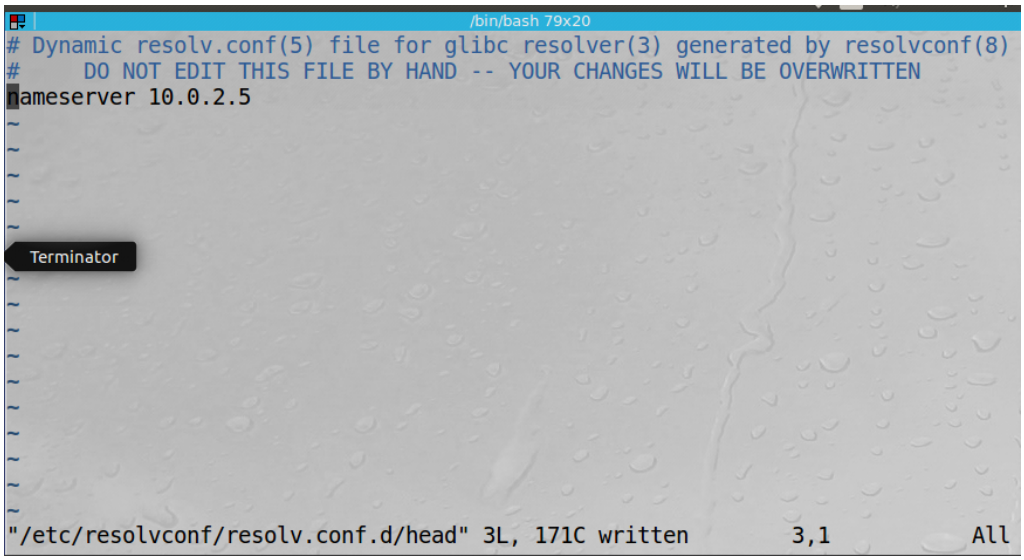
Lab 4 – Kevin Martin

VM A (10.0.2.15) will be the attacker, VM B (10.0.2.4) will be the user, and VM C (10.0.2.5) will be the server

Task 1 Using Firewall

First, to configure the user machine, I will change the the **resolver configuration** file for VM B and force it to use VM C as the local DNS server. To make sure the command takes effect, I will invoke the **resolvconf** command as well:

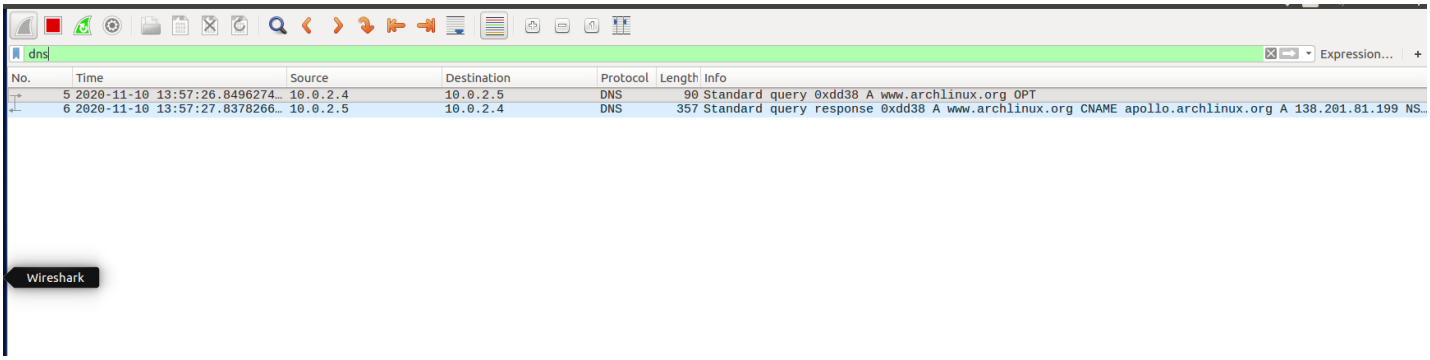


Next, I opened up Wireshark on VM B and used the **dig** command on a remote website. First, the output of the **dig** command indicating the target ip address, as well as the server used:

```
[11/10/20]seed@VM:~$ dig www.archlinux.org

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.archlinux.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56632
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 7

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

;; ANSWER SECTION:
www.archlinux.org.      86400   IN      CNAME   apollo.archlinux.org.
apollo.archlinux.org.   600     IN      A       138.201.81.199

;; AUTHORITY SECTION:
archlinux.org.          86399   IN      NS      robotns3.second-ns.com.
archlinux.org.          86399   IN      NS      ns1.first-ns.de.
archlinux.org.          86399   IN      NS      robotns2.second-ns.de.

;; ADDITIONAL SECTION:
ns1.first-ns.de.        600     IN      A       213.239.242.238
ns1.first-ns.de.        299     IN      AAAA    2a01:4f8:0:a101::a:1
robotns2.second-ns.de.  86399   IN      A       213.133.105.6
robotns2.second-ns.de.  299     IN      AAAA    2a01:4f8:d0a:2004::2
robotns3.second-ns.com. 7199    IN      A       193.47.99.3
robotns3.second-ns.com. 599     IN      AAAA    2001:67c:192c::add:a3

;; Query time: 988 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Tue Nov 10 13:57:27 EST 2020
;; MSG SIZE  rcvd: 313
```

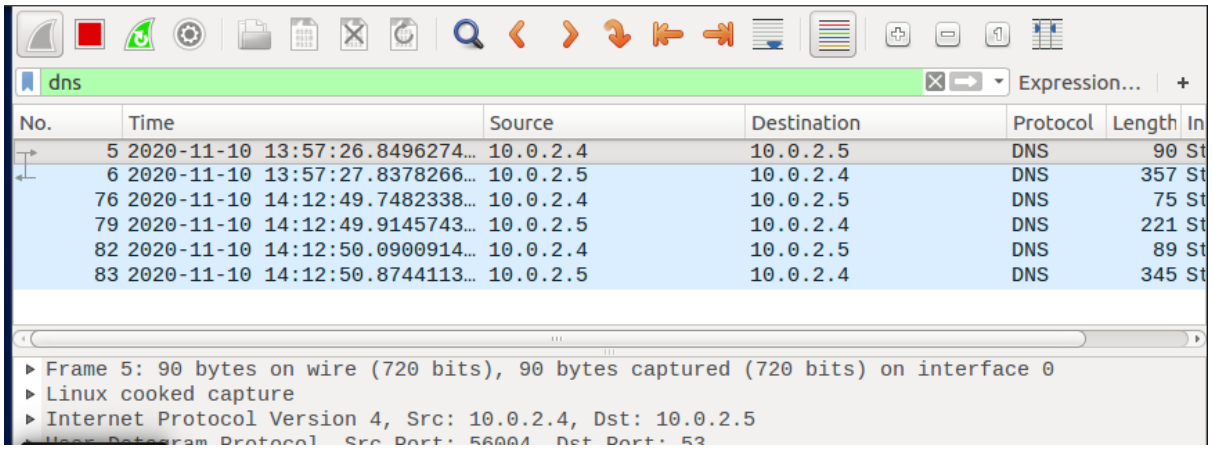
Terminator

Finally, the wireshark capture confirming this request:



I also ran a **ping** request to the same website. The interesting part is that from the terminal, it is getting the correct packets back and the ip address of the website. But wireguard is showing just the traffic between VM B and VM C:

```
[11/10/20]seed@VM:~$ ping archlinux.org
PING archlinux.org (138.201.81.199) 56(84) bytes of data.
Terminator
rom apollo.archlinux.org (138.201.81.199): icmp_seq=1 ttl=48 time=17
5 ms
64 bytes from apollo.archlinux.org (138.201.81.199): icmp_seq=2 ttl=48 time=17
0 ms
64 bytes from apollo.archlinux.org (138.201.81.199): icmp_seq=3 ttl=48 time=16
3 ms
^C
--- archlinux.org ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 163.804/169.992/175.285/4.729 ms
[11/10/20]seed@VM:~$
```

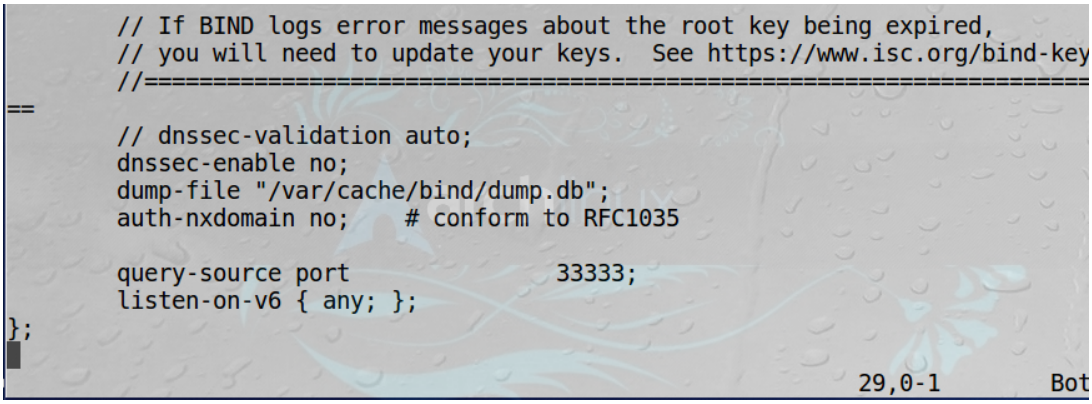


Observation: By modifying the configuration file for the user VM, we were able to force it to use another local VM as its DNS server. From the perspective of the user, there was no change in behavior as we were still able to sucessfully ping a remote server.

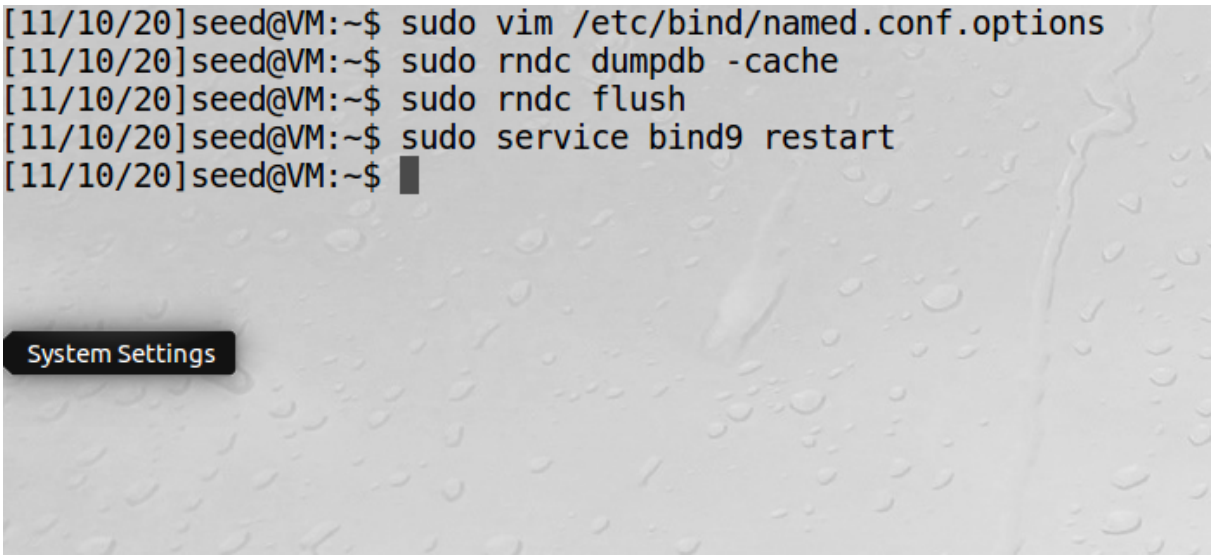
Explanation: This modification can be easily accomplished through the use of the default config files. Our local “server” will execute requests normally.

Task 2 Setup a Local DNS Server

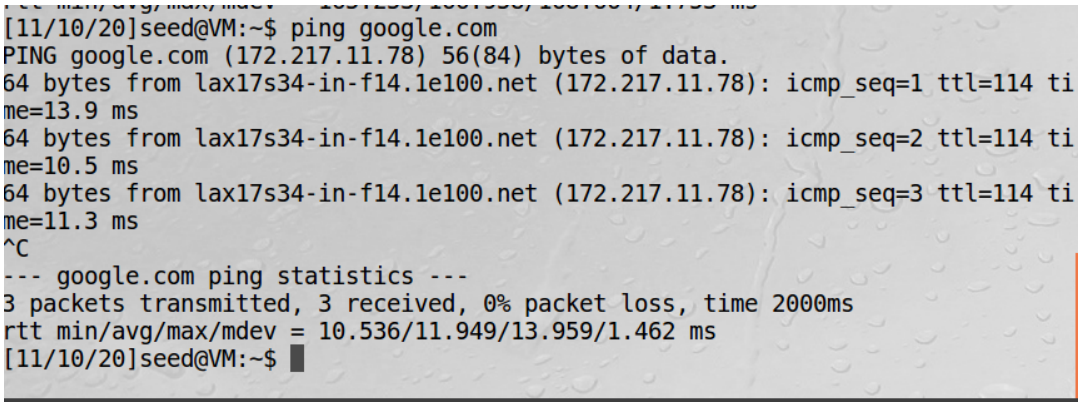
To setup VM C as the local DNS server, I first confirm that the options at **/etc/bind/named.conf.options** are configured correctly:

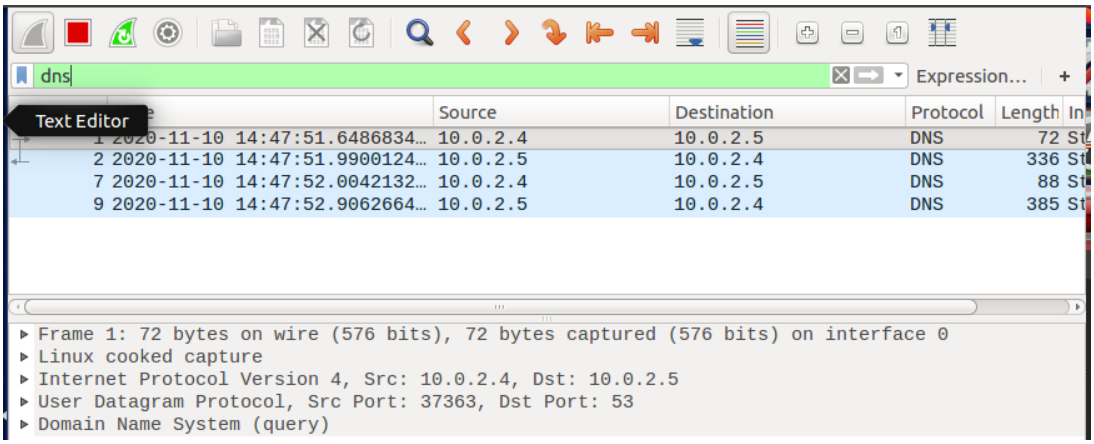


I then check to make sure the cache is properly dumped and then clear the cache. Finally, I restart the service to ensure my changes have taken effect:



I ping a different website this time from the user (VM B) and capture the packets on the user's wireshark, observing similar results. The bind9 server was already setup and running correctly from step 1, so no changes were to be expected:





Observation: We see similar results to Task 1. Here the user again uses the server VM to route its traffic. On wireshark, the packets are being transferred only from VM B to VM C. However, the output of the **ping** request in the terminal is functioning completely normal.

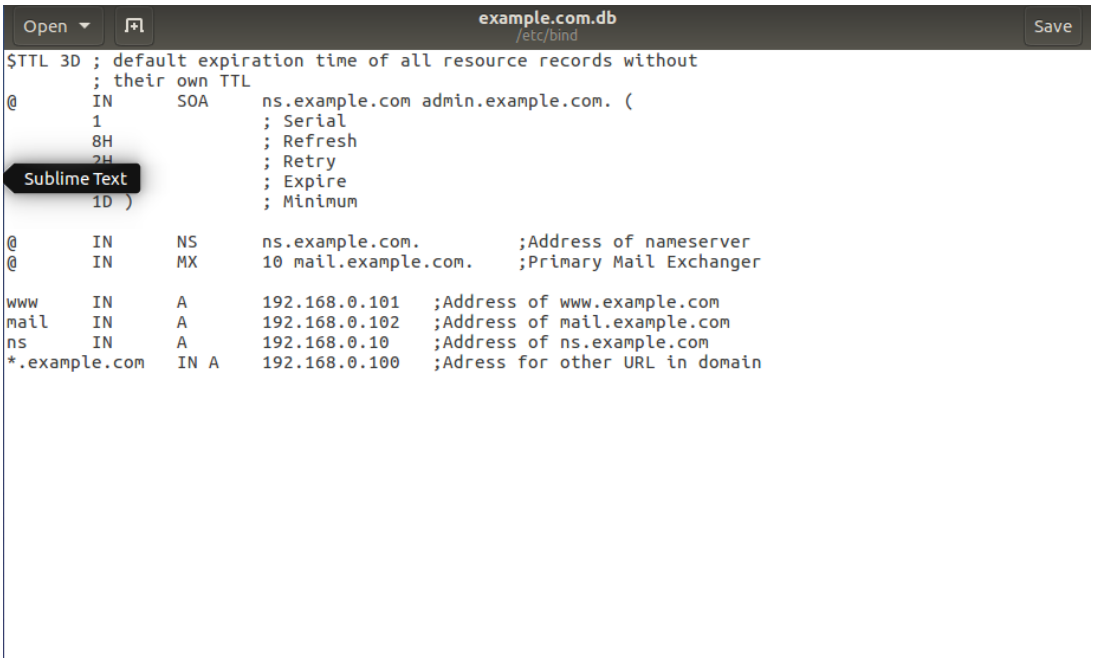
Explanation: In this case, **bind9** was already setup and running in VM C by default. It was still good to double check that the configuration was correct. As such, VM B is now routing its DNS traffic to VM C.

Task 3 Host a Zone in the Local DNS Server

To host a zone on the local DNS server (VM C), we again modify the file **/etc/bind/named.conf** to create the zones for **www.example.com**:

```
[11/10/20]seed@VM:~$ sudo vim /etc/bind/named.conf
[11/10/20]seed@VM:~$ cat /etc/bind/named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
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 "0.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/192.168.0.db";
        };

[11/10/20]seed@VM:~$
```

Next, I created two new files in the **/etc/bind/** directory: **192.168.0.db** and **example.com.db:**

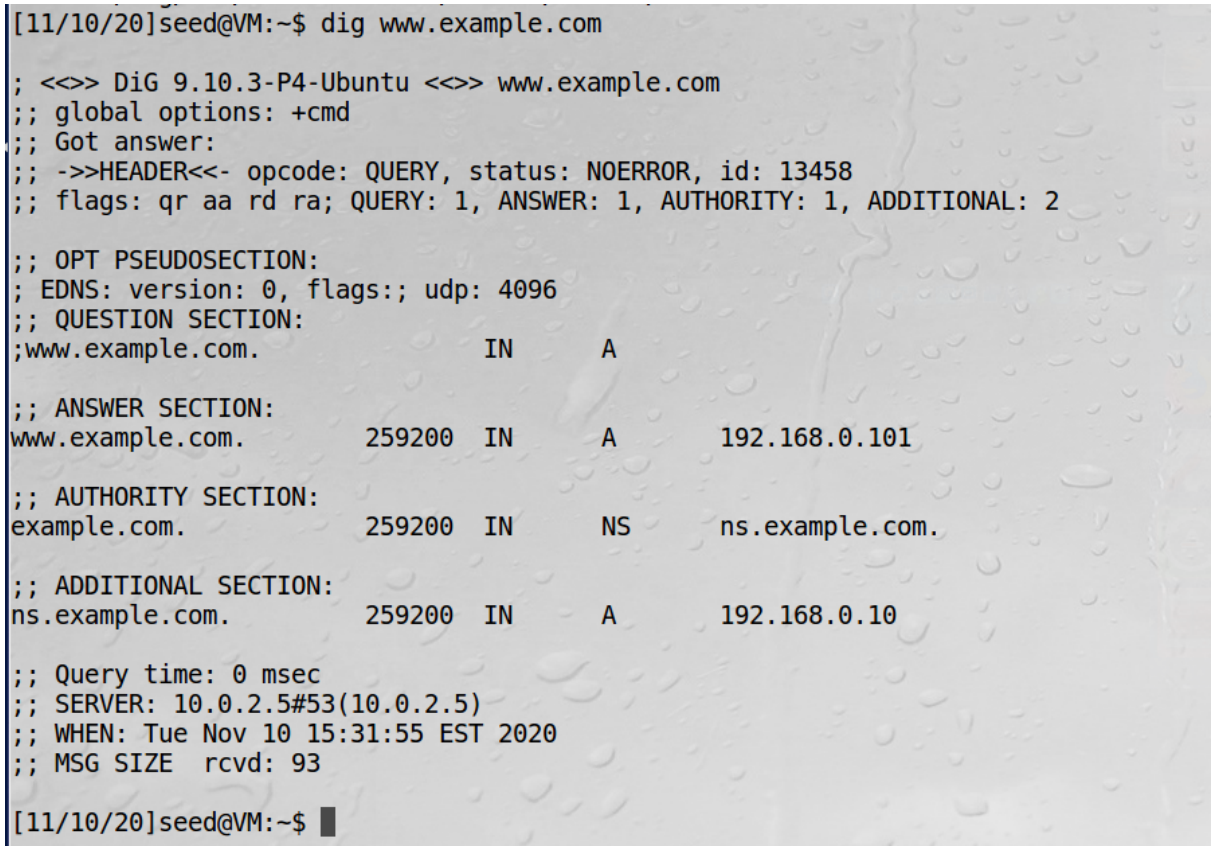



```
$TTL 3D ; default expiration time of all resource records without
        ; their own TTL
@       IN      SOA     ns.example.com admin.example.com. (
        1               ; Serial
        8H              ; Refresh
        2H              ; Retry
        4W              ; Expire
        1D )            ; Minimum

@       IN      NS      ns.example.com.         ;Address of nameserver
@       IN      MX      10 mail.example.com.    ;Primary Mail Exchanger

www     IN      A       192.168.0.101   ;Address of www.example.com
mail    IN      A       192.168.0.102   ;Address of mail.example.com
ns      IN      A       192.168.0.10    ;Address of ns.example.com
*.example.com   IN A    192.168.0.100   ;Adress for other URL in domain
```

Reverse lookup zone file:


```
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                        1
                        8H
                        2H
                        4W
                        1D)
@       IN      NS      ns.example.com.

101     IN      PTR     www.example.com.
102     IN      PTR     mail.example.com.
10      IN      PTR     ns.example.com.
```

I restart **bind9** on VM C. Then, on VM B, I run **dig** for **www.example.com**:

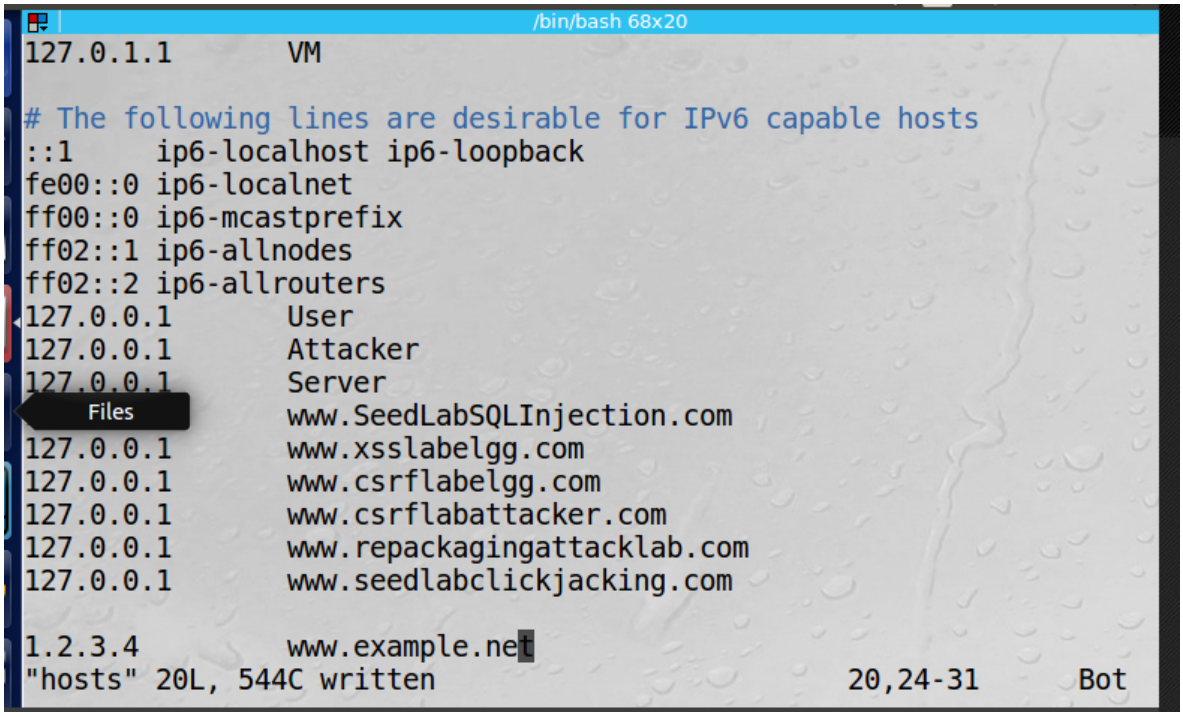


Observation: From the client, we can run the **dig** command to give us the nameserver information about www.example.com, which normally shouldn't be possible. From the perspective of VM B, this looks like a regular ip request.

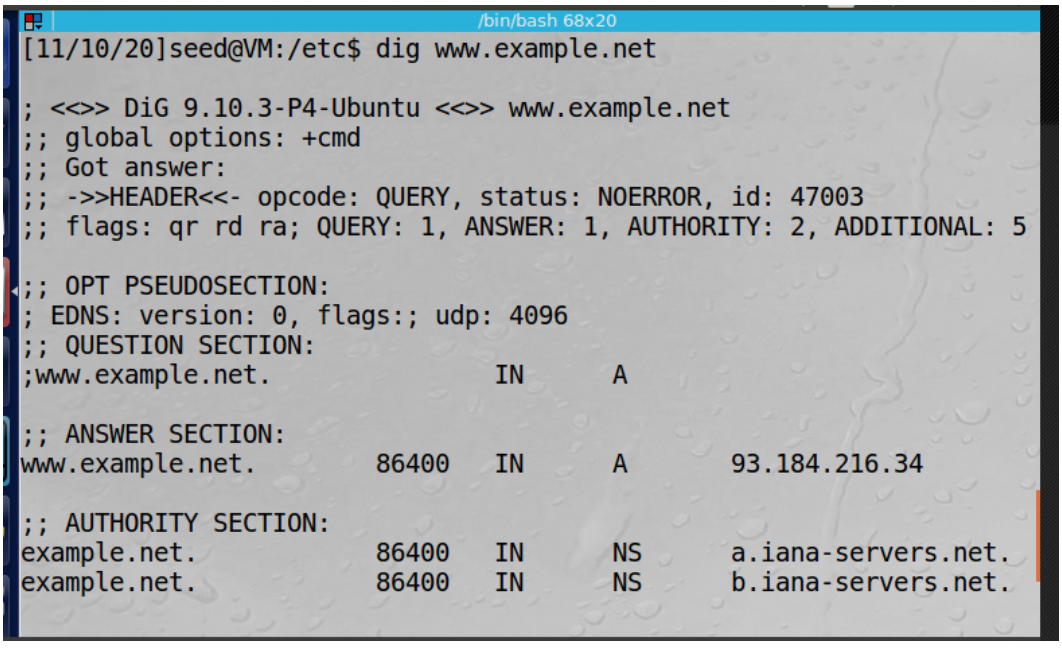
Explanation: On VM C, we successfully edited the **bind** config file and support the use of its local ip address. We have created an authoritative server for the example domain.

Task 4 Modifying the Host File

First I modify the user (VM B)'s **/etc/hosts** to include an entry for www.example.net :



Then I check to see if it executes a DNS query using a **dig** command. Note it returned a different ip address (93.184.216.34) which is not what we wanted (although expected at this point):



Next, we try a **ping** request which shows that the machine is correctly trying to ping our defined 1.2.3.4 ip address (and did **not** query the local DNS on our "server", VM C):

```
[11/10/20]seed@VM:/etc$ ping www.example.net
PING www.example.net (1.2.3.4) 56(84) bytes of data.
^C
--- www.example.net ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2024ms

[11/10/20]seed@VM:/etc$
```

Wireshark

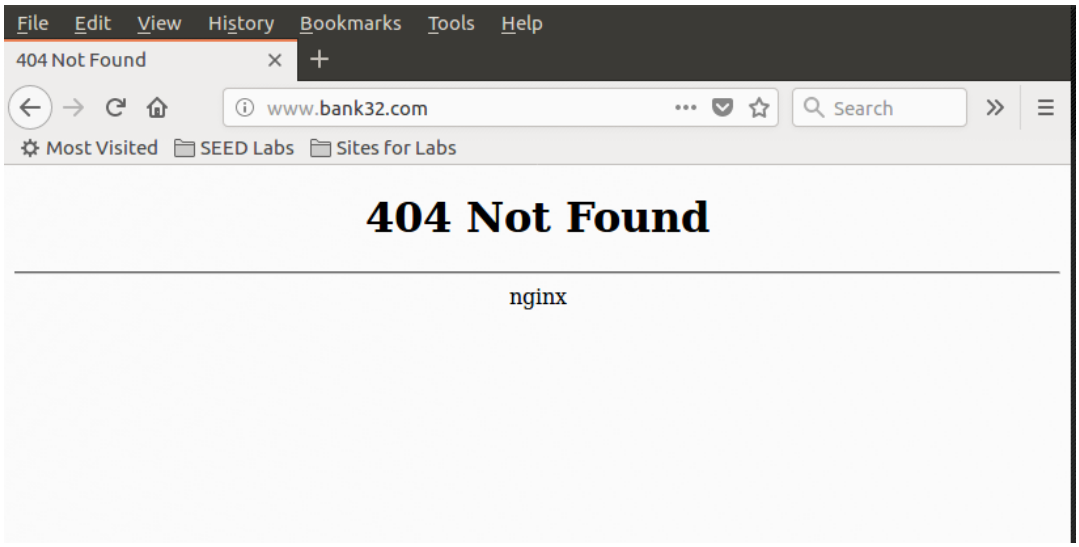
Now to simulate a compromised machine, I will add an entry in the user machine for www.bank32.com, with the ip address of another webiste, 138.201.81.199. First note how the **dig** request ignores this ip and tries to find the actual one:

```
[11/10/20]seed@VM:/etc$ dig www.bank32.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.bank32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40911
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.bank32.com.                        IN      A

;; ANSWER SECTION:
www.bank32.com.         3553    IN      CNAME   bank32.com.
bank32.com.             553     IN      A       34.102.136.180
```

However on the **ping** request, we get the expected result:

```
[11/10/20]seed@VM:/etc$ sudo vim hosts
[11/10/20]seed@VM:/etc$ ping www.bank32.com
PING www.bank32.com (138.201.81.199) 56(84) bytes of data.
64 bytes from www.bank32.com (138.201.81.199): icmp_seq=1 ttl=48 tim
e=168 ms
64 bytes from www.bank32.com (138.201.81.199): icmp_seq=2 ttl=48 tim
e=163 ms
64 bytes from www.bank32.com (138.201.81.199): icmp_seq=3 ttl=48 tim
e=162 ms
^C
--- www.bank32.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 162.569/164.835/168.752/2.800 ms
[11/10/20]seed@VM:/etc$
```

Finally, I open a web browser and try to navigate to www.bank32.com, instead of taking me to a missing page, I get the following error:

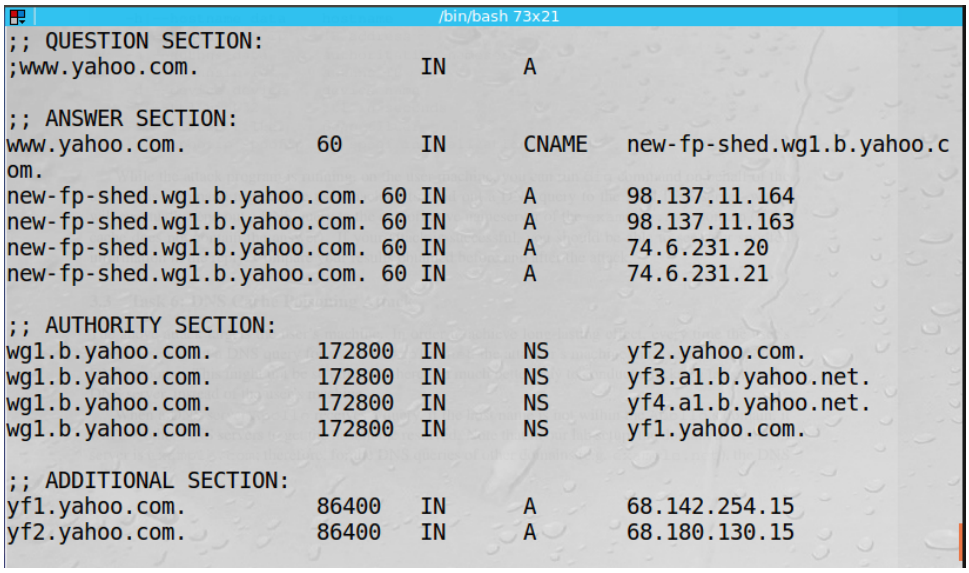


Observation: We updated the user's host file to give ip addresses for both www.example.com and www.bank32.com. In both cases, the **dig** command ignored this and sought the ip addresses externally, while the **ping** command did in fact follow the nameserver protocol.

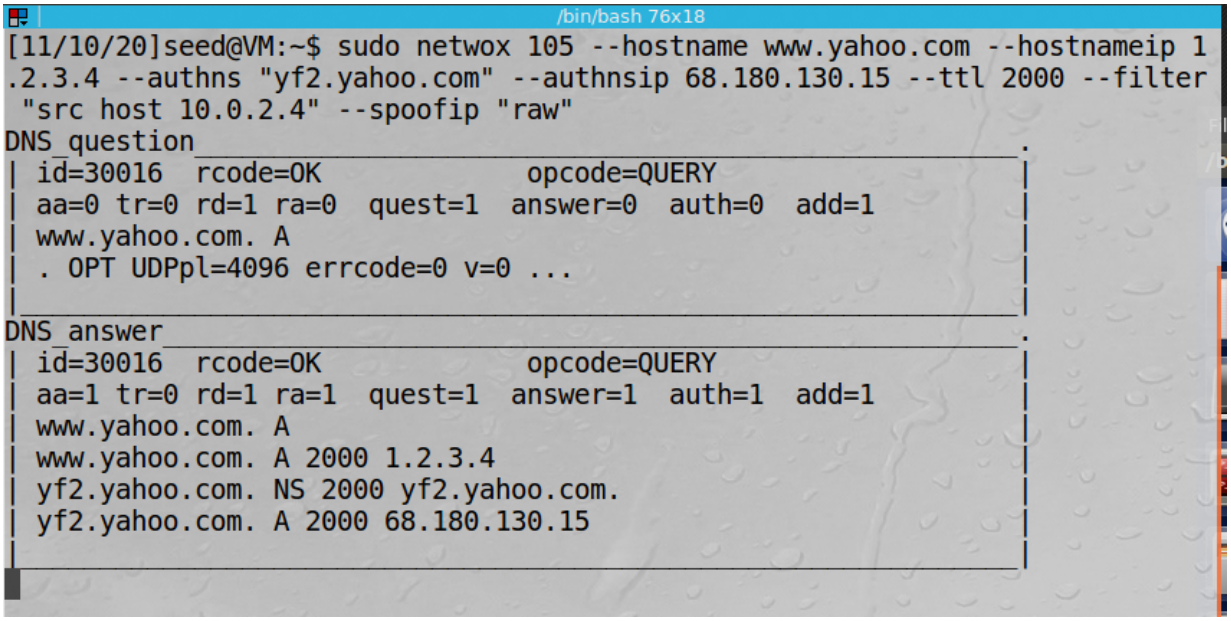
Explanation: When the host file is edited, the DNS server can be overridden and the specified domains can be redirected. It is worth noting that only those explicity specified in this file can be manipulated.

Task 5 Directly Spoofing Response to User

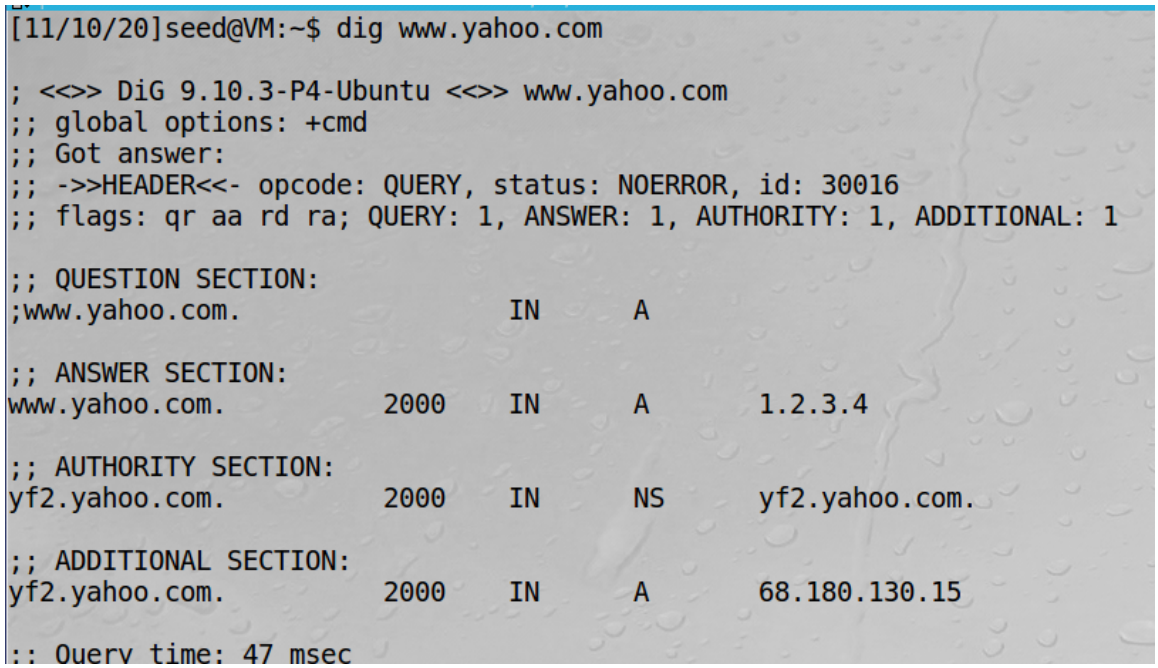
To spoof directly to the user, we must be on the same network. In this instance, we will use VM A, which up until this point has not been used, to spoof a response to VM B. To do so, we will use the **netwox 105** command on VM A and enter the relevant details from VM B's initial **dig** request. First, VM B sends a request to a website:



Next, I clear the caches and flush on all three VM's, and issue the **netwox 105** command from VM A, and the result after trying a new **dig** command from VM B:



And the resulting **dig** output from VM B. Note how we can now see our spoofed ip 1.2.3.4:

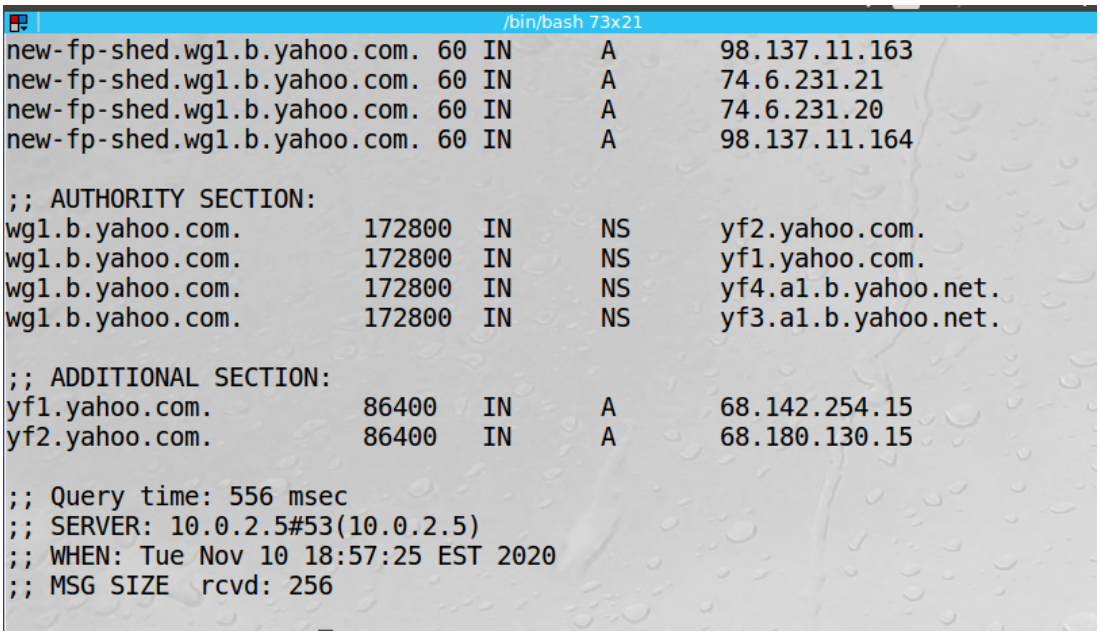


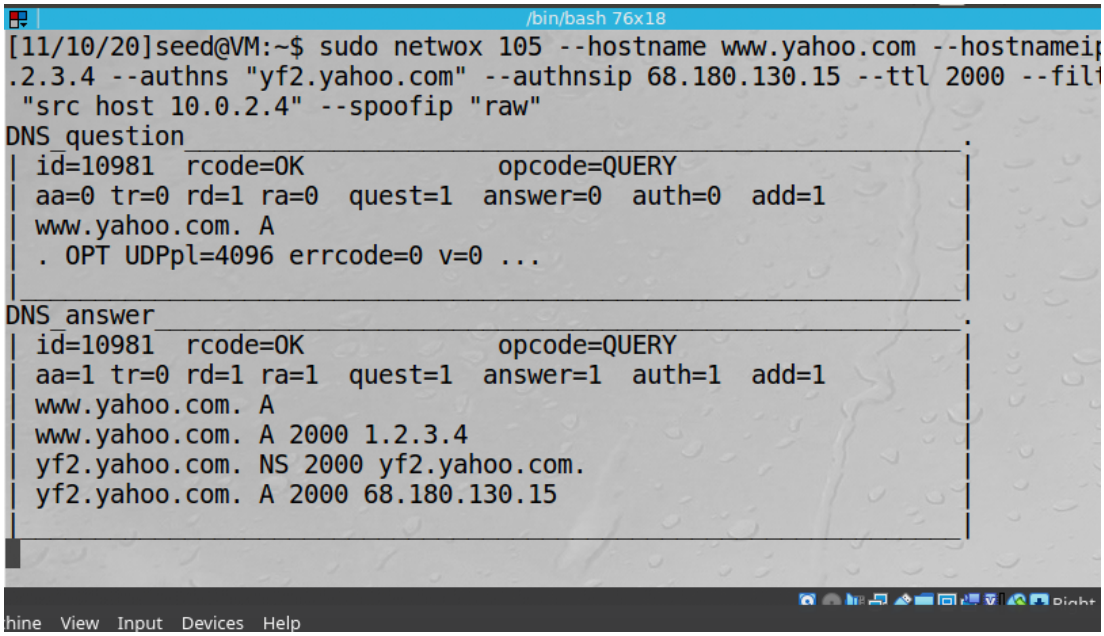
Observation: Here we use another **netwox** command, 105, to spoof a DNS query response. By gathering the important details from the client, VM B, using **dig**, we could enter those fields into the attack from VM A. VM B tries the same request only this time sees the response from a different, spoofed, ip address.

Explanation: The **netwox 105** command does an excellent job at allowing an attacker on the same network to quickly spoof this type of response. The only caveat is that the attacker would need to know the results of the initial **dig** request in order to properly execute this attack.

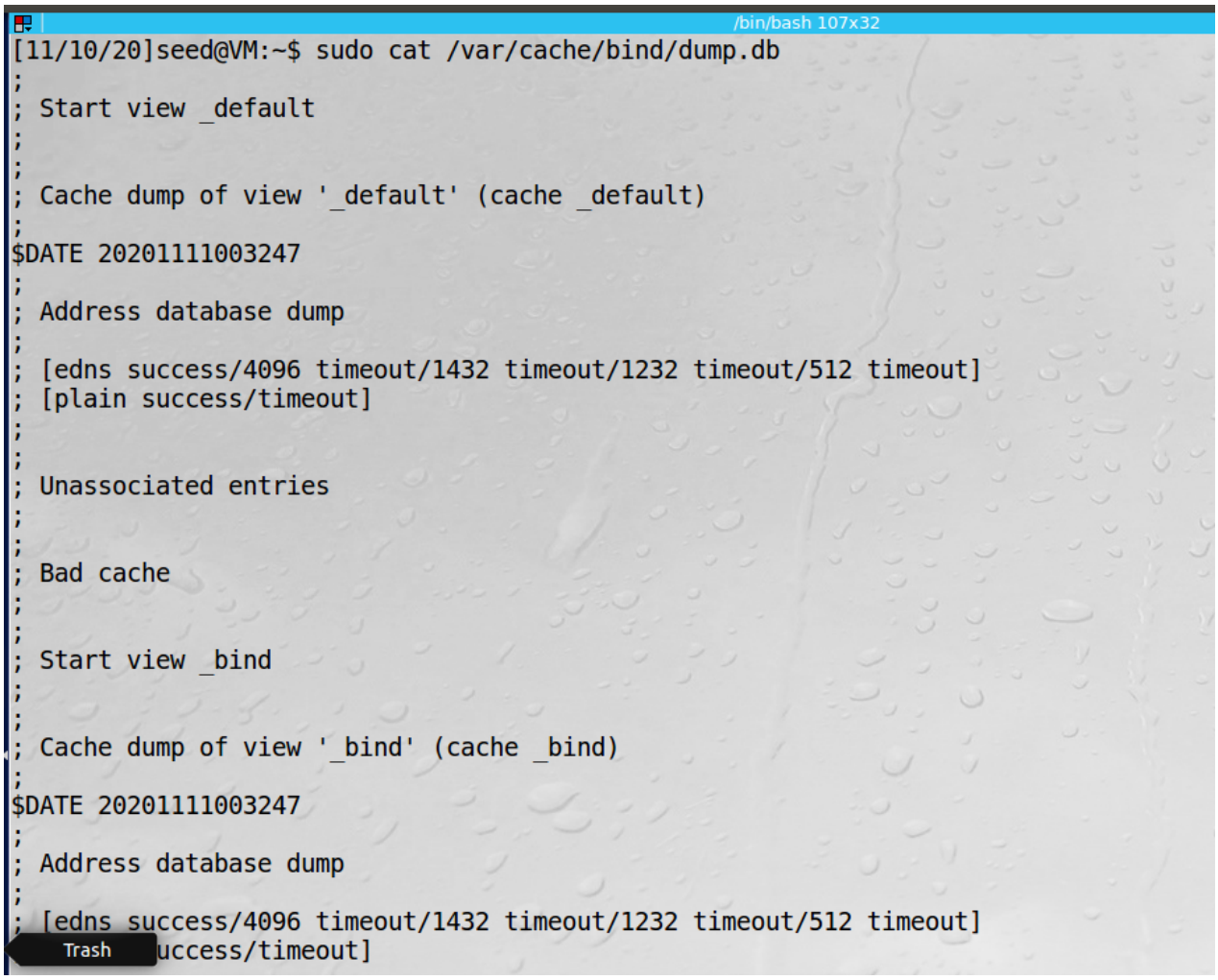
Task 6 DNS Cache Poisoning Attack

To effectively poison the DNS forces the server to continually send back a spoofed ip address, not just a one-time thing. I will clear the caches from the VM's, run the same **dig** request and issue the same **netwox 105** attack. This time, we will confirm the attack has impacted the cache by examining the cach dump on the server, VM C. First, the **dig** from VM B, and the **netwox** from VM A:





Finally, the dump file on the local server:



Unfortunately we do not see the results here, even after several flushes/clearing of the cache.

Observation: Once again we issue the same **netwox** command and observe similar results. Unfortunately the server's cache did not show the expected spoofed ip address.

Explanation: By getting the relevant results from the initial **dig** request, we can spoof the response back to the user without their knowledge. In this case, we would have hoped to see this ip in the cache of the server. In this way, the server would continually serve the incorrect/spoofed address, thus creating a persistant attack.