Network Packet Analysis and Scapy Introduction

BLOSSOM
Manchester Metropolitan University
(Funded by Higher Education Academy)
l.han@mmu.ac.uk

1. Learning Objectives

This lab aims to understand network packet and how to use Scapy for network packet analysis

2. Preparation

1) Under Linux environment

2) Some files that you will need from /home/user/BlossomFiles/IntroToScapy:

- 'SecLab1.zip'
- SimplePacketParser.py'
- 'SwitchCase.py'

3) Some documents that you may need to refer to:

- 'Virtual-MachineGuide.pdf'
- 'Linux-Guide.pdf'
- 'BLOSSOM-UserGuide.pdf'

3. Tasks

Setup & Installation:

- Start two virtual machines as you have done with previous exercises (see Virtual Machine Guide):

  # kvm -cdrom /var/tmp/BlossomFiles/blossom-0.98.iso -m 512 -net nic,macaddr=52:54:00:12:34:57 -net vde -name node-one

- Unzip the file 'SecLab1.zip' as it contains all the files required for this lab:

  # unzip SecLab1.zip

Task 1 Network Traffic Analysis:

1.1 Download the Python and Trace files

1.2 Store all of the files in a single folder

1.3 Run each of the Python files one at a time through command prompt using the command:

#python SimplePacketParser.py <trace_filename>

1.4 Follow the simple instructions that should be displayed in order to discover packet header information for Ethernet, IP, ARP, TCP and ICMP headers.

Questions:

What are the sources and destination ports of the TCP Header?
What are the source and target hardware addresses of the ARP Header?
What are the source and destination addresses of the Ethernet Header?
What is the sequence number of the ICMP Header?
What are the source and destination addresses of the IP Header?

Task 2 Scapy Introduction

2.1 Scapy is a packet manipulation tool written in Python. It provides an python interface to libpcap. We can use it to create, send and receive packets. We can also create our own network monitor tool based on Scapy. Please refer to Scapy (http://www.secdev.org/projects/scapy/doc/) for more information.

After scapy has been successfully installed on both of the virtual machines, packets can now be sent between both of them using scapy. Run scapy using the following command:

#scapy

2.2 Scapy can create many different packets and send them between machines on a network. There are two useful commands to remember, which are:

#ls()
#ls(protocol)

The first command lists all of the available protocols, and the second command lists all of the details relevant to a specific protocol. For instance, if you type ls(IP), you will see all of the details that can be edited when creating an IP packet.

2.3 In order to create a basic IP packet and send it to the other machine, an IP packet must be created with the destination address set to that of the other machine, so if the first machines IP address is 10.0.2.16 and the second machines IP address is 10.0.2.17, the following command must be used to create and send the packet:

#send(IP(dst='10.0.2.17'))

This will send an IP packet from machine 1 to machine 2, and if we were watching the network traffic, we would see the packet being transferred.

2.4 In order to create a basic ARP packet and send it to the other machine, the process is very similar to that of an IP packet, except with a few minor differences. The command used is:

#send(ARP(pdst='10.0.2.17'))

The packet name is listed as ARP instead of IP, and pdst is used instead of dst. Each packet in scapy has different variable names, so if you wish to find out about any other packets, it's important to bear in mind the ls(protocol) command mentioned previously.

2.5 In order to create a basic TCP packet and send it to the other machine, the process is again very similar to that of an ARP and IP packet. The command used is:

```
#topt=[('Timestamp', (10,0))]
#send(TCP(dport=22, options=topt))
```

TCP packets make use of destination ports as opposed to IP addresses, so the requirement of a destination address isn't necessary, except for specifying the type of TCP communication that is to be created (ie, port 22 for SSH connections). The timestamp option must be added when sending TCP packets over a virtual network to ensure that the packets are not dropped instantly.

Question: What would the command be to create a UDP packet and send it across a network? This should be very similar to creating and sending a TCP packet.

Task 3 Protocols / Layers – HTTP/FTP

3.1 Scapy has the ability to layer packets, as well as the ability to create sets of packets that mimic specific protocol sessions, such as HTTP and FTP communication sessions. Start off with a standard IP packet with the source and destination addresses set accordingly:

#packet = IP(src='10.0.2.16', dst='10.0.2.17')

3.2 Now that the initial packet has been created, an extra layer can be added on to the packet, such as a TCP layer:

#packet = packet/TCP()

This will add a TCP layer to the IP packet. Each detail of the packet can be edited as well, by using such commands as the following:

#packet.ttl = 10
#packet[TCP].sport = 1025

The first command will change the time to live of the IP packet, whereas the second packet specifies that the source port of the TCP layer will be changed to 1025. This can be done to any detail at any layer of the packet.

Question: What would the list of commands be to create a packet with two layers, IP and TCP, with the IP source address of '192.168.0.1' and the IP destination address of '192.168.0.2', and the TCP source port of 22.

3.3 In order to simulate a HTTP session, a payload must also be layered inside the packet by doing the following:

#HTTPpacket=packet/'GET HTTP/1.1\r\n\r\n'

This will add the payload of a GET HTTP request to the packet, creating a makeshift HTTP packet.

Question: In a similar fashion to the HTTPpacket created previously, can you create an FTP packet?

HINT: Look up FTP packet payloads, and the FTP TCP port number.

Task 4 Scapy - Sniffing

4.1 Scapy can also be used to sniff packets on a network very easily. When scapy is running, use the following command:

#sniff()

4.2 After running the sniff() command on one virtual machine, go to the other virtual machine and send a few packets to across the virtual network using the commands from task 2. After a few packets have been sent, press CTRL+C. A brief summary of the types of packets that have been sniffed should be displayed.

4.3 In order to display the packets that have been sent in more detail, the following two commands must be used:

#a=_
#a.nsummary()

In python, the underscore displays the latest result, so the first command stores the latest result as the variable, a. The second command displays a summary of the packets that have been sniffed in more detail. If even more detail is required for each packet, the following command can be used:

#a[0]

The above command will display the full information for the first packet

4.4 The sniff() command can also have filters applied to it, allowing for the amount of packets that will be sniffed to be significantly reduced. Use the following command:

#sniff(iface='eth0', filter='ip', count=5)

After using the above command, send some IP and ARP packets across the virtual network and then view the summary as discussed in 3.3. The iface option specifies the interface to listen on, for which we chose eth0. The filter option specifies the type of packet to listen for, for which we chose ip, and count specifies how many packets should be listened to before the sniffing stops.

The result of the previous command should result in only 5 IP packets being displayed in the summary, with the ARP packets being filtered out.

Question: What command would you use to sniff 100 ARP packets on the interface eth0?