Erik Kriz
Paige Sheridan
DNS Poisoning Lab

Student task 1:

_____In this task, we are simply providing screenshots to prove that we have successfully configured our Machine A to resolve queries from Machine B. In Figure 1.1., we ping the site *www.cse.uconn.edu* which results in 100% packet loss. However, we can prove that the DNS cache was still used when pinging and Machine B was still the server being queried. In Figure 1.2., the captured Wireshark packets show that Machine A (10.0.2.7) queried Machine B (10.0.2.8) and the DNS cache was used. This indicates that our setup was configured correctly.

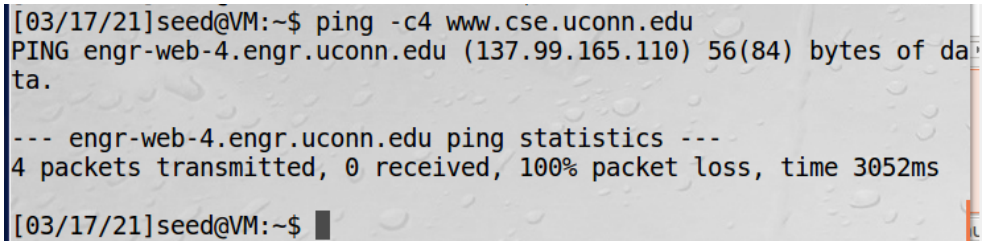

Figure 1.1. Here I ping cse.uconn.edu on my user machine, with my DNS machine set up as a DNS server.

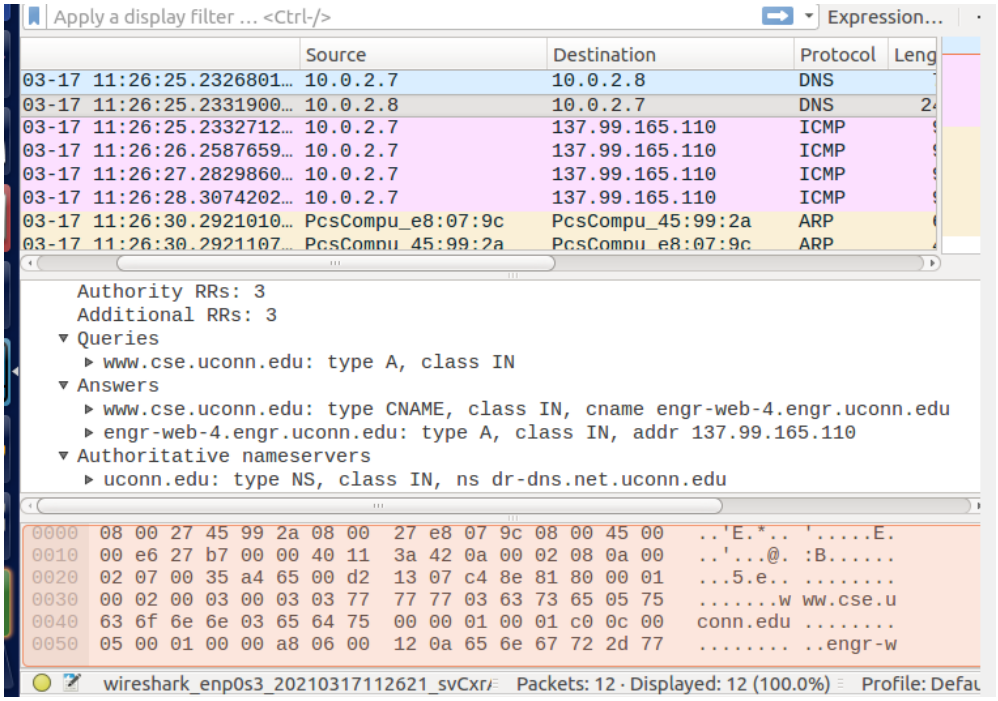

Figure 1.2. You can see 2 DNS packets here, one from the user computer (10.0.2.7) to the DNS server (10.0.2.8), and one from the DNS server to the user. The second packet is highlighted, and you can see that the answers section tells the user where to find cse.uconn.edu.

Student Task 2:

In this task, we are simply using screenshots to prove that the Forward and Reverse Lookup Zone Files were configured correctly. Note that Machine B (the DNS server) is 10.0.2.7 in these screenshots, rather than 10.0.2.8 in the previous task. First we use a simple `dig www.example.com` command to have Machine A ask the local DNS to resolve the IP address of www.example.com. The result of this query can be seen in Figure 2.1. below. The returned IP addresses correctly match all of the values we configured in the Lookup Zone Files. We can also see that this response did go through our local DNS server machine with the line '`SERVER: 10.0.2.7#53(10.0.2.7)`'.

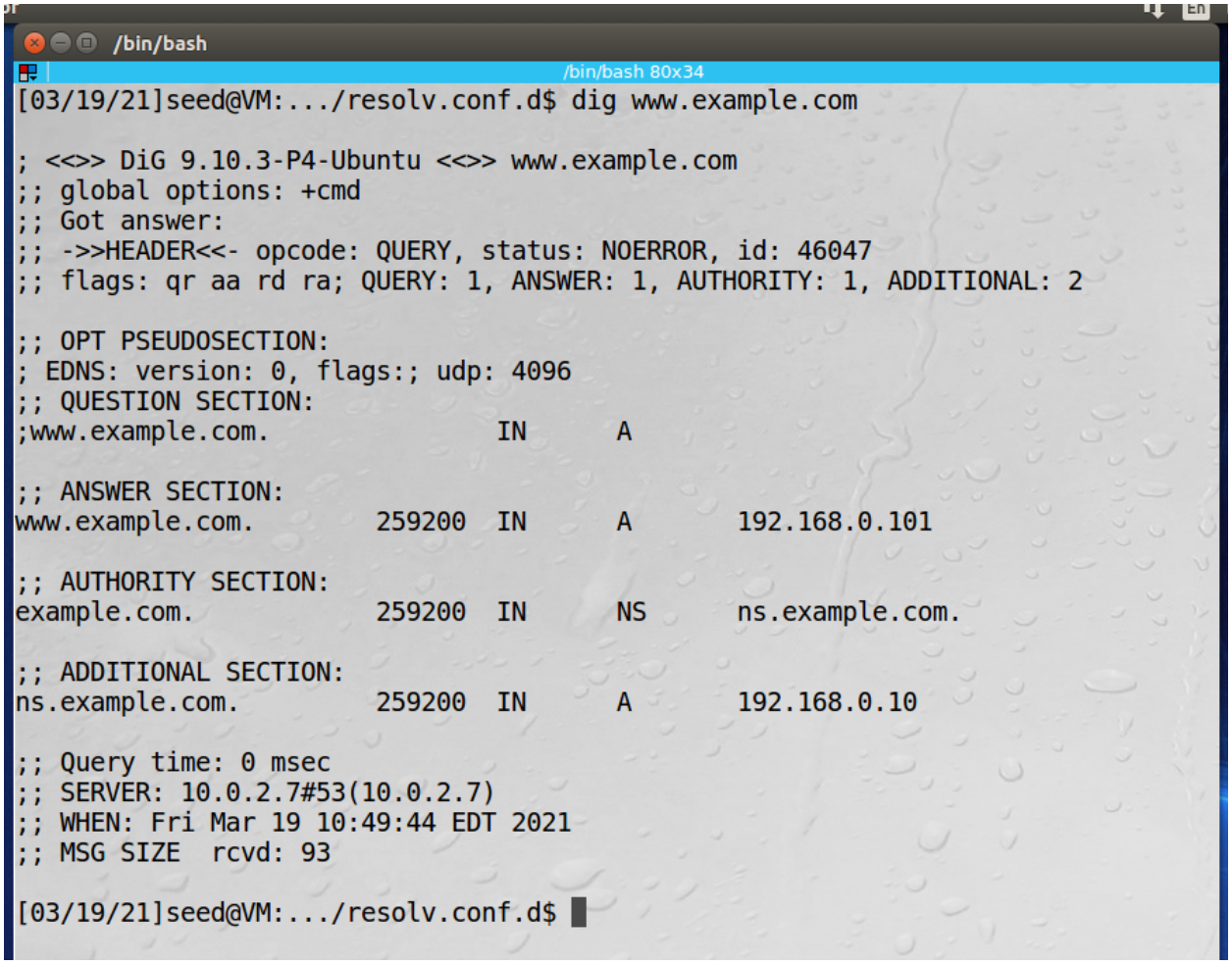

Figure 2.1. This is the resulting of my 'dig www.example.com' command.

After successfully doing this, we will proceed to do the reverse lookup. In order to do the reverse, we use the `-x` flag in the dig command. The entire command we will use is '`dig -x 192.168.0.101`'. After running this command, we are given www.example.com in the

answer section, and the corresponding IP address for the nameserver that we configured in the Lookup Zone Files. This shows us that we successfully completed this task.

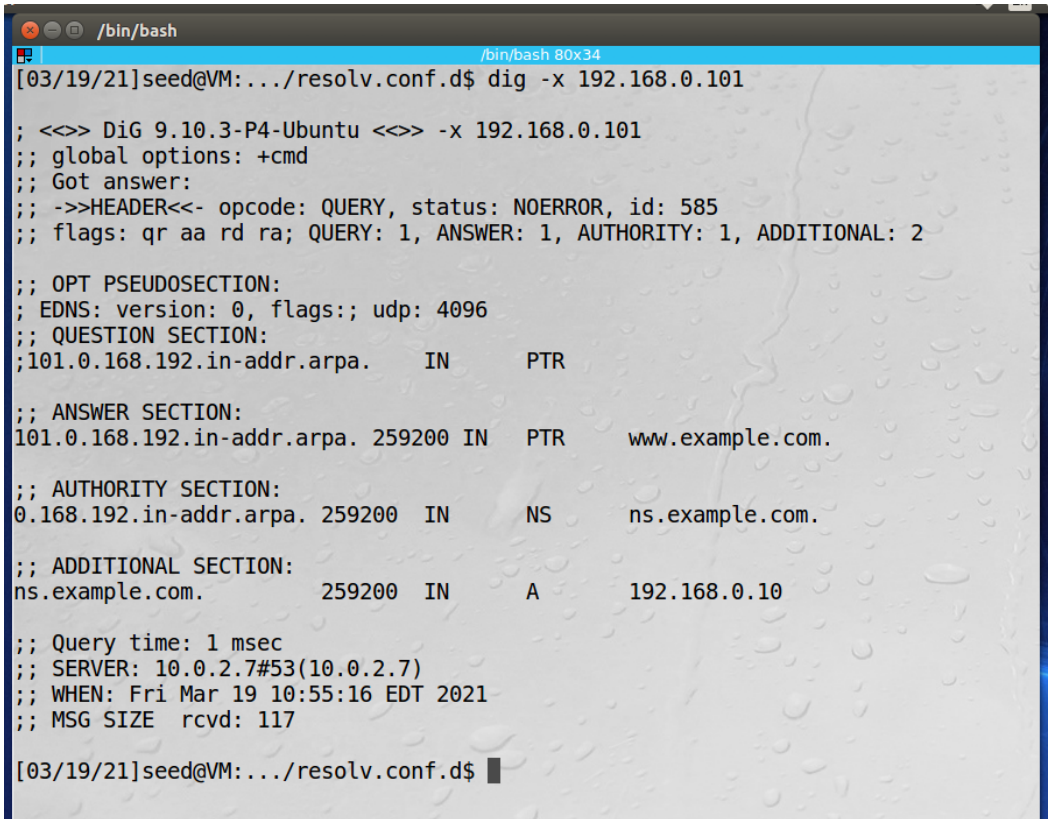

Figure 2.2. The result of the 'dig 192.168.0.101' command

Student Task 3:

In this task, we will be redirecting *www.bank32.com* to IP 8.8.8.8. In Figures 3.1. And 3.2., we can see the hosts of the DNS server and user machine before we complete the task. We display these using a simple `cat` command.

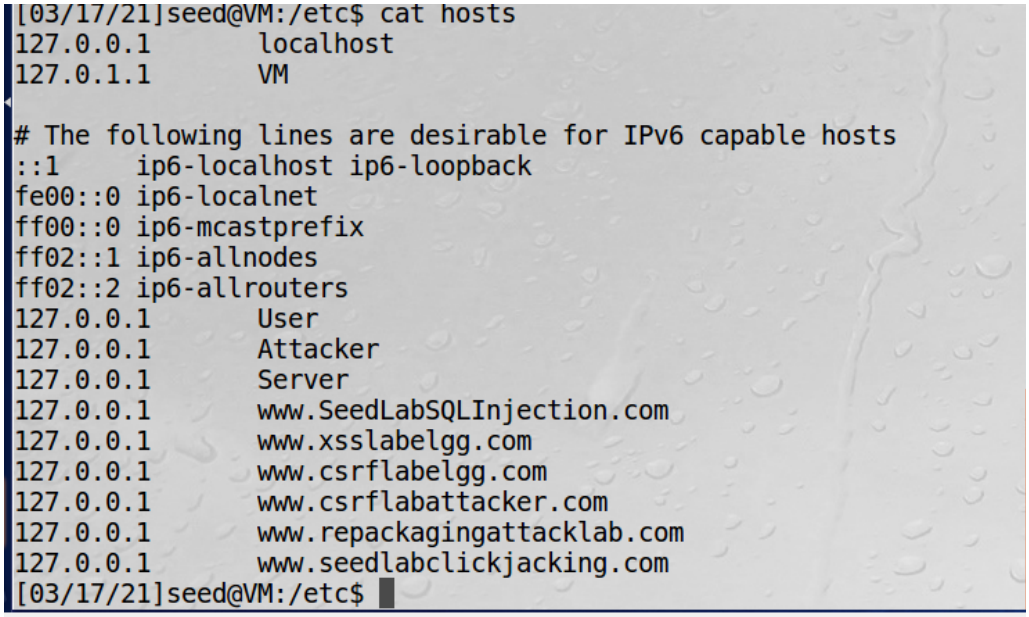

Figure 3.1. Hosts of DNS server machine before completion of task

Figure 3.2. Hosts of user machine before completion of task

To complete this task, we edit the `host` file on the user machine. To do this, we simply open the file and add the line '`8.8.8.8 www.bank32.com`' to the end to create an association. The changed file on the user machine can be seen in Figure 3.3. below.

Figure 3.3. The changed host file for the user machine

From here, we turn on Wireshark capture on Machine B and then ping www.bank32.com on the user machine. The results of this can be seen in Figure 3.4. below. Machine A is using its `hosts` file to resolve the IP address of www.bank32.com rather than sending the request to the DNS server. The ICMP Echo request process can be seen in Figure 3.5. Below. This means that we have successfully completed this task.

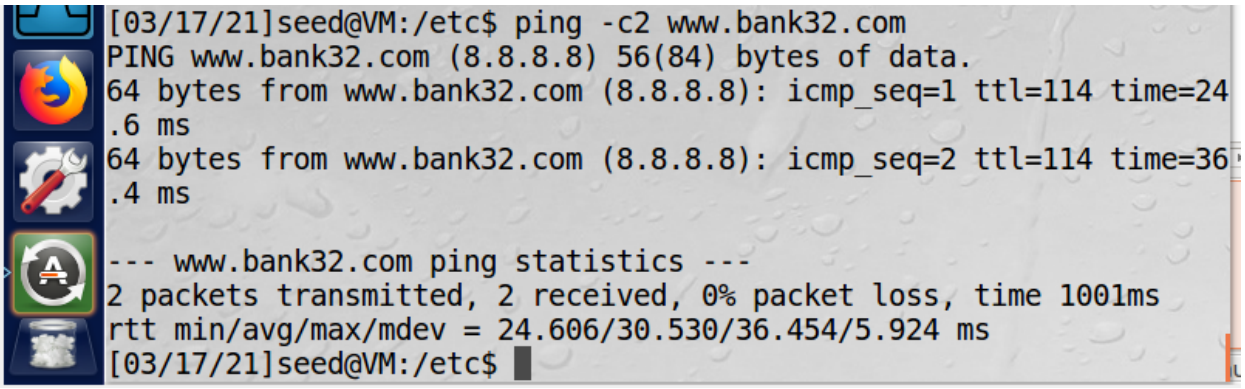

Figure 3.4. The user machine pinging www.bank32.com, and pinging 8.8.8.8 in order to do so.

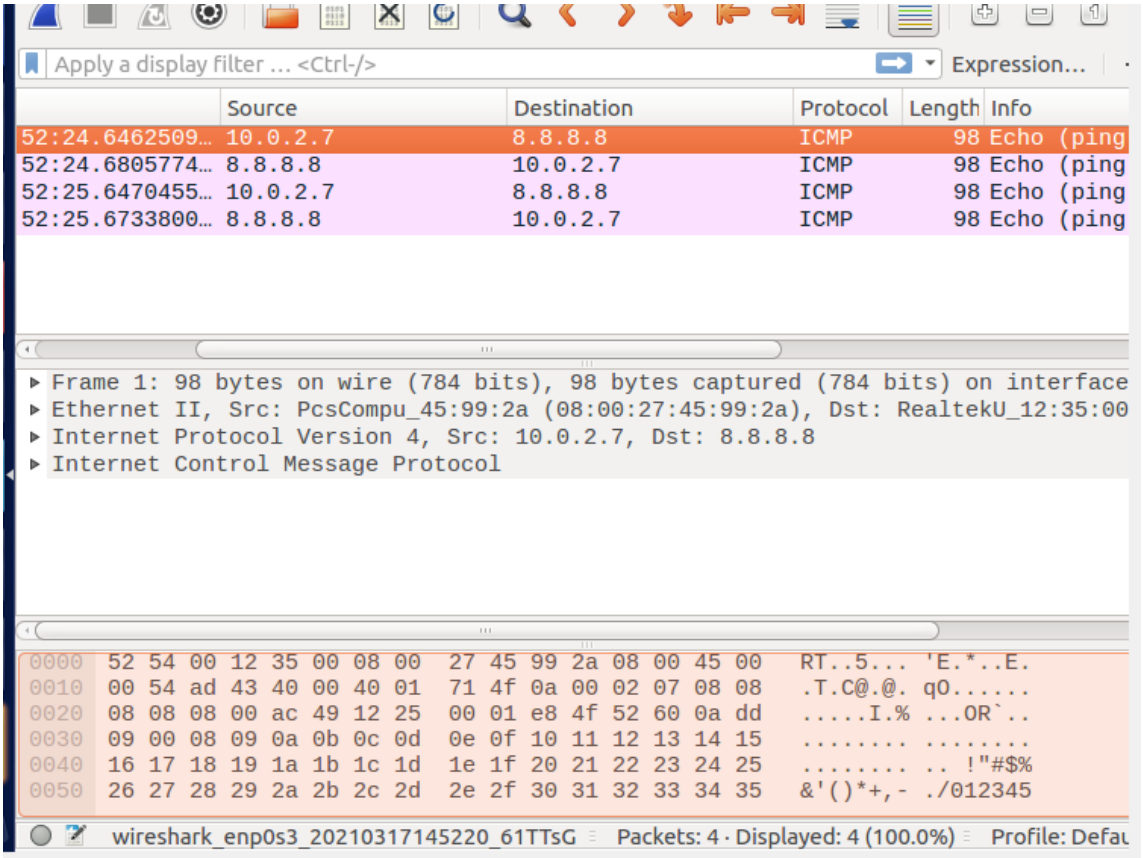

Figure 3.5. Wireshark on the DNS server machine confirms that the user machine does not even contact the DNS server machine, as it only needs to check hosts

Student Task 1.4:

In this task, we will be using the Netwox 105 tool on Machine C to poison the cache of Machine B. Before completing this task, we use Wireshark capture to look at packets of Machine A using the command `dig www.bank32.com` before actually running the attack. The results of this capture can be seen in Figure 1.4.1. below. The packet highlighted is the packet the final DNS server sends to our local DNS server.

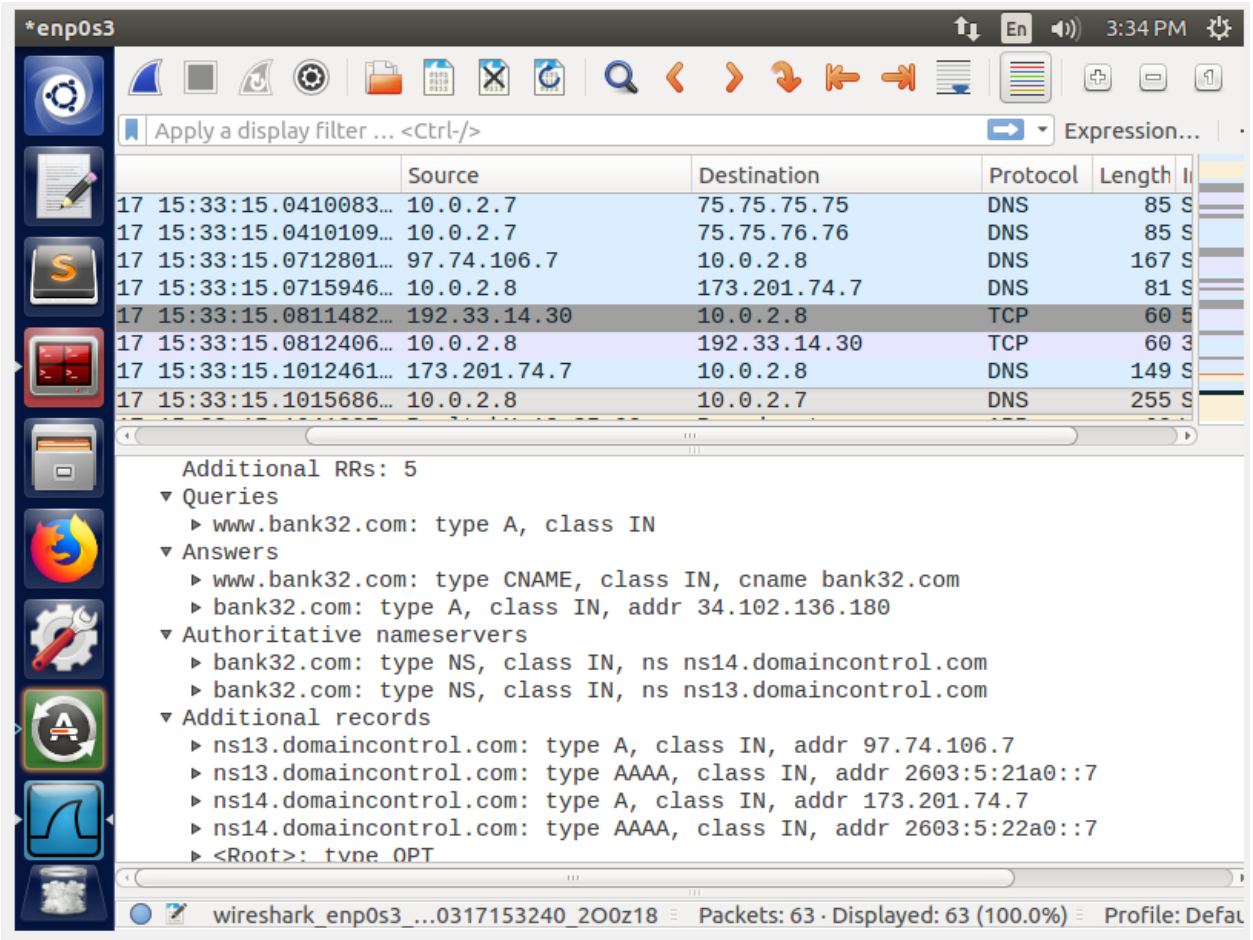

Figure 1.4.1. Wireshark capture of 'dig www.bank32.com' prior to completing the attack

From here, we will run the `netwox 105` command on Machine C with all of the appropriate information for our attack. Through this command, we tell the DNS server machine that the IP address of *www.bank32.com* is "137.99.165.110", which is actually the IP address of *www.cse.uconn.edu*. We also tell the DNS server machine that the nameserver of this site, *ns.bank32.com*, is at the IP address "1.2.3.4".

```
[03/17/21]seed@VM:~$ sudo netwox 105 -h "www.bank32.com" -H "137.
99.165.110" -a "ns.bank32.com" -A "1.2.3.4" -f "src host 10.0.2.8
" -T 600 -s raw
```

Figure 1.4.2. The full netwox command used in Machine C for this attack

After running this command, we will attempt to dig *www.bank32.com* on the user machine again. As can be seen in Figure 1.4.3., the machine believes that the IP of *www.bank32.com* is actually "137.99.165.110", which is actually the address of *www.cse.uconn.edu*. The machine also believes that the nameserver of the website is at IP address "1.2.3.4", when in reality this is a fake address.

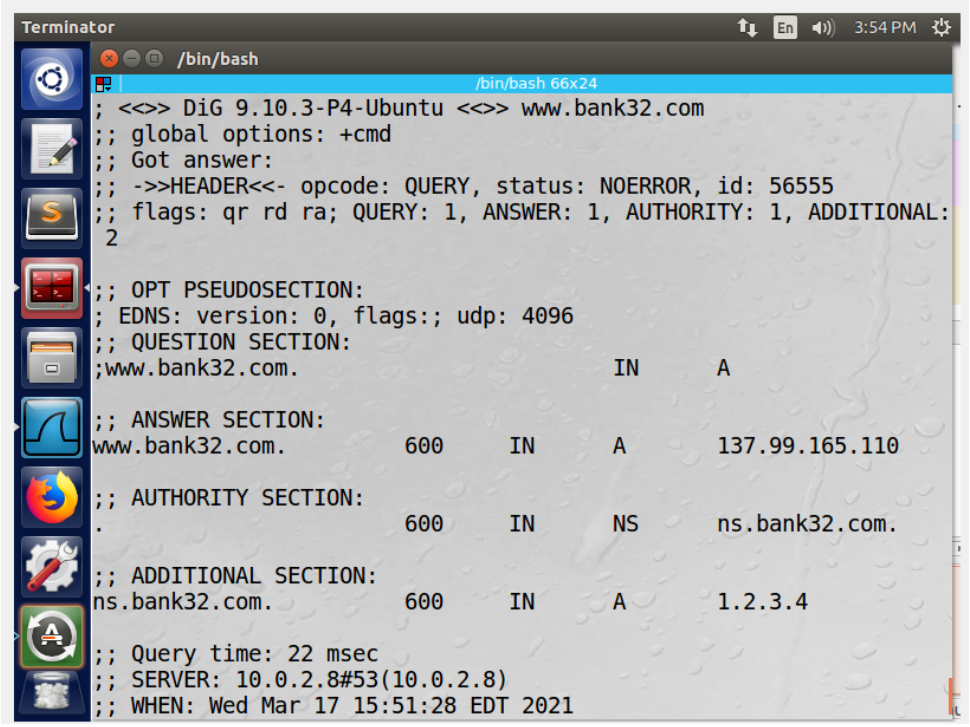

Figure 1.4.3. The result of the command 'dig www.bank32.com' on Machine A after the attack has been completed

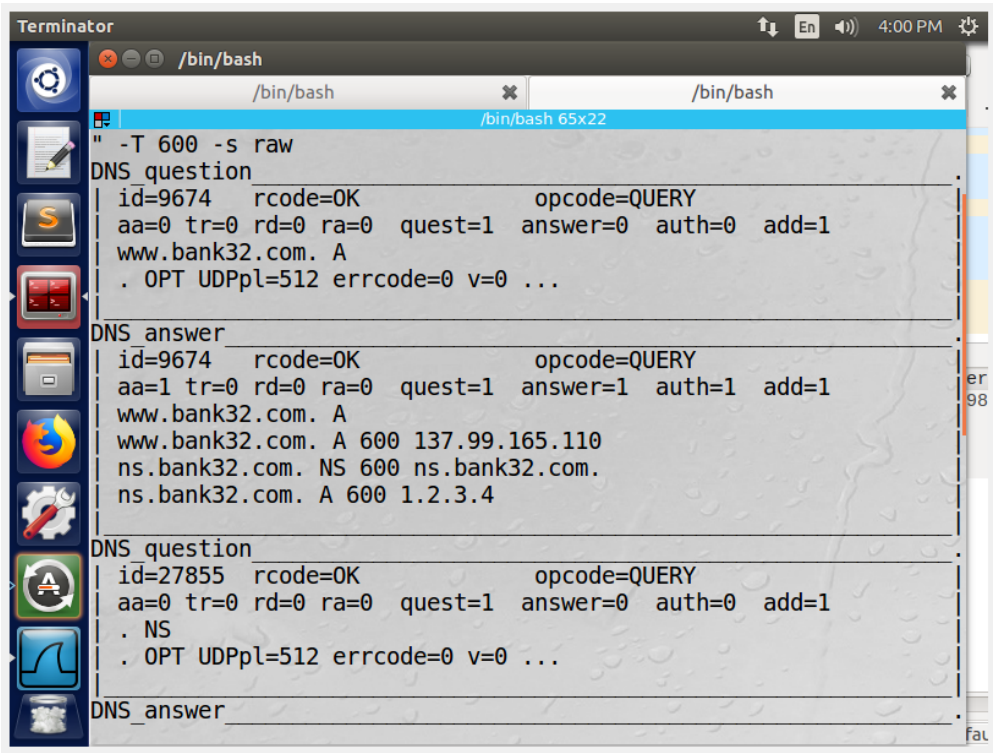

Figure 1.4.4. The attack machine running its netwox command after it has intercepted the DNS query for 'www.bank32.com'

Finally, Figure 1.4.5. below shows the wireshark from the attacker machine. The highlighted packet is the spoofed packet from the attacker machine. This is because it has the address for *www.bank32.com* as "137.99.156.110", which is not correct. What's interesting is that there are still several DNS packets detected after this one. These packets are the true DNS server for *www.bank32.com* responding to the dig command from the user machine. But since the spoofed packet arrived first, that packet is what ultimately got cached.

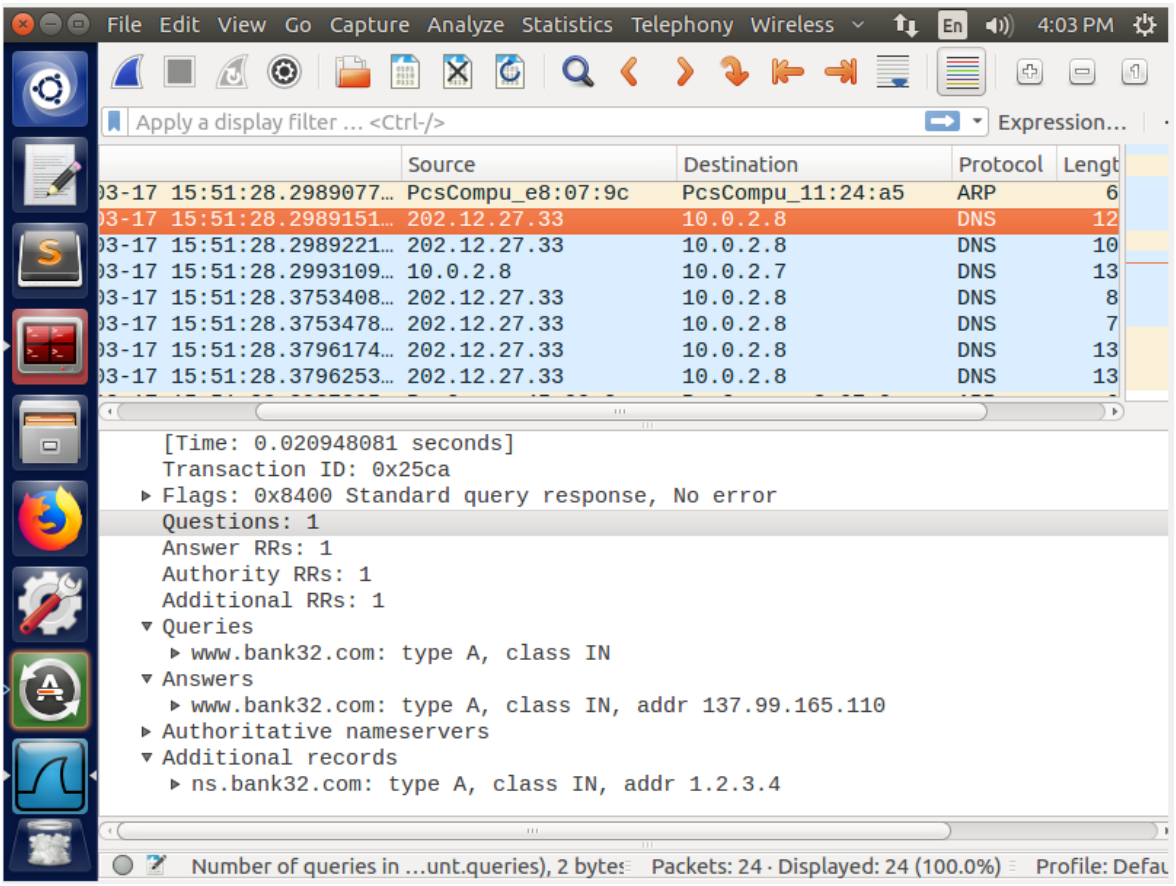

Figure 1.4.5. The Wireshark capture from the attack, displaying the spoofed packet from the attacker machine

Student Task 4:

In this task we will carry out the Authority Section Attack. In order to complete this task, we want both *example.net* and *google.com* to appear in the 'authoritative name servers' section of our Wireshark packets. To do this, we will follow the technique shown in Task 1.4.5.

For this attack to work, we use the provided `spoofauthority2.py` file. We do not need to change anything in this file for the attack to work, so I won't include a screenshot of the file. We then begin Wireshark packet capture and launch the attack by giving ourselves executable permissions using `chmod` and then using the command `sudo`

`./spoofauthority2`. The results of launching the attack on Machine C (10.0.2.15) can be seen in the screenshot below:

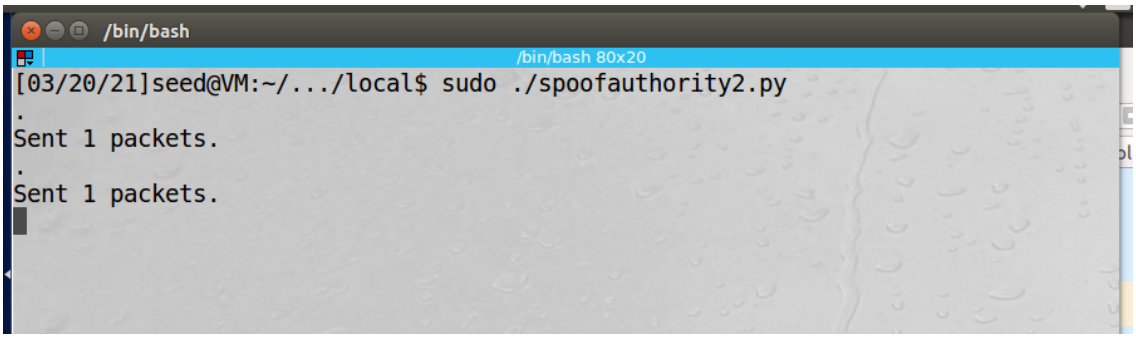

Figure 4.1. Launching the Authority Section Attack and receiving the response that two packets were sent

Once we receive this response from launching the attack, we can use the command `dig app.example.net` on our Machine A (10.0.2.6) to check if the attack worked. As we can see in Figure 4.2., both *example.net* and *google.com* are present as name servers in the authority section of the response. We can also see that the response comes from our local DNS server Machine B (10.0.2.7) from the line `'SERVER: 10.0.2.7#53(10.0.2.7)'`.

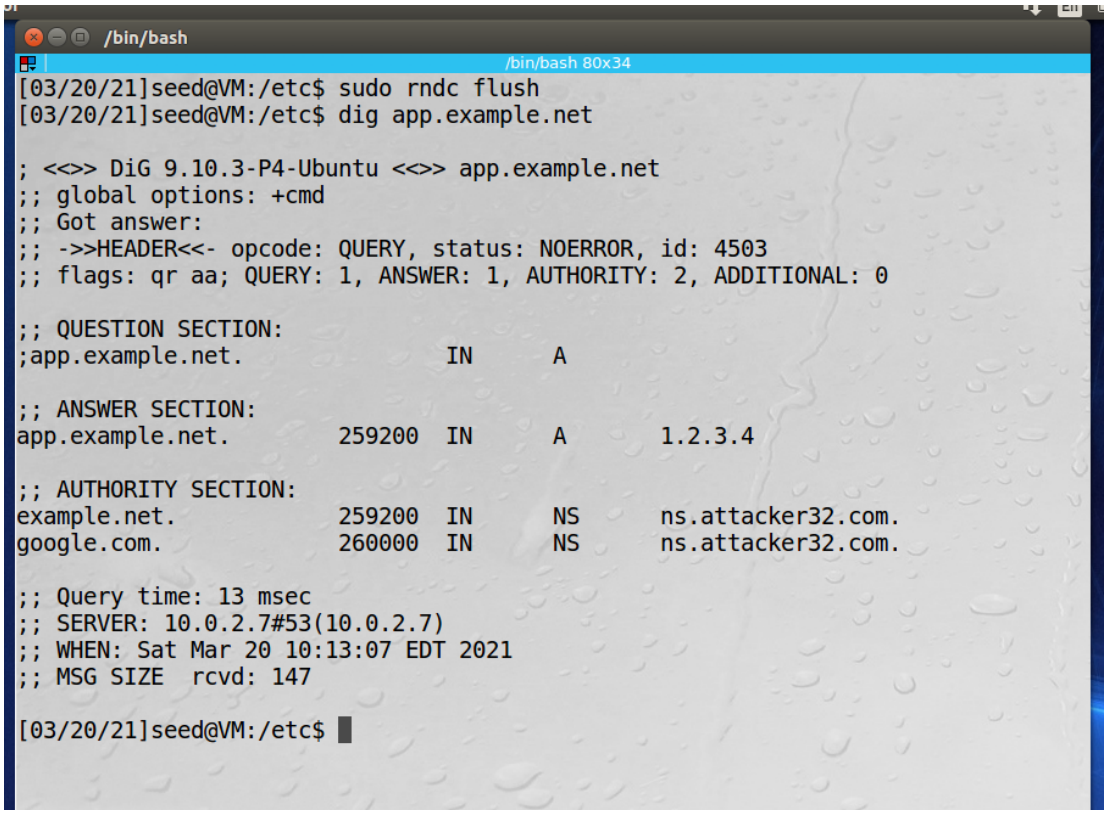

Figure 4.2. The result of the 'dig app.example.net' command on Machine A; both 'example.net' and google.com' are listed as name servers and the response comes from our Machine B (10.0.2.7)

We can also observe packets in Wireshark to ensure that our attack worked. As can be seen in Figure 4.3., the packet sent from our local DNS server Machine B (10.0.2.7) to our user Machine A (10.0.2.6) has both *example.net* and *google.com* listed under the 'Authoritative nameservers' section of the packet. This is also proof that our attack was a success.

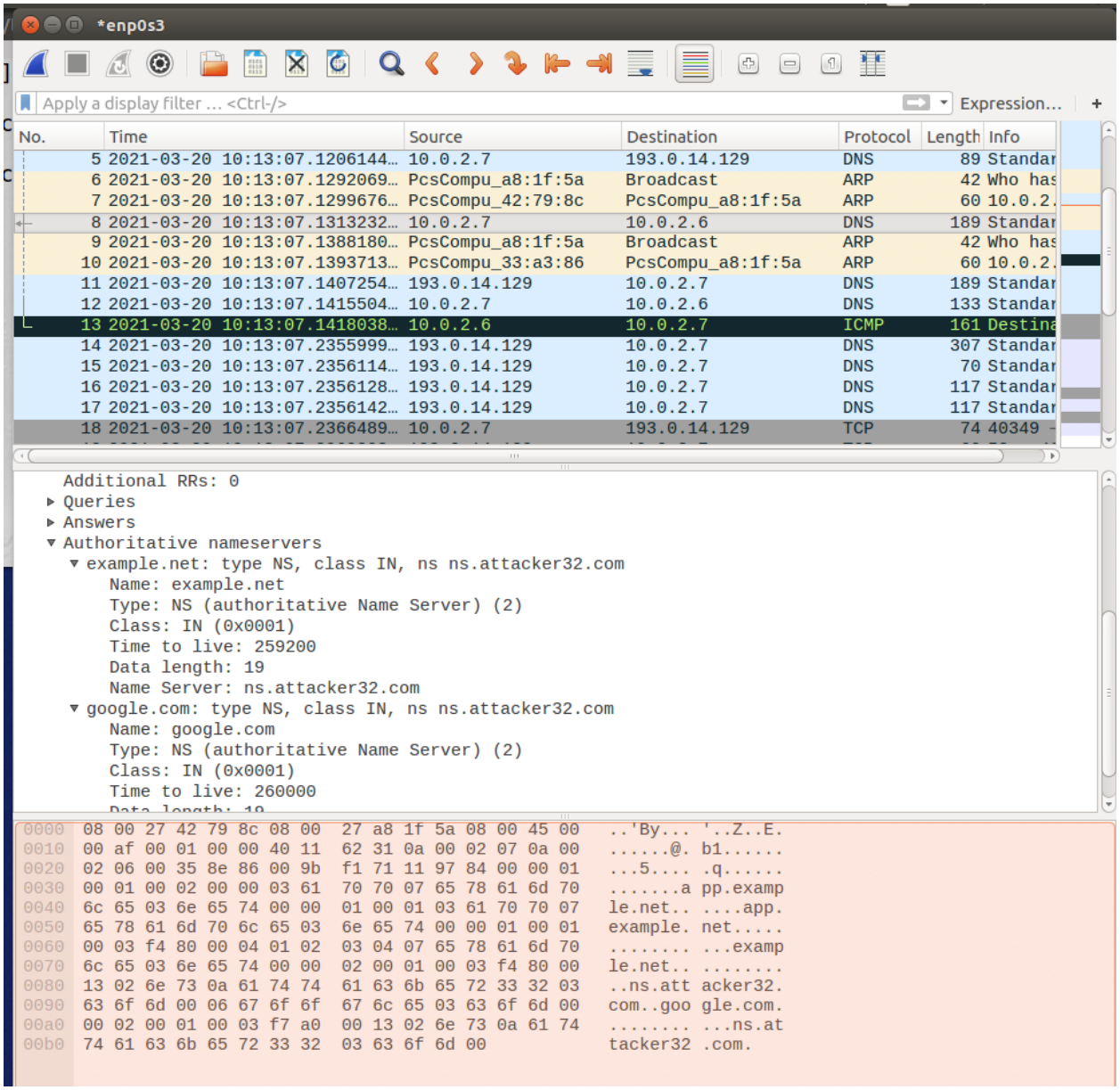

Figure 4.3. Using Wireshark capture, we find the packet sent from Machine B to Machine A containing both 'example.net' and 'google.com' under the 'Authoritative nameservers' section

Student Task 5

In this task, we will be implementing a Kaminsky attack. We first set up our machines; however, most of this has already been done through Task 1, so we do not have to change much. The only change we needed to make was changing the forward zone in

`/etc/bind/named.conf`. This is set up exactly like the first ones we have done, so we simply insert the code given in the lab guide and proceed to dump the cache, flush the DNS cache, and restart bind9.

We proceed to configuring the Attacker's VM. To do this, we use the `attacker32.com.zone` and `example.com.zone` files given to us. In Figure 5.1., it can be seen that we simply place our attacker machine's IP (10.0.2.15) in all of the places it needs to be. We follow the example given in the lab guide to do this.

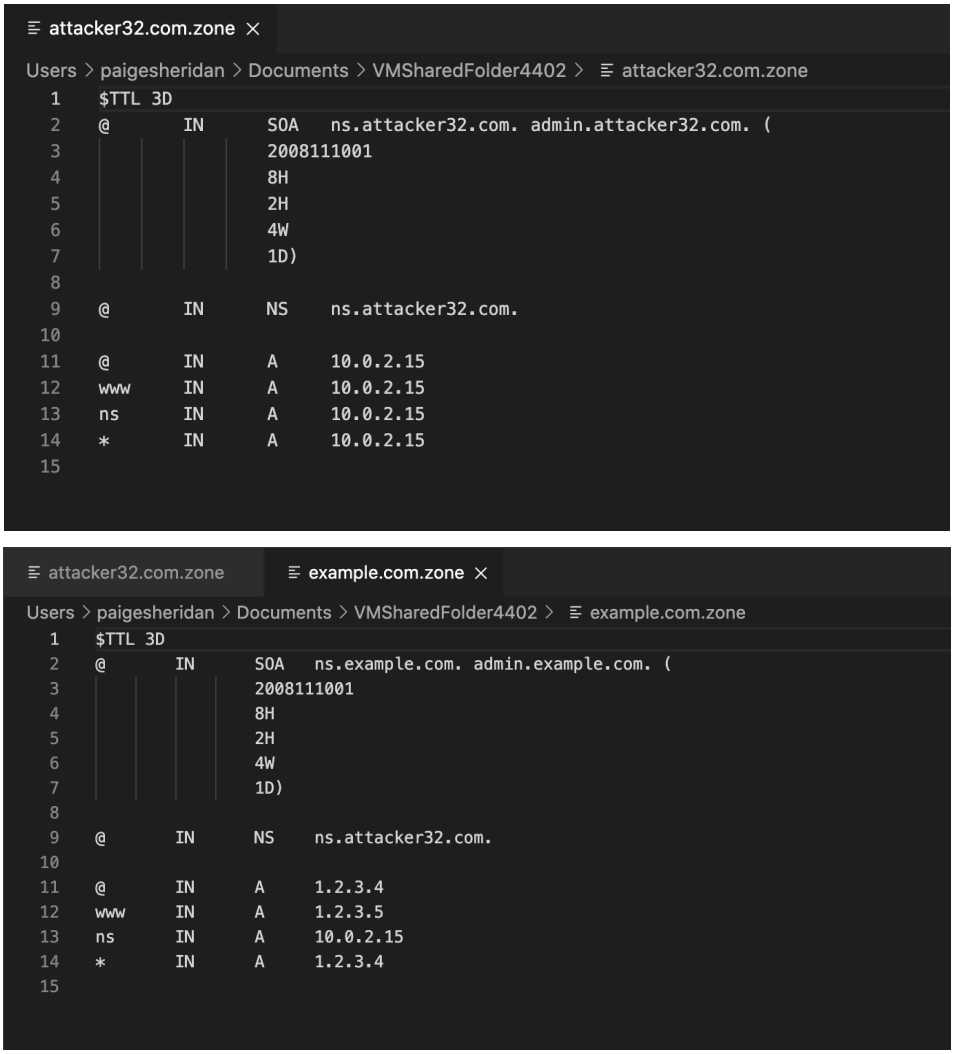

Figure 5.1. The changed 'attacker32.com.zone" and "example.com.zone" files for our specific personal network setup

We then add the entries given in the lab guide to the `/etc/bind/named.conf` file and test the setup by using the command `dig ns.attacker32.com` on Machine A (10.0.2.6). As we can see in Figure 5.2. below, the answer comes from the "attacker32.com.zone" file that was set up with the attacker, which is what we want.

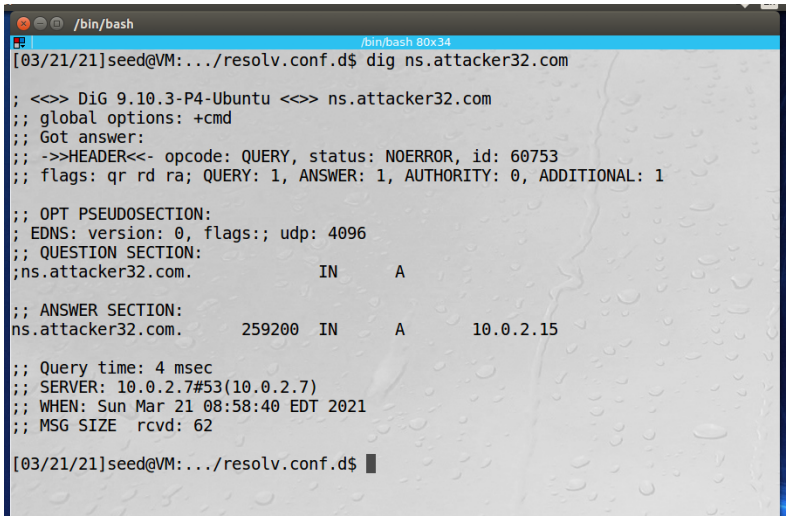

Figure 5.2. As can be seen in the first line of the answer section, it comes from the "attacker32.com.zone" file that was set up with the attacker (10.0.2.15)

Before we proceed with the attack procedure, we want to check the outcomes of running the commands `dig www.example.com` and `dig @ns.attacker32.com www.example.com`. When we do not tell it to specifically look at "ns.attacker32.com", the domain's official nameserver will be asked for answers. This can be seen in Figure 5.3. However, when we ask to specifically look at "ns.attacker32.com", the victim will ask "ns.attacker32.com" for the IP address of "example.com". This is the whole point of the attack we will be performing. We want running these commands to do the same thing.

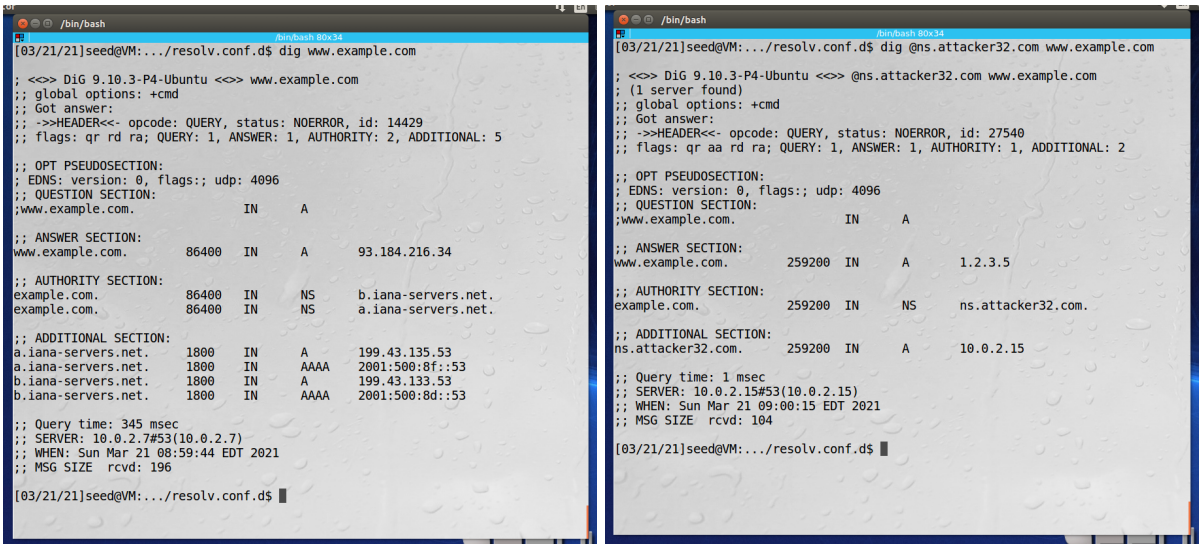

Figure 5.3. Running the 'dig' command without specifying (left) causes the domain's official nameserver to be asked for answers; Specifying '@ns.attack32.com' (right) causes 'ns.attacker32.com' to be asked for answers

Now we will proceed to run the attack. First, we will edit the `dnsrequest.py` and `spoofdnsreply.py` files. The changes we make can be seen in Figure 5.4. below. We simply change the corresponding IP addresses for our own personal network setup (Machine A - 10.0.2.6, Machine B - 10.0.2.7, and Machine C - 10.0.2.15). We use *twysw.example.com* for the name variable as it is the non-existing name in *example.com* that we wish to query DNS server Apollo for. We use *example.com* for the domain variable as that is the domain of the non-existing name we wish to query. Finally, we use *ns.attacker32.com* for the ns variable as that is the nameserver we are using to attack that will resolve the query for this non-existing name.

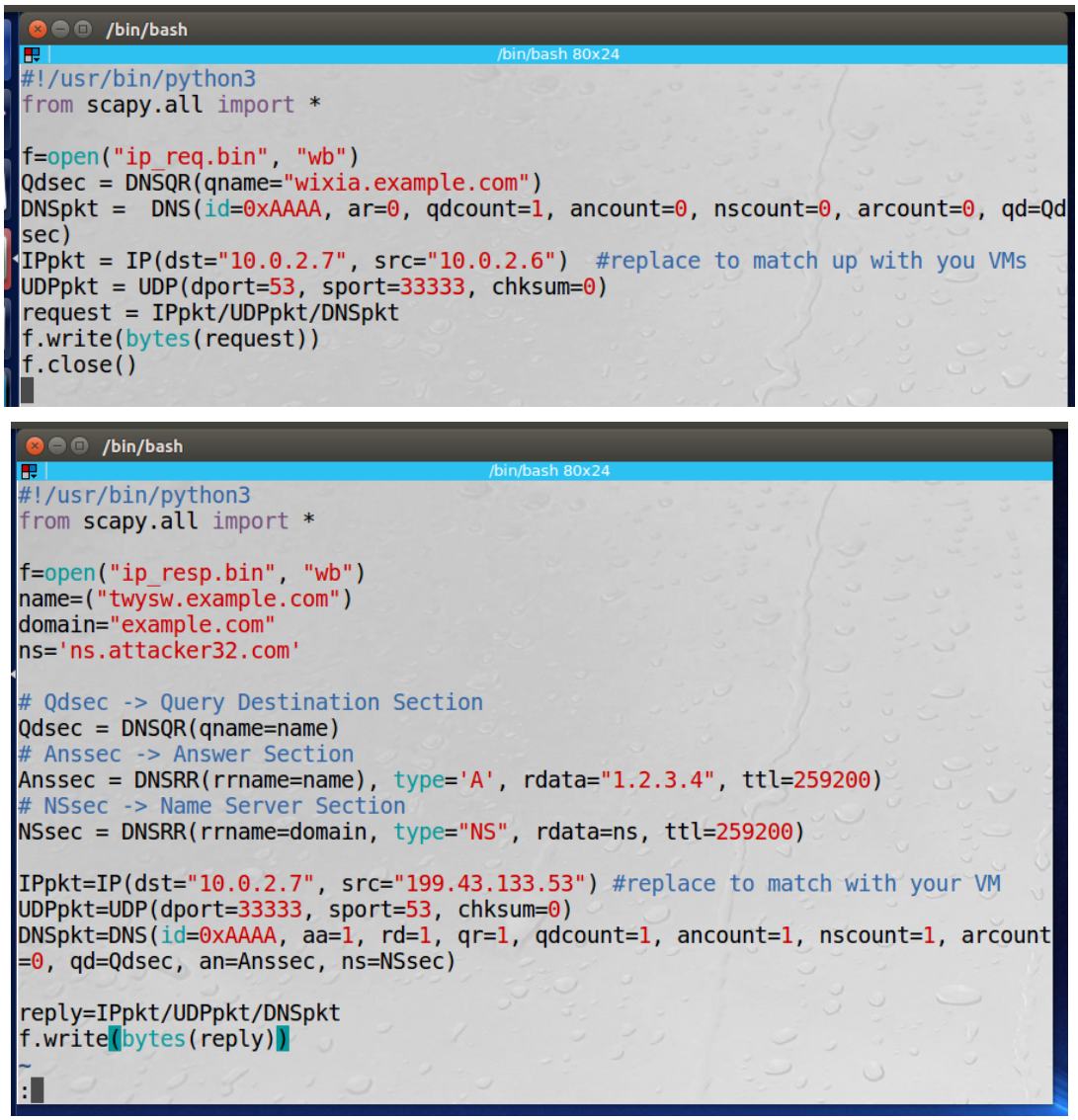

Figure 5.4. The updated 'dnsrequest.py' and 'spoofdnsreply.py' files with changes to reflect our personal network settings (IP address changes)

From here we will launch the actual Kaminsky attack using the `attack.c` file. Note that we did not have to change anything in this file. Since we are using the exact same

non-existing name (‘twysw.example.com’), the offset of the name in the binary file is the same as the values used in the lab guide. We double checked the file in bless just to be safe; however, the offsets are still 41 and 64. The only part of the `attack.c` file that was changed was the `sleep(2)` function which was changed to `sleep(0.5)` so we would not have to wait long between launching the attack and the correct answer being found.

We begin by flushing the DNS cache of Machine B and running our `dnsrequest.py` program on the attacker machine. We then run `dig b.iana-servers.net` on the user machine and receive the output in Figure 5.5. below.

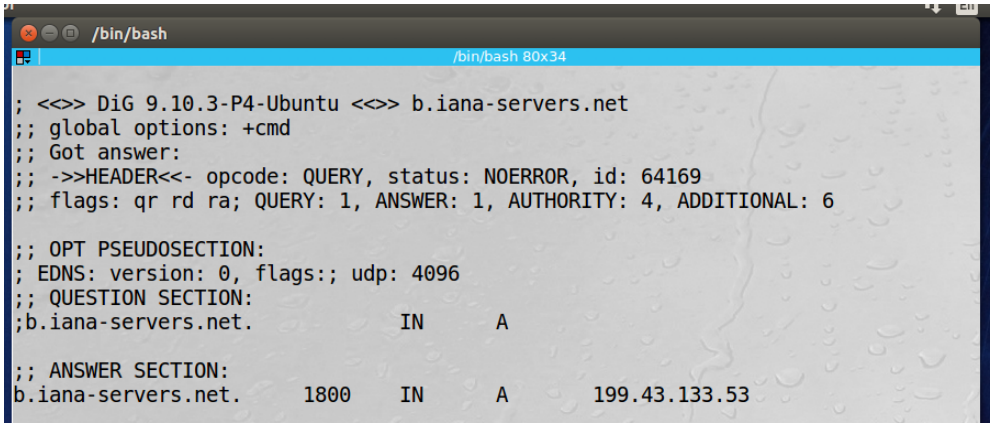

Figure 5.5. The output of running the command ‘dig b.iana-servers.net’ on the user machine; we see that the IP address is 199.43.133.53

We then proceed to run our `spoofdnsreply.py` command on the attacker machine and begin Wireshark capture of packets. After this, we compile and run the `attack.c` file. After doing this, we continuously run the command `sudo rndc dumpdb -cache; cat /var/cache/bind/dump.db | grep attacker` until `grep` returns ‘attacker’ showing up in the cache. The result of this command can be seen in Figure 5.6. below.

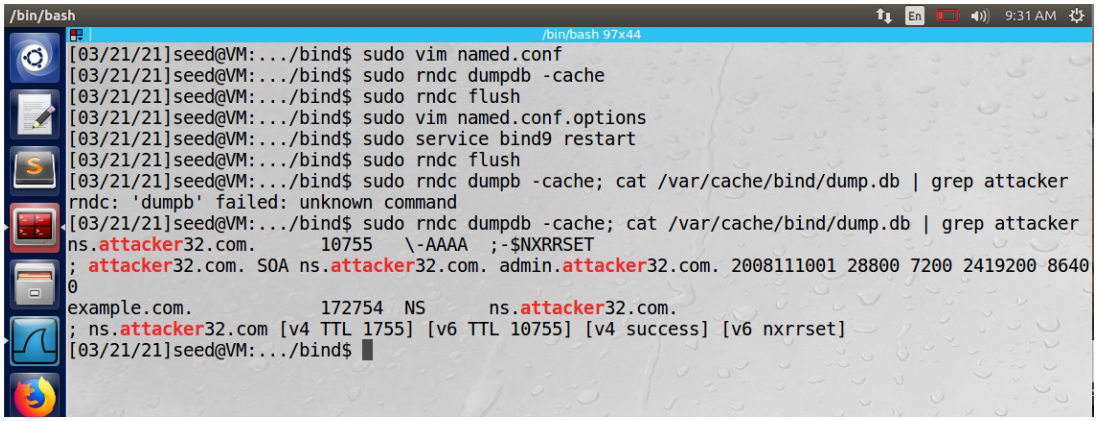

Figure 5.6. The result of ‘grep’ telling us that ‘attacker’ is showing up in the cache once we launch the attack

Once we get this result, we can stop the attack using ctrl + C, but keep the Wireshark capture running. We can see the results of the Wireshark capture in Figure 5.7. We can see the DNS query responses being sent between the server chosen (199.43.135.53) and the actual server (10.0.2.7).

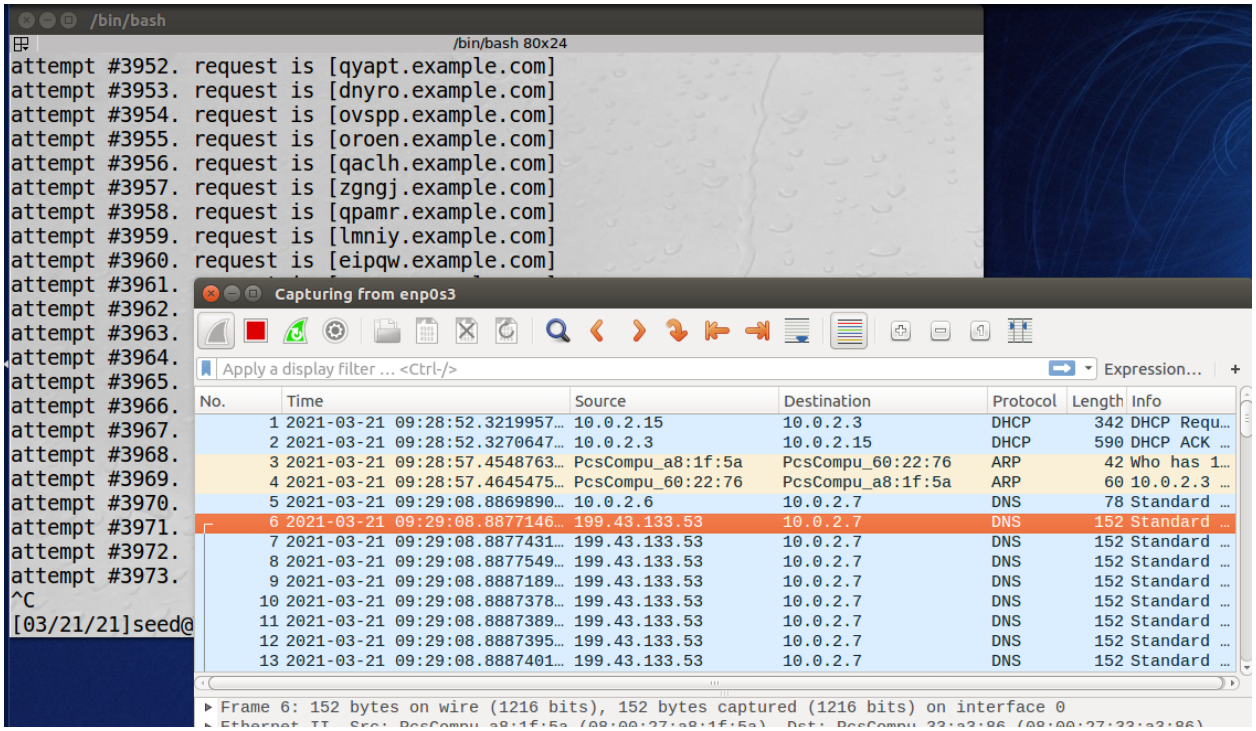

Figure 5.7. Wireshark capture of the DNS query responses being sent between the IP address of the server chosen (199.43.135.53) and the actual server (10.0.2.7)

We then go to the user machine and try our `dig` commands again. This time, both of them should produce the same results; specifically, the `dig www.example.com` command should not result in the authentic result from the domain's legitimate nameserver. We can see the results of this command in Figure 5.8. below. As we can see, the attack was successful and the fake result from the attacker shows up. We can see in the Wireshark capture the query goes from User to Server (10.0.2.6 to 10.0.2.7) and the Server to User (10.0.2.7 to 10.0.2.6).

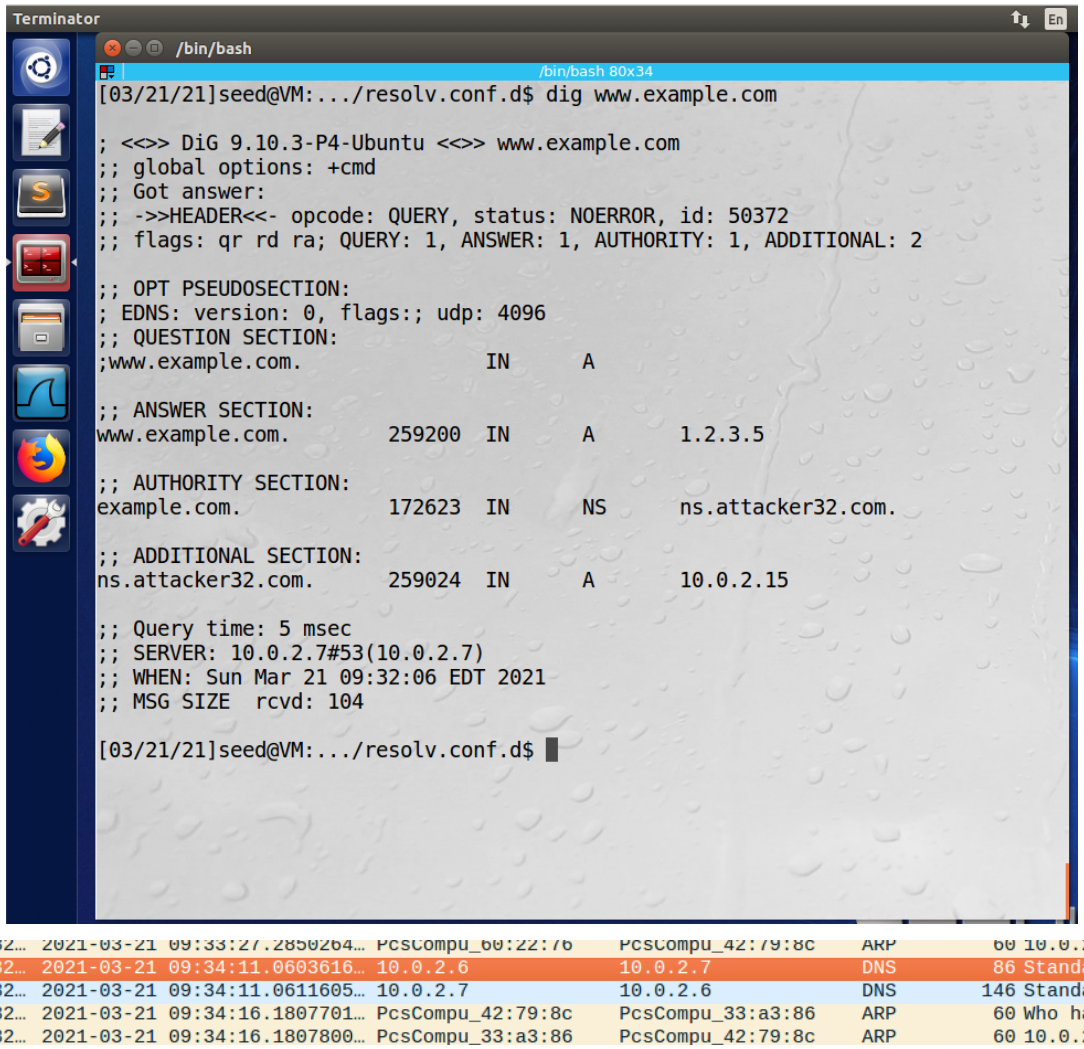

Figure 5.8. The result of using the 'dig www.example.com' command on the user machine after the attack is completed; the Wireshark packets showing the attack worked

Further, when we run the *'dig @ns.attacker32.com www.example.com'* command again, we receive the same result as the previous command. The results of this can be seen below in Figure 5.9. We also include the Wireshark capture to show how the command is working. This command causes Machine A to communicate with Machine B and then further querying Machine C to get the fake answer.

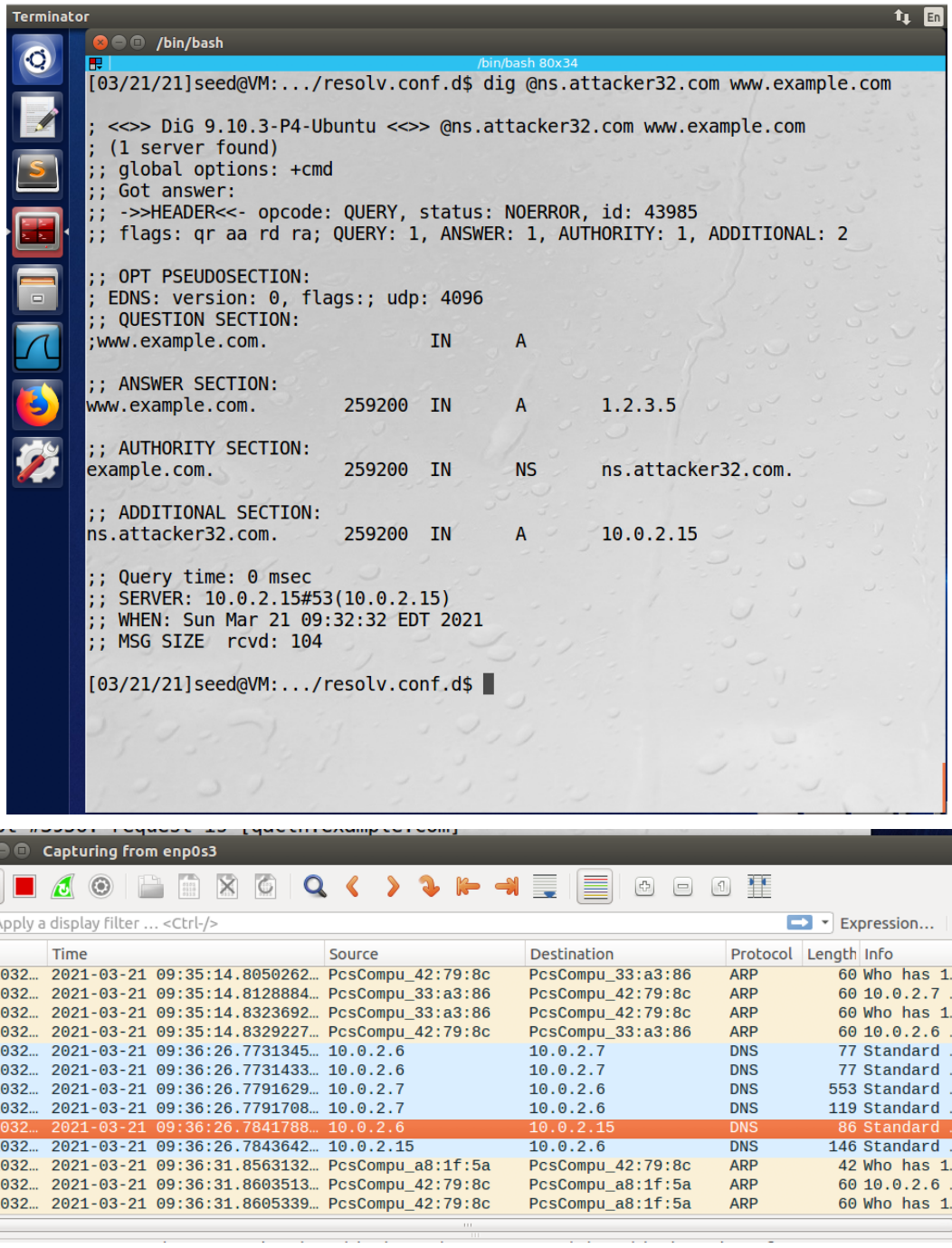

Figure 5.9. The result of running 'dig @ns.attacker23.com www.example.com'; the results of running the Wireshark capture during this command