal run the command:

 # python Task1.2B.py

Provide a screenshot of your observations.

Task 1.3 : Traceroute

The objective of this task is to implement a simple traceroute tool using Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination.

The below code is a simple traceroute implementation using Scapy. It takes hostname or IP address as the input. We create an IP packet with destination address and TTL value and ICMP packet. We send the packet using function sr1(). This function waits for the reply from the destination. If the ICMP reply type is 0, we receive an echo response from the destination, else we increase the TTL value and resend the packet.

Provide a screenshot of the Wireshark capture that shows the ICMP requests sent with increasing TTL and the error response from the routers with a message as “Time to live exceeded”.

On the Attacker terminal run the command:

 # python3 Task1.3.py 157.240.23.35

157.240.23.35 is the IP address for facebook.com

On running the above python code, provide a screenshot of the response.

Task 1.4 : Sniffing and-then Spoofing

In this task, the victim machine pings a non-existing IP address “1.2.3.4”. As the attacker machine is on the same network, it sniffs the request packet, creates a new echo reply packet with IP and ICMP header and sends it to the victim machine. Hence, the user will always receive an echo reply from a non-existing IP address indicating that the machine is alive.

The below code sniffs ICMP packets sent out by the victim machine. Using the callback function, we can use the packets to send the spoofed packets. We retrieve source IP and destination IP from the sniffed packet and create a new IP packet. The new source IP of the spoofed packet is the sniffed packet's destination IP address and vice versa. We also generate ICMP packets with id and sequence number. In the new packet, ICMP type should be 0 (ICMP reply). To avoid truncated packets, we also add the data to the new packet.

Keep wireshark open before running the python program. Provide wireshark screenshots of the spoofed packets being sent.

Fill in the interface of the attacker machine in the given program and run the code.

On the Attacker terminal run the command:

 # python3 Task1.4.py

Provide a screenshot of your observations.

From the **host A** machine's terminal ping 1.2.3.4

On the Host A terminal run the command::

 # ping 1.2.3.4

Provide a screenshot of 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 provide explanations 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.