Part 3: Python Network Scanner with Scapy

Contents

Time required: 90 minutes

Python Tabs and Spaces Issue

Visual Studio Code automatically changes a tab into four spaces. Other editors, like geany and nano in Linux, do not. You can end up with a combination of spaces and tabs. Python doesn't like a combination, it wants either one or the other. The preferred method is spaces.

Recommendation:

1. Create your Python files in Visual Studio Code in Windows.
2. Copy and paste the code into either nano or geany in Linux.

Tutorial 1: Send and Receive ARP Request

Time to use our custom ARP program to find others on the network. The first step is to send out our packet to everyone, and combine the responses into 2 lists.

1. Save **network_scanner_2.py** as **network_scanner_3a.py**
2. Make the following modifications.

```python
#!/usr/bin/env python3
"""
    Name: network_scanner_3a.py
    Author:
    Created:
    Purpose: Send an ARP packet print answered and unanswered responses
"""
# Windows
# pip install scapy
# Import the scapy module
import scapy.all as scapy
```

```python
def scan(ip_address_range):
    """
    Create ARP request for targeted IP address range
      Print the answered and unanswered packet information.

    Args:
        ip_address_range (str): The IP address or IP address range to scan.

    Returns:
        None: The function prints the summary
          of the unanswered and answered packet lists.
    """

    # pdst is Target IP address
    arp_request = scapy.ARP(pdst=ip_address_range)

    # Source MAC address is local computer
    # dst sets destination MAC, in this case MAC broadcast address
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")

    # Combine the first two packets together with scapy / operator
    arp_request_broadcast = broadcast/arp_request

    # scapy.srp sends and receives packets with custom layer
    # Returns answered and unanswered packet information in two lists
    answered_list, unanswered_list = scapy.srp(
        arp_request_broadcast,
        timeout=2               # Time to wait for a response
    )

    # Print unanswered and answered lists
    print(unanswered_list.summary())
    print(answered_list.summary())
```

Here is the main function for reference.

```python
def main():
    print("Network Scanner 3")
    # Substitute your local network address
    # Press Enter at the input prompt to use this address
    default_local_network = "192.168.9.0/24"

    # IP range to scan
    # Substitute your network IP range
    ip_address_range = input(
        "Enter your IP address range (ex. 192.168.0.0/24): "
    ) or default_local_network

    # Call the scan function with an ip address range argument
    scan(ip_address_range)
    input("Press the Enter key to exit.")


# Call main function
if __name__ == "__main__":
    main()
```

Windows example run:

```
Network Scanner 3
Enter your IP address range (ex. 192.168.0.0/24): 192.168.9.0/24
Begin emission:
Finished sending 256 packets.
....*.......*.....................................................
..............*.........*.........................................
.....................*.**.......................
Received 287 packets, got 8 answers, remaining 248 packets
Ether / ARP who has 192.168.9.0 says 192.168.9.130
Ether / ARP who has 192.168.9.2 says 192.168.9.130
Ether / ARP who has 192.168.9.3 says 192.168.9.130
Ether / ARP who has 192.168.9.4 says 192.168.9.130
Ether / ARP who has 192.168.9.5 says 192.168.9.130
Ether / ARP who has 192.168.9.6 says 192.168.9.130
Ether / ARP who has 192.168.9.7 says 192.168.9.130
Ether / ARP who has 192.168.9.8 says 192.168.9.130
Ether / ARP who has 192.168.9.9 says 192.168.9.130
Ether / ARP who has 192.168.9.11 says 192.168.9.130
Ether / ARP who has 192.168.9.12 says 192.168.9.130
Ether / ARP who has 192.168.9.13 says 192.168.9.130
Ether / ARP who has 192.168.9.14 says 192.168.9.130
Ether / ARP who has 192.168.9.15 says 192.168.9.130
Ether / ARP who has 192.168.9.16 says 192.168.9.130
Ether / ARP who has 192.168.9.17 says 192.168.9.130
Ether / ARP who has 192.168.9.18 says 192.168.9.130
```

Skip to the end of the program run.

```
Ether / ARP who has 192.168.9.251 says 192.168.9.130
Ether / ARP who has 192.168.9.252 says 192.168.9.130
Ether / ARP who has 192.168.9.253 says 192.168.9.130
Ether / ARP who has 192.168.9.254 says 192.168.9.130
Ether / ARP who has 192.168.9.255 says 192.168.9.130
None
Ether / ARP who has 192.168.9.1 says 192.168.9.130 ==> Ether / ARP is at 5c:a6:e6:16:09:f0 says 192.168.9.1 / Padding
Ether / ARP who has 192.168.9.10 says 192.168.9.130 ==> Ether / ARP is at 6c:0b:84:09:b4:a6 says 192.168.9.10 / Padding
Ether / ARP who has 192.168.9.111 says 192.168.9.130 ==> Ether / ARP is at 0c:8b:7d:6c:3c:f5 says 192.168.9.111 / Padding
Ether / ARP who has 192.168.9.130 says 192.168.9.130 ==> Ether / ARP is at 2c:f0:5d:a2:ac:3e says 192.168.9.130
Ether / ARP who has 192.168.9.138 says 192.168.9.130 ==> Ether / ARP is at 4c:1b:86:9a:2b:3c says 192.168.9.138 / Padding
Ether / ARP who has 192.168.9.102 says 192.168.9.130 ==> Ether / ARP is at 10:2c:6b:be:c6:76 says 192.168.9.102 / Padding
Ether / ARP who has 192.168.9.245 says 192.168.9.130 ==> Ether / ARP is at b0:7f:b9:36:66:9a says 192.168.9.245 / Padding
Ether / ARP who has 192.168.9.103 says 192.168.9.130 ==> Ether / ARP is at 88:c2:55:20:58:b4 says 192.168.9.103 / Padding
None
```

Linux example run:

```
┌──(user㉿kalibill)-[~/Code2]
└─$ sudo python3 network_scanner_3a.py
[sudo] password for user:
Network Scanner 3
Enter your IP address range (ex. 192.168.0.0/24): 192.168.9.0/24
Begin emission:
Finished sending 256 packets.
*******.**.........
Received 20 packets, got 9 answers, remaining 247 packets
Ether / ARP who has 192.168.9.0 says 192.168.9.120
Ether / ARP who has 192.168.9.2 says 192.168.9.120
Ether / ARP who has 192.168.9.3 says 192.168.9.120
Ether / ARP who has 192.168.9.4 says 192.168.9.120
Ether / ARP who has 192.168.9.5 says 192.168.9.120
Ether / ARP who has 192.168.9.6 says 192.168.9.120
Ether / ARP who has 192.168.9.7 says 192.168.9.120
Ether / ARP who has 192.168.9.8 says 192.168.9.120
```

Skip to the end of the program run.

```
Ether / ARP who has 192.168.9.1 says 192.168.9.120 ⟹ Ether / ARP is at 5c:a6:e6:16:09:f0 s
ays 192.168.9.1 / Padding
Ether / ARP who has 192.168.9.10 says 192.168.9.120 ⟹ Ether / ARP is at 6c:0b:84:09:b4:a6
says 192.168.9.10 / Padding
Ether / ARP who has 192.168.9.111 says 192.168.9.120 ⟹ Ether / ARP is at 0c:8b:7d:6c:3c:f5
 says 192.168.9.111 / Padding
Ether / ARP who has 192.168.9.130 says 192.168.9.120 ⟹ Ether / ARP is at 2c:f0:5d:a2:ac:3e
 says 192.168.9.130 / Padding
Ether / ARP who has 192.168.9.138 says 192.168.9.120 ⟹ Ether / ARP is at 4c:1b:86:9a:2b:3c
 says 192.168.9.138 / Padding
Ether / ARP who has 192.168.9.102 says 192.168.9.120 ⟹ Ether / ARP is at 10:2c:6b:be:c6:76
 says 192.168.9.102 / Padding
Ether / ARP who has 192.168.9.101 says 192.168.9.120 ⟹ Ether / ARP is at cc:c0:79:f3:70:02
 says 192.168.9.101 / Padding
Ether / ARP who has 192.168.9.245 says 192.168.9.120 ⟹ Ether / ARP is at b0:7f:b9:36:66:9a
 says 192.168.9.245 / Padding
Ether / ARP who has 192.168.9.103 says 192.168.9.120 ⟹ Ether / ARP is at 88:c2:55:20:58:b4
 says 192.168.9.103 / Padding
None
Press the Enter key to exit.
```

The information display is a bit of a mess. We have returned the information we want, all the host IP and MAC addresses on our network. It is now time to display only the information we want.

Tutorial 2: Showing IP and MAC from List

Time to clean up our display. The information we receive in the **answered_list** is a Python list. This is the information we want. We don't care about the unanswered_list.

We can iterate through the answered_list and make the result easier to read.

1. Save **network_scanner_3a.py** as **network_scanner_3b.py**
2. Remove **print(answered_list.summary())** and **print(unanswered.summary())**
3. Add the following code instead.

```
    # scapy.srp sends and receives packets with custom layer
    # Returns answered and unanswered packet information in two lists
    answered_list, unanswered_list = scapy.srp(
        arp_request_broadcast,
        timeout=2               # Time to wait for a response
    )

    # Iterate through each element in the answered_list
    for element in answered_list:
        print(element)
        print("-" * 25)
```

Windows Example run:

```
WARNING: Wireshark is installed, but cannot read manuf !
Network Scanner 3
Enter your IP address range (ex. 192.168.0.0/24): 192.168.9.0/24
Begin emission:
Finished sending 256 packets.
..*.........*.....................................................
.......................*..............*.........*.....................
.....................................................................
.....................................
Received 312 packets, got 12 answers, remaining 244 packets
QueryAnswer(query=<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  pdst=192.168.9.1 |>>, answer
=<Ether  dst=2c:f0:5d:a2:ac:3e src=5c:a6:e6:16:09:f0 type=ARP |<ARP  hwtype=Ethernet (10Mb)
ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=5c:a6:e6:16:09:f0 psrc=192.168.9.1 hwdst=2c:f0:5d:a
2:ac:3e pdst=192.168.9.130 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00    ' |>>>)
-------------------------
QueryAnswer(query=<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  pdst=192.168.9.10 |>>, answe
r=<Ether  dst=2c:f0:5d:a2:ac:3e src=6c:0b:84:09:b4:a6 type=ARP |<ARP  hwtype=Ethernet (10Mb)
 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=6c:0b:84:09:b4:a6 psrc=192.168.9.10 hwdst=2c:f0:5d
:a2:ac:3e pdst=192.168.9.130 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00' |>>>)
-------------------------
```

pair separator

Linux example run:


```
┌──(user㉿kalibill)-[~/Code2]
└─$ sudo python3 network_scanner_3b.py
Network Scanner 3
Enter your IP address range (ex. 192.168.0.0/24): 192.168.9.0/24
Begin emission:
Finished sending 256 packets.
********.............
Received 22 packets, got 8 answers, remaining 248 packets
QueryAnswer(query=<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  pdst=192.168.9.1 |>>, answer
=<Ether  dst=08:00:27:c5:4d:80 src=5c:a6:e6:16:09:f0 type=ARP |<ARP  hwtype=Ethernet (10Mb)
ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=5c:a6:e6:16:09:f0 psrc=192.168.9.1 hwdst=08:00:27:c
5:4d:80 pdst=192.168.9.120 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00    ' |>>>)
------------------------
QueryAnswer(query=<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  pdst=192.168.9.10 |>>, answe
r=<Ether  dst=08:00:27:c5:4d:80 src=6c:0b:84:09:b4:a6 type=ARP |<ARP  hwtype=Ethernet (10Mb)
 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=6c:0b:84:09:b4:a6 psrc=192.168.9.10 hwdst=08:00:27
:c5:4d:80 pdst=192.168.9.120 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00' |>>>)
------------------------
QueryAnswer(query=<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  pdst=192.168.9.111 |>>, answ
er=<Ether  dst=08:00:27:c5:4d:80 src=0c:8b:7d:6c:3c:f5 type=ARP |<ARP  hwtype=Ethernet (10Mb
) ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=0c:8b:7d:6c:3c:f5 psrc=192.168.9.111 hwdst=08:00:
27:c5:4d:80 pdst=192.168.9.120 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
```

This shows all the information contained in each packet response. The pair separator is the , (comma). There are two pieces of information, the sent packet, and the received packet. We just want the received packet information which is designated [1].

Tutorial 3: Clean Up the Output

1. Save **network_scanner_3b.py** as **network_scanner_3c.py**

The following code will show the source IP and MAC address of the response packet in a nice format. Replace the loop with this code.

```
    # scapy.srp sends and receives packets with custom layer
    # Returns answered and unanswered packet information in two lists
    answered_list, unanswered_list = scapy.srp(
        arp_request_broadcast,
        timeout=2              # Time to wait for a response
    )

    print("-" *40)
    # Iterate through each couple/pair element in the answered_list
    for element in answered_list:
        # psrc IP source address of answer
        source_ip = element[1].psrc

        # hwsrc MAC source address of answer
        source_mac = element[1].hwsrc

        print(f"{source_ip} - {source_mac}")
```

How It Works

Our **answered_list** stores pairs of information. We want the second half, the response half, designated by [1].

1. **for element** goes through our list one couple element (item) at a time.
2. We print the IP src address and MAC source address.

Windows example run:

```
Network Scanner 3
Enter your IP address range (ex. 192.168.0.0/24):
Begin emission
..*.........*.............................................
............................................................
Finished sending 256 packets
......*..........*.......................................
............................................................
Received 476 packets, got 10 answers, remaining 246 packets
------------------------------------
192.168.9.1 - 5c:a6:e6:16:09:f0
192.168.9.10 - 6c:0b:84:09:b4:a6
192.168.9.111 - 0c:8b:7d:6c:3c:f5
192.168.9.101 - 10:2c:6b:be:c6:76
192.168.9.102 - cc:c0:79:f3:70:02
192.168.9.116 - 40:b4:cd:8b:5e:66
192.168.9.130 - 2c:f0:5d:a2:ac:3e
192.168.9.138 - 4c:1b:86:9a:2b:3c
192.168.9.245 - b0:7f:b9:36:66:9a
192.168.9.115 - 58:ef:68:ea:92:a1
Press the Enter key to exit.
```

Linux example run:

```
Network Scanner 3
Enter your IP address range (ex. 192.168.0.0/24):
Begin emission
******.**
Finished sending 256 packets
.****.
Received 16 packets, got 13 answers, remaining 243 packets
------------------------------------
192.168.9.1 - 5c:a6:e6:16:09:f0
192.168.9.10 - 6c:0b:84:09:b4:a6
192.168.9.111 - 0c:8b:7d:6c:3c:f5
192.168.9.130 - 2c:f0:5d:a2:ac:3e
192.168.9.138 - 4c:1b:86:9a:2b:3c
192.168.9.101 - 10:2c:6b:be:c6:76
192.168.9.102 - cc:c0:79:f3:70:02
192.168.9.116 - 40:b4:cd:8b:5e:66
192.168.9.245 - b0:7f:b9:36:66:9a
192.168.9.112 - c4:5b:be:f9:d6:94
192.168.9.122 - a0:20:a6:14:61:f6
192.168.9.115 - 58:ef:68:ea:92:a1
192.168.9.100 - f0:f5:bd:b8:bc:98
Press the Enter key to exit.
```

We now have the source IP and MAC address of all responding hosts on our network.

Test your Python file on Windows and Kali Linux.

Assignment Submission

Attach all program files and screenshots of your results from both operating systems to the assignment in BlackBoard.