

Project 4: DNS Cache Poisoning

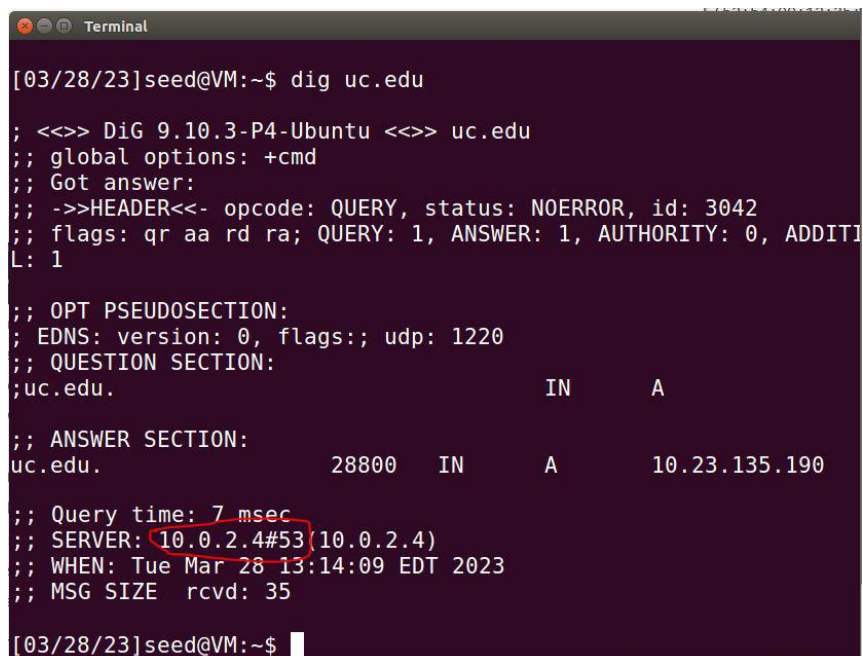
In this project, I was tasked with demonstrating a local DNS cache poisoning attack using three separate virtual machines. First, the user is at 10.0.2.15 and uses the server as a local DNS. The server is at 10.0.2.4 and is running a local DNS server. Finally, the attacker is at 10.0.2.5 and is responsible for running the spoof.py scapy program to sniff packets and send spoofed replies.

Task 1

First, on the user's machine, I used "sudo nano /etc/resolvconf/resolv.conf.d/head" and added "nameserver 10.0.2.4", which is the IP of the server.

```
[03/28/23]seed@VM:~$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by re
solvconf(8)
# DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRI
TTEN
nameserver 10.0.2.4
nameserver 127.0.1.1
search lcob.uc.edu
```

I then used "sudo resolvconf -u" to regenerate the resolv.conf file with the updated nameserver. Then I used "dig uc.edu" to make sure the response is coming from the server:



```
Terminal
[03/28/23]seed@VM:~$ dig uc.edu

; <<>> DiG 9.10.3-P4-Ubuntu <<>> uc.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3042
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITI
L: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1220
;; QUESTION SECTION:
;uc.edu.                                IN      A
;; ANSWER SECTION:
uc.edu.                                28800   IN      A      10.23.135.190

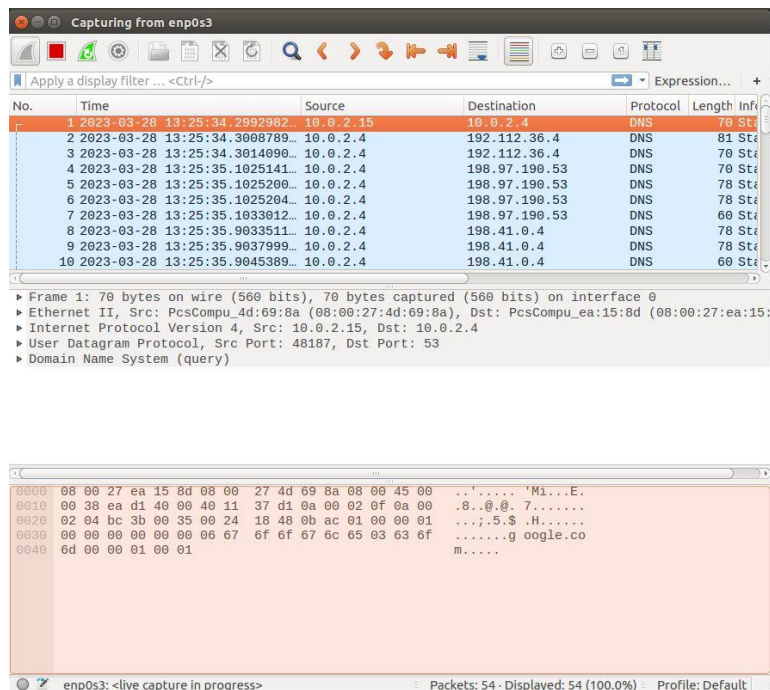
;; Query time: 7 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 28 13:14:09 EDT 2023
;; MSG SIZE rcvd: 35

[03/28/23]seed@VM:~$
```

Task 2

For task 2, I ensured bind 9 was installed on the server (10.0.2.4) using “sudo apt-get install bind9”. I then checked the options using “cat /etc/bind/named.conf.options” and made sure all the necessary options are set (dump-file, dnssec, port, etc). Next, I used “sudo service bind9 restart” to restart bind 9 with the updated options. Next, on the user’s machine (10.0.2.15), I used “ping google.com” and Wireshark to make sure it’s using the server:

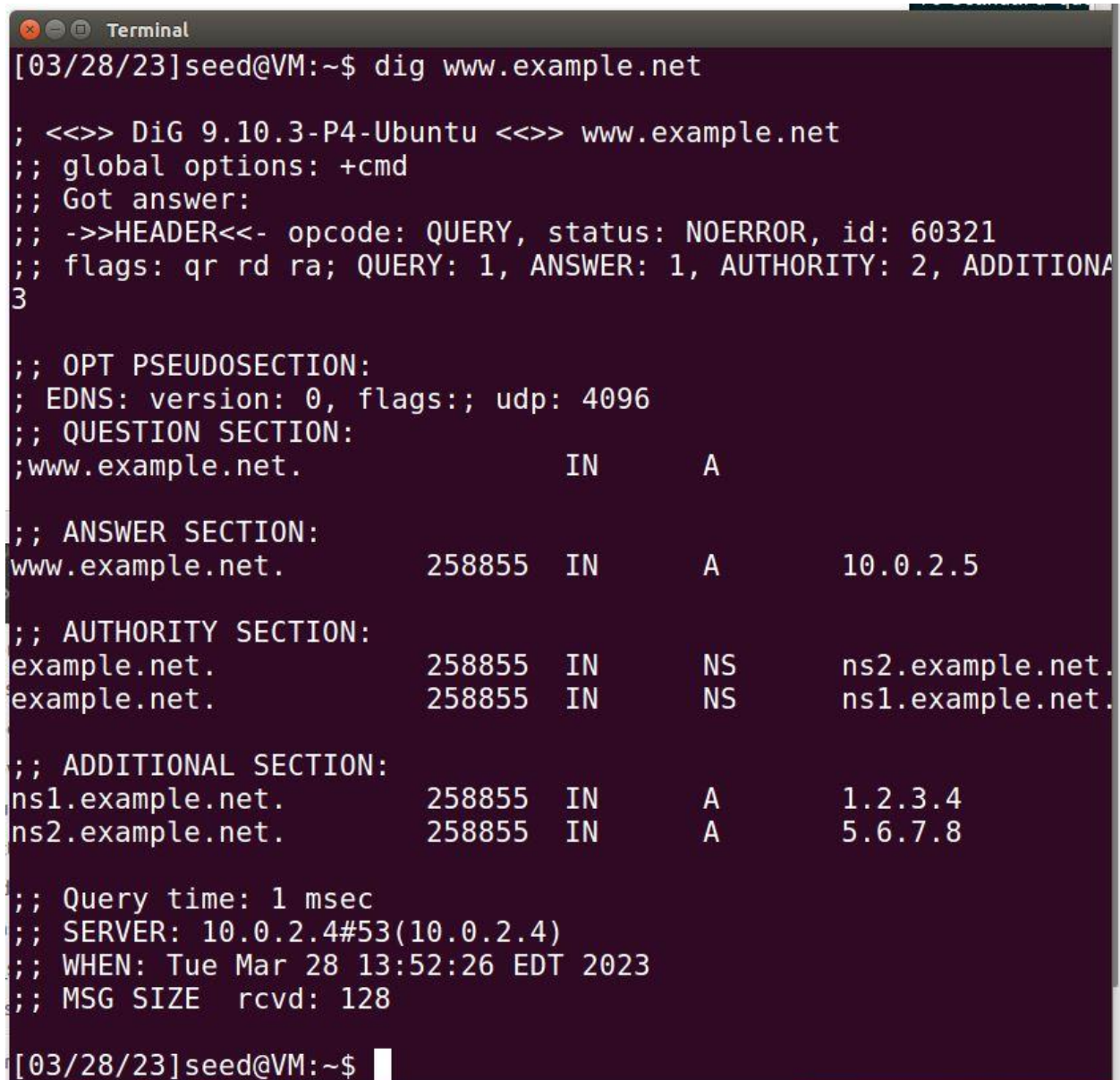
```
[03/28/23]seed@VM:~$ ping google.com
PING google.com (142.250.191.206) 56(84) bytes of data.
64 bytes from ord38s3l-in-f14.1e100.net (142.250.191.206): icmp_seq
=1 ttl=108 time=11.1 ms
64 bytes from ord38s3l-in-f14.1e100.net (142.250.191.206): icmp_seq
=2 ttl=108 time=10.8 ms
64 bytes from ord38s3l-in-f14.1e100.net (142.250.191.206): icmp_seq
=3 ttl=108 time=10.8 ms
64 bytes from ord38s3l-in-f14.1e100.net (142.250.191.206): icmp_seq
=4 ttl=108 time=9.63 ms
64 bytes from ord38s3l-in-f14.1e100.net (142.250.191.206): icmp_seq
=5 ttl=108 time=9.46 ms
64 bytes from ord38s3l-in-f14.1e100.net (142.250.191.206): icmp_seq
=6 ttl=108 time=9.83 ms
64 bytes from ord38s3l-in-f14.1e100.net (142.250.191.206): icmp_seq
```



From the above picture, we can see that the user’s machine (10.0.2.15) calls the server (10.0.2.4) with the DNS protocol immediately after pinging google.com.

Task 3

In task 3, I created a scapy program that generates a spoofed DNS reply to the server. I used the starter code included in the SEED lab and from in-class, and ran it on the attacker's machine (10.0.2.5). After running the spoof.py scapy program, I flushed DNS cache with "sudo rndc flush" and on the user's machine (10.0.2.15) I ran "dig www.example.net":



```
[03/28/23]seed@VM:~$ dig www.example.net

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60321
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

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

;; ANSWER SECTION:
www.example.net.                258855  IN      A      10.0.2.5

;; AUTHORITY SECTION:
example.net.                    258855  IN      NS      ns2.example.net.
example.net.                    258855  IN      NS      ns1.example.net.

;; ADDITIONAL SECTION:
ns1.example.net.                258855  IN      A      1.2.3.4
ns2.example.net.                258855  IN      A      5.6.7.8

;; Query time: 1 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Tue Mar 28 13:52:26 EDT 2023
;; MSG SIZE rcvd: 128

[03/28/23]seed@VM:~$
```

From the above picture on the user's machine (10.0.2.15), we can see that the attacker (10.0.2.5) successfully sniffs the packet and sends a spoofed DNS reply to the server (10.0.2.4). The

Server's DNS is poisoned and the server replies to the user with the poisoned IP for example.net. On the attacker's machine (10.0.2.5), we can see a packet was sent:

```
[03/28/23]seed@VM:~/Desktop$ sudo python spoof.py  
.  
Sent 1 packets.
```

On the server's machine, I ran “sudo rndc dumpdb -cache” and “cat /var/cache/bind/dump.db” to check the cache and make sure it was successfully poisoned:

```
[03/28/23]seed@VM:~$ sudo rndc dumpdb -cache  
[03/28/23]seed@VM:~$ cat /var/cache/bind/dump.db  
;  
; Start view _default  
;  
;  
; Cache dump of view '_default' (cache _default)  
;  
$DATE 20230328175626  
; authauthority  
example.net.                258615  IN  NS      ns1.example.net.  
                           258615  IN  NS      ns2.example.net.  
; additional  
ns1.example.net.            258615  A      1.2.3.4  
; additional  
ns2.example.net.            258615  A      5.6.7.8  
; authanswer  
www.example.net.            258615  A      10.0.2.5  
;  
; Address database dump  
;
```

In conclusion, after completing these tasks, I successfully demonstrated a local DNS cache poisoning attack using three separate virtual machines. First, I configured the user machine (10.0.2.15) to use “namespace 10.0.2.4” as the local DNS server. Next, I set up a local DNS server on the server machine (10.0.2.4) using bind 9. Last, I used scapy to create a python program to sniff out packets and generate a spoofed DNS reply and ran it on the attacker's machine (10.0.2.5).