Part 4: Python Network Scanner

Contents

Time required: 30 minutes

Network Scanner – The Final Chapter

Save your **network_scanner_3.py** as **network_scanner.py**

We have everything working. We can make it look better. Let's format our response packets and print a nice title.

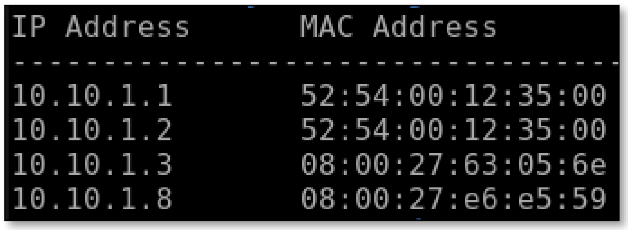
- For a cleaner look, let's get rid of the feedback from the scapy.srp packet sending. **verbose=False** turns off all srp feedback.
- We added a 1 second timeout to wait for responses.
- We added a nice heading and put the IP and MAC information on the same line.

```
# srp sends and receives packets with custom layer
# returns answered and unanswered packet information in 2 lists
# [0] returns element 0 of the first list of answered packets
answered_list = scapy.srp(arp_request_broadcast,
                          timeout=1,          # timeout=1 second
                          verbose=False)[0]   # no feedback on request

# Print a nice heading
print("IP\t\tMAC Address")
print("-" * 35)

# Iterate through each element in the answered_list
for element in answered_list:
    # psrc IP source address of answer
    # hwsrc MAC source address of answer
    print(element[1].psrc + "\t" + element[1].hwsrc)
```

Example run:



```
IP Address          MAC Address
-----------------------------------
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
```

Our finished product looks pretty good!

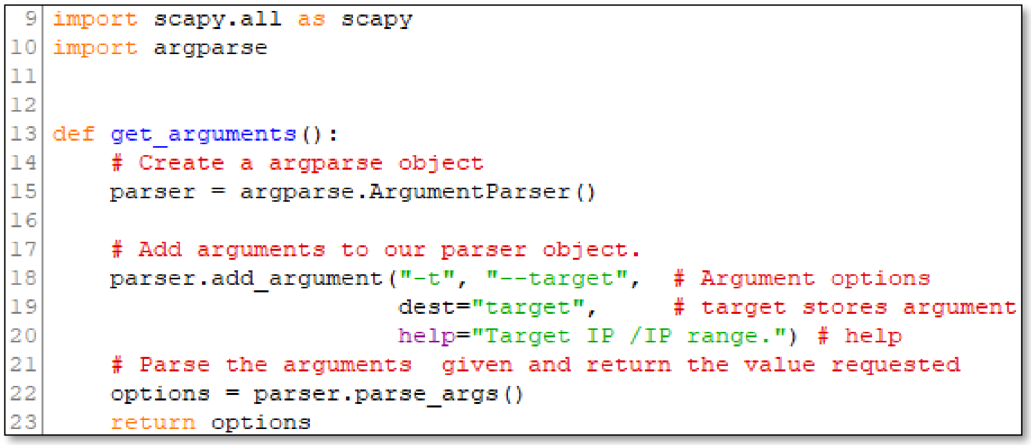
Passing Arguments

One last item. We have to hard code the program to get it to work. With command line programs it is common to pass arguments to change how the program functions.

python network_scanner.py -t 192.168.9.0/24

This is an example of running our scanner with a -t switch and providing a network address argument.

We are going to refactor our code a bit to make it more modular. Each function will have a specific purpose.



```python
import scapy.all as scapy
import argparse


def get_arguments():
    # Create a argparse object
    parser = argparse.ArgumentParser()

    # Add arguments to our parser object.
    parser.add_argument("-t", "--target",  # Argument options
                        dest="target",     # target stores argument
                        help="Target IP /IP range.") # help
    # Parse the arguments  given and return the value requested
    options = parser.parse_args()
    return options
```

This function imports the **argparse** library. This allows us to capture arguments from the command line.

```python
def scan(ip):
    # Create ARP request for targeted ip address
    # pdst is Target IP address
    arp_request = scapy.ARP(pdst=ip)

    # 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

    # srp sends and receives packets with custom layer
    # returns answered and unanswered return packet information in 2 lists
    # [0] returns element 0 of the first list of answered packets
    answered_list = scapy.srp(arp_request_broadcast,
                              timeout=1,         # timeout=1 second
                              verbose=False)[0]  # no feedback on request

    return answered_list
```

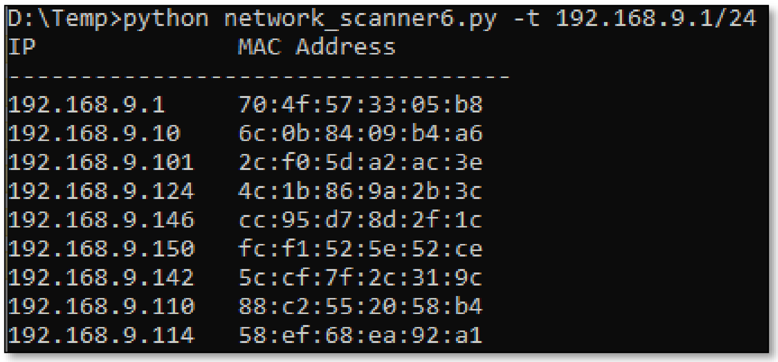
We have taken everything out of the scan function except the code that scans the network.

```python
def print_result(answered_list):
    # Print title
    print("IP Address\tMAC Address")
    print("-" * 35)
    # Iterate through each element in the answered_list
    for element in answered_list:
        # psrc IP source address of answer
        # hwsrc MAC source address of answer
        print(element[1].psrc + "\t" + element[1].hwsrc)
```

The print_result function prints the results of the scan function.

```python
def main():
    # Get IP argument from command line
    options = get_arguments()
    # Use IP argument to scan network
    scan_result = scan(options.target)
    # Print list results
    print_result(scan_result)
```

Example run:



```
D:\Temp>python network_scanner6.py -t 192.168.9.1/24
IP               MAC Address
-----------------------------------
192.168.9.1      70:4f:57:33:05:b8
192.168.9.10     6c:0b:84:09:b4:a6
192.168.9.101    2c:f0:5d:a2:ac:3e
192.168.9.124    4c:1b:86:9a:2b:3c
192.168.9.146    cc:95:d7:8d:2f:1c
192.168.9.150    fc:f1:52:5e:52:ce
192.168.9.142    5c:cf:7f:2c:31:9c
192.168.9.110    88:c2:55:20:58:b4
192.168.9.114    58:ef:68:ea:92:a1
```

That's it, we are done. We can use this hand-built network scanner on any 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.