Scapy

1. Introduction

You will use a virtual machine with Ubuntu as the operating system. The network card will have to be put in bridge mode in order to have the same IP address as your physical machine.

Upgrade your Ubuntu distribution and install the following packages:

- wireshark,

- python3-scapy.

Scapy is a Python library that allows you to listen to network traffic, forge your own packets and send them. In this context, it is possible to define the values contained in the different fields of a protocol from the data link layer to the application layer. Our goal is to use Scapy to realize a function that detects the open ports of network services.

2. Scapy : Mode intercation

To discover Scapy, we will use it in interaction mode. In this mode, which is close to the console mode, you just have to type the commands as you go along. Execute the following command in super user mode

```
root:~# scapy
```

The *lsc()* command lists the available commands. The *ls()* command lists the protocols available with Scapy. We will now forge our first IP packet. The first way is to construct and instantiate the object at the same time (constructor with parameters)

```
>>> packet = IP(src = '127.0.0.1', ttl=1, dst = '127.0.0.1')
>>> packet #     Display what the instance of the IP object contains
<IP  ttl=1 src=127.0.0.1 dst=127.0.0.1 |>
>>> send (packet)
```

The second is to create the object with the empty constructor and then parameterize it

```
>>> p=IP()
>>> p.ttl=1
>>> p.src='127.0.0.1'
>>> p.dst='127.0.0.1'
>>> send (p)
```

You can see by listening to the local interface (lo) with Wireshark that the packet was sent and received from the local interface. It is possible to build the different layers of a packet with the / operator, for example by adding raw data encapsulated within an IP packet with the Raw object (ls(Raw)).

Now send an IP packet containing a message of your choice.

```
>>> packet=IP(src='127.0.0.1', ttl=1, dst='127.0.0.1')/Raw(load='coucou')
>>> send (packet)
```

If you want to send and receive packets you can use the « sr() » function. As you manupilate IP packet, you can also use « sr1() » to send one IP packet and receive its response.

- With the help of the lsc () command, send an ICMP echo-request message and capture the response obtained with Scapy.
- To display the the traffic, use the « show() » function for a developped view of the packet or « summary() » function for a one-line summary
    - Example : ***pkt=sr1(packet)***
      pkt.show()

3. Script mode

The use of Scapy in interactive mode has its limits, especially when you want to complex tasks. It is then necessary to write scripts written in Python. The name of the Python scripts must have the extension .py.

A Python script must always contain two lines. The first one indicates the location of the Python interpreter, the second one indicates the libraries to use, here Scapy.

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

The sniff () function allows to listen to messages on an interface. The script given below allows to capture on the "eth0" interface, the messages of the ARP protocol.

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

def  pkt_callback(pkt) :
     if (pkt.getlayer(ARP).op == 0x01) :                (#op set 1 means a ARP request message)
          print("the MAC source is : " + pkt.getlayer(ARP).hwsrc)

          to complete

sniff(iface="eth0",   prn=pkt_callback(), filter="arp", store=0)
```

For each captured message the pkt_callback() function is triggered. The store parameter indicates that no packet is stored. As the sniff function is blocking you have to quit the program exit the program with the key combination Ctrl+C.

Complete the script to display the source IP address and destination MAC address

To get the response the « op » field must have a code equal to « 0x02 ». complete the script to display the resolved MAC address and its mapped IP Address, the destination MAC and IP addresses

To test, use ping command to communicate with new IP addresses.

4. Scanning ports

In order to determine which application is running on a machine, one technique is to test whether the ports associated with the target applications are open or closed.

Determine if the SSH, HTTP and telnet server applications are installed on your host if not, proceed with the installation. You will have to start and stop the services manually during the rest of this lab.

TCP is a transport protocol that guarantees the routing and scheduling of data. To provide this service, TCP uses a sequence number that is exchanged during the sequence number that is exchanged during the three-step connection phase. With the help of your course and Wireshark, observe the TCP messages received when the target application's port is opened and closed. Based on these observations, we will create a script to test whether a TCP port is open or closed.

Following the template below, create a script that tests and displays whether a TCP port port is open or closed. Be sure to close the server socket if it returns a SYN-ACK.

Attention in Python the indentation determines the scope of a function.

Define end host and TCP port range

host = 'x.x.x.x' #destination host

portRange = [22, 23, 80, 443, 3389]

Send SYN with random Src Port for each Dst port

for dstPort in portRange:

srcPort = random.randint(1025,65534)

#build your own request packet based on ls(IP) and ls(TCP)

resp = sr1(IP(….)/ TCP(……), timeout=1,verbose=0)

IP= #specify the destination host

TCP= #specify : src-port, dst-port, and the flags in this case flags= ‘’S’’ (connection establishement request)

if(resp.getlayer(TCP).flags == ????):

to be completed

elif (resp.getlayer(TCP).flags == ????):

to be completed

if resp is None:

print('{}:{} is filtered (silently dropped).' .format(host, str(dstPort)))

- In a second end host, install the SSH, http and telnet services and test the script