Luke Evans

January 16th, 2021

**Internet Security – Lab 1**

*Sniffing and Spoofing*

**Task 1: Using tools to sniff and spoof packets**

*1.1 Sniffing Packets:*

1.1.A:

The python code for sniffing icmp packets is shown in the screenshot below:

Graphical user interface, text, chat or text message

Description automatically generated

We can run the sniffer program with root privileges and ping another ip from a different terminal tab to capture the icmp packet:

A screenshot of a computer

Description automatically generated with low confidence

Text, letter

Description automatically generated

Text, letter

Description automatically generated

When we attempt to run the program without root privileges, we get the following error:

Text, letter

Description automatically generated

Observations: Our simple sniffer.py program shows how we can use scapy to sniff packets. The program works when running with root privileges, however, it will crash when trying to run as a normal user. The reason the program is unable to run without root privileges is it requires the use of a raw socket and the network interface card must be put in promiscuous mode to sniff all icmp packets on the network. This is not allowed unless funning as root which is why we see the permission error.

1.1.B:

The code for capturing only icmp packets is the same code that was used in section 1.1.A:

Graphical user interface, text, chat or text message

Description automatically generated

If we send a ping request to google.com we see the echo-request and echo-reply:

Text, letter

Description automatically generated

Text, letter

Description automatically generated

The code to only sniff tcp packets is shown below. We can use the same code as above but we change the icmp filter to tcp:

Text

Description automatically generated

If we then open firefox we can capture the tcp packets being transmitted:

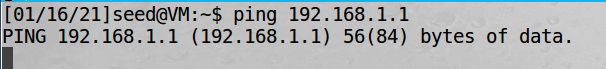
Text

Description automatically generated

If we only want to capture packets from a specific subnet we can update the filter to use the net keyword with the classless IPv4 address. In our example we will use subnet 192.168.0.0/16 and then ping an ip within the specified subnet to capture the packet. The python code is shown below:

Graphical user interface, text, application, chat or text message

Description automatically generated



Text, letter

Description automatically generated

In this case we do not get an echo response because the provided ip address within the specified server is not alive and able to respond.

Observations: The screenshots above show how filters can be applied to sniff packets of a specific type. We first were only interested in ICMP packets, then tcp packets, and finally packets to or from the specified subnet of 192.168.0.0/16.

Explanation: Scapy allows for various filters to easily be applied to our packet sniffer using BPF syntax.

*1.2 Spoofing ICMP Packets:*

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, application

Description automatically generated

Observation: When we run our script from the VM with ip address 10.0.2.15 and set the destination ip to 10.0.2.4 we can observe this ICMP packet on wireshark. We are currently not spoofing the source address. Since we have a.src commented out above the source of the request is auto populated with our real ip address. The code below we will set our ip to 1.1.1.1.

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generated

Observation/Explanation: With a.src set in the code to 1.1.1.1 the source IP is spoofed and so the reply is sent back to that IP. We have successfully spoofed our icmp request. Scapy makes this operation trivial.

*1.3 Traceroute:*

Graphical user interface, text, application, chat or text message

Description automatically generated

The output from the code above is shown below:

Text

Description automatically generated with medium confidence

Observation: It takes 11 hops to get from our VM to [www.google.com](http://www.google.com) at ip address 216.58.194.174. We can see exactly how our request was routed.

Explanation: Scapy can be used to easily view the route a request takes by setting the appropriate parameters and looping through calling the sr1 function. The sr1 function is part of the send and receive packets function (sr) and only returns the packet that answered.