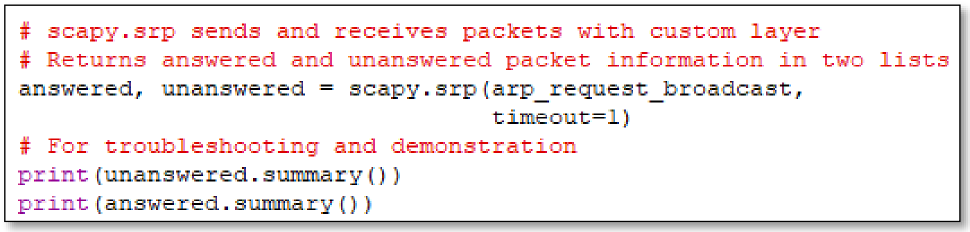
Part 3: Python Network Scanner

Contents

Time required: 90 minutes

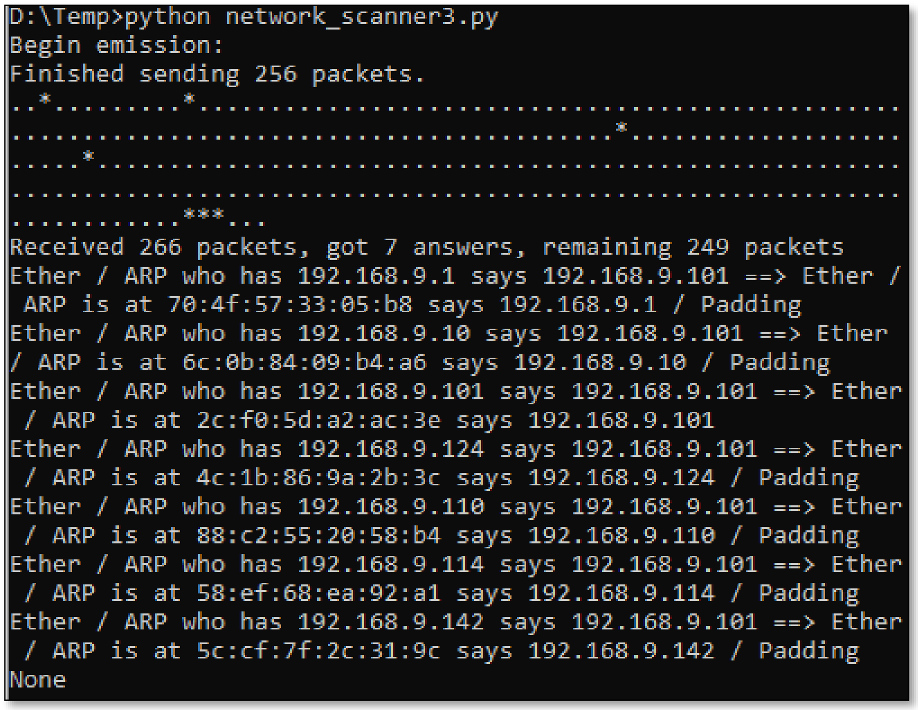
Send and Receive ARP Request

1. Save **network_scanner2.py** as **network_scanner3.py**
2. Comment out the first three **.show()** methods. These were for troubleshooting and demonstration purposes.
3. Add the following to your **scan()** function.



```
# scapy.srp sends and receives packets with custom layer
# Returns answered and unanswered packet information in two lists
answered, unanswered = scapy.srp(arp_request_broadcast,
                                 timeout=1)
# For troubleshooting and demonstration
print(unanswered.summary())
print(answered.summary())
```

Example run:



```
D:\Temp>python network_scanner3.py
Begin emission:
Finished sending 256 packets.
..*.........*..............................................
..........................................*.................
.....*......................................................
............................................................
............***...
Received 266 packets, got 7 answers, remaining 249 packets
Ether / ARP who has 192.168.9.1 says 192.168.9.101 ==> Ether /
 ARP is at 70:4f:57:33:05:b8 says 192.168.9.1 / Padding
Ether / ARP who has 192.168.9.10 says 192.168.9.101 ==> Ether
/ ARP is at 6c:0b:84:09:b4:a6 says 192.168.9.10 / Padding
Ether / ARP who has 192.168.9.101 says 192.168.9.101 ==> Ether
 / ARP is at 2c:f0:5d:a2:ac:3e says 192.168.9.101
Ether / ARP who has 192.168.9.124 says 192.168.9.101 ==> Ether
 / ARP is at 4c:1b:86:9a:2b:3c says 192.168.9.124 / Padding
Ether / ARP who has 192.168.9.110 says 192.168.9.101 ==> Ether
 / ARP is at 88:c2:55:20:58:b4 says 192.168.9.110 / Padding
Ether / ARP who has 192.168.9.114 says 192.168.9.101 ==> Ether
 / ARP is at 58:ef:68:ea:92:a1 says 192.168.9.114 / Padding
Ether / ARP who has 192.168.9.142 says 192.168.9.101 ==> Ether
 / ARP is at 5c:cf:7f:2c:31:9c says 192.168.9.142 / Padding
None
```

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.

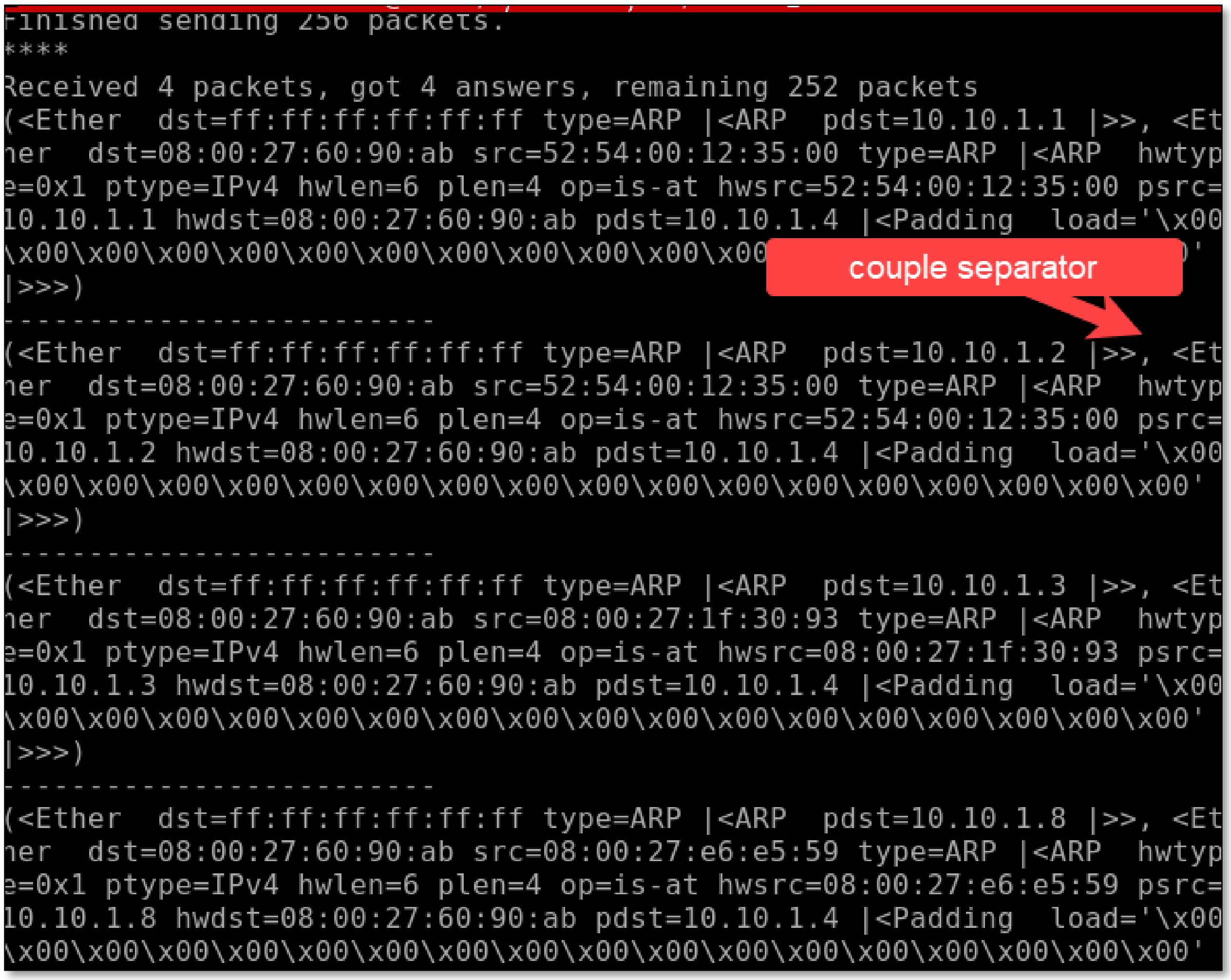
Showing IP and MAC from List

Time to clean up our display. The information we receive in the **answered_list** is a Python list. We can iterate through the list and make the list look nicer.

1. Remove **print(answered.summary())** and **print(unanswered.summary())**

Add the following code instead.

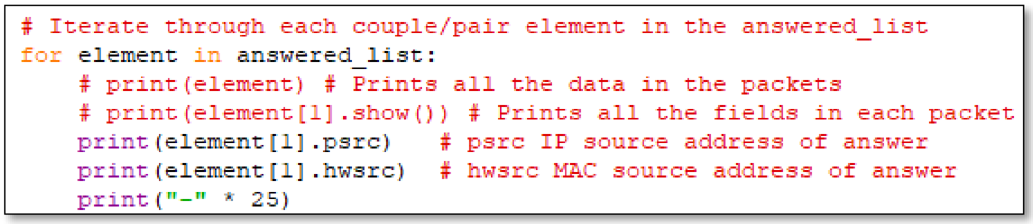
```
# Iterate through each element in the answered_list
for element in answered_list:
    print(element)
    print("-" * 25)
```



```
Finished sending 256 packets.
****
Received 4 packets, got 4 answers, remaining 252 packets
(<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  pdst=10.10.1.1 |>>, <Et
her  dst=08:00:27:60:90:ab src=52:54:00:12:35:00 type=ARP |<ARP  hwtyp
e=0x1 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=52:54:00:12:35:00 psrc=
10.10.1.1 hwdst=08:00:27:60:90:ab pdst=10.10.1.4 |<Padding  load='\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00                         0'
|>>>)
-------------------------
(<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  pdst=10.10.1.2 |>>, <Et
her  dst=08:00:27:60:90:ab src=52:54:00:12:35:00 type=ARP |<ARP  hwtyp
e=0x1 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=52:54:00:12:35:00 psrc=
10.10.1.2 hwdst=08:00:27:60:90:ab pdst=10.10.1.4 |<Padding  load='\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
|>>>)
-------------------------
(<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  pdst=10.10.1.3 |>>, <Et
her  dst=08:00:27:60:90:ab src=08:00:27:1f:30:93 type=ARP |<ARP  hwtyp
e=0x1 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=08:00:27:1f:30:93 psrc=
10.10.1.3 hwdst=08:00:27:60:90:ab pdst=10.10.1.4 |<Padding  load='\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
|>>>)
-------------------------
(<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  pdst=10.10.1.8 |>>, <Et
her  dst=08:00:27:60:90:ab src=08:00:27:e6:e5:59 type=ARP |<ARP  hwtyp
e=0x1 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=08:00:27:e6:e5:59 psrc=
10.10.1.8 hwdst=08:00:27:60:90:ab pdst=10.10.1.4 |<Padding  load='\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

This shows all the information contained in each packet response. The couple 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].

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



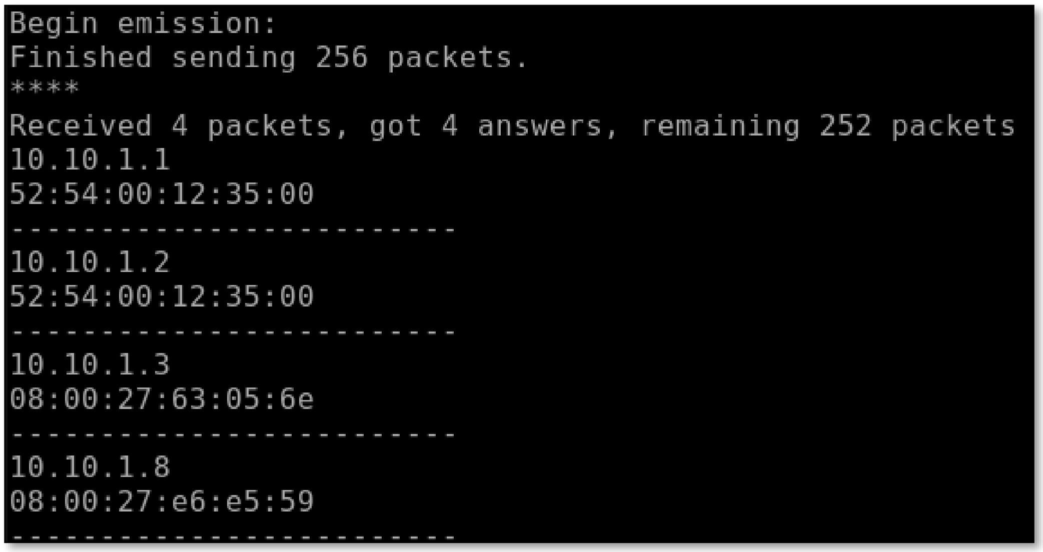
```python
# Iterate through each couple/pair element in the answered_list
for element in answered_list:
    # print(element) # Prints all the data in the packets
    # print(element[1].show()) # Prints all the fields in each packet
    print(element[1].psrc)    # psrc IP source address of answer
    print(element[1].hwsrc)  # hwsrc MAC source address of answer
    print("-" * 25)
```

How It Works

Our **answered_list** stores couples/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.

Example run:



```
Begin emission:
Finished sending 256 packets.
****
Received 4 packets, got 4 answers, remaining 252 packets
10.10.1.1
52:54:00:12:35:00
-------------------------
10.10.1.2
52:54:00:12:35:00
-------------------------
10.10.1.3
08:00:27:63:05:6e
-------------------------
10.10.1.8
08:00:27:e6:e5:59
-------------------------
```

We now have the source IP and MAC address of all responding machines 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.