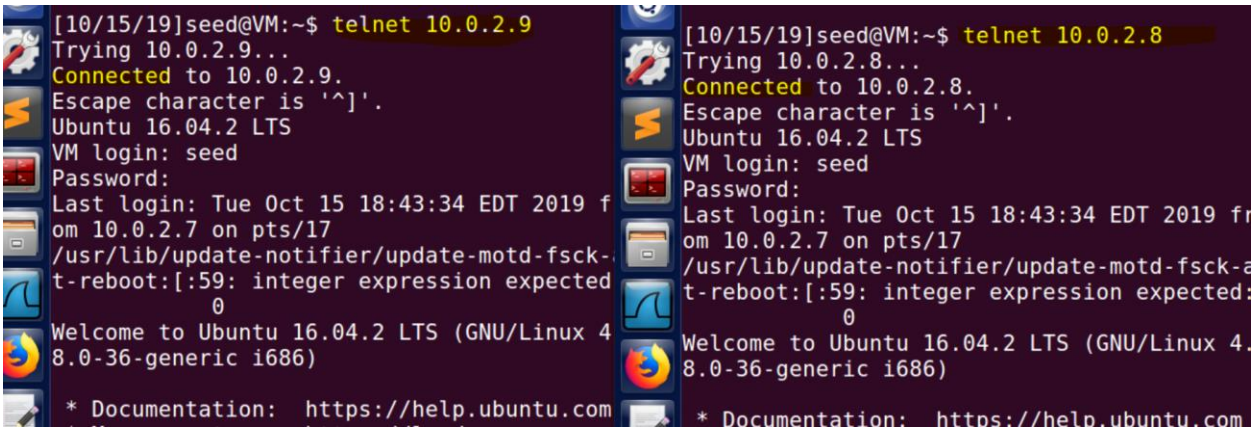
CSCI 401
Lab -12
Prof. Kadri Brogi
May 12, 2020
Emranul Hakim
23467834

Linux Firewall Exploration Lab
&
Local and Remote DNS Attack Lab

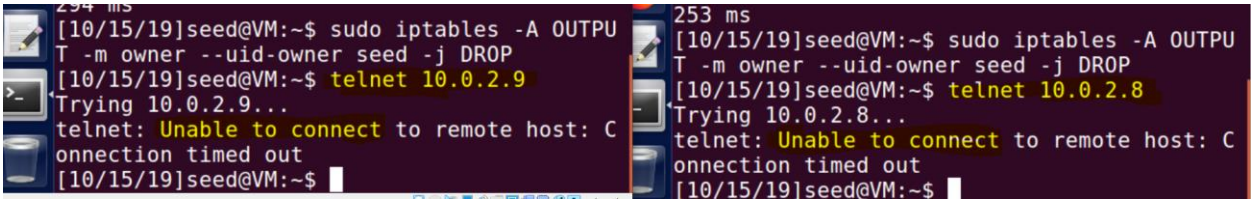
Linux Firewall Exploration Lab

2.1 Task1: Using Firewall

Telnet Works!

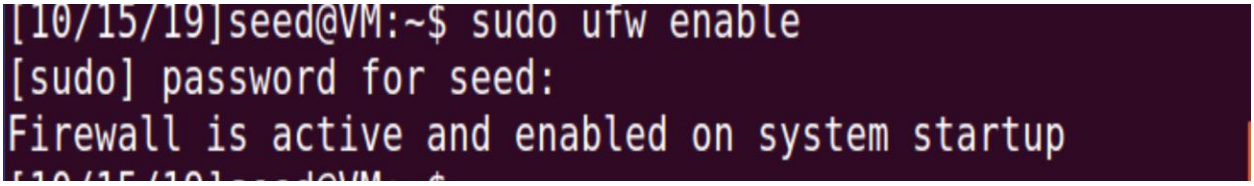


From A <->B connection is blocked both ways.



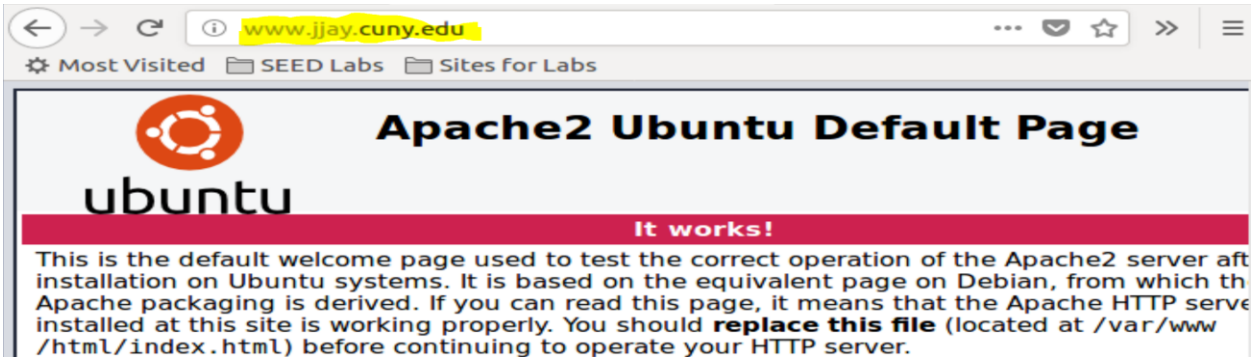
Prevent A from visiting an external web site.

Using gedit



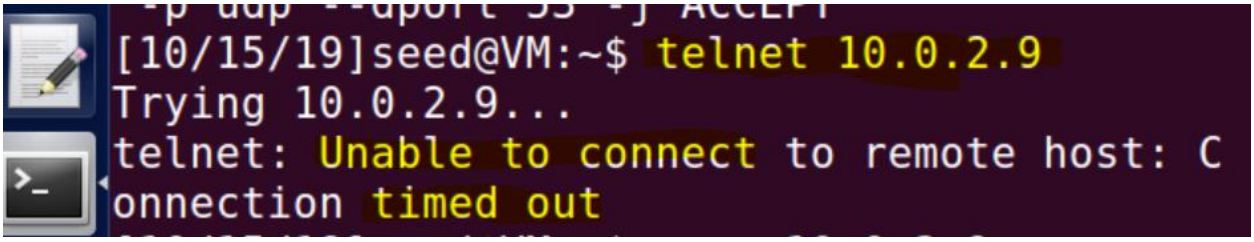
```
[10/15/19]seed@VM:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
172.217.12.142             DENY OUT    Anywhere on enp0
s3
```

```
127.0.0.1       localhost
127.0.1.1       VM
127.0.0.1  www.jjay.cuny.edu
::1 www.jjay.cuny.edu
127.0.0.1 http://www.jjay.cuny.edu
127.0.0.1  www.jjay.cuny.edu
::1 www.jjay.cuny.edu
```



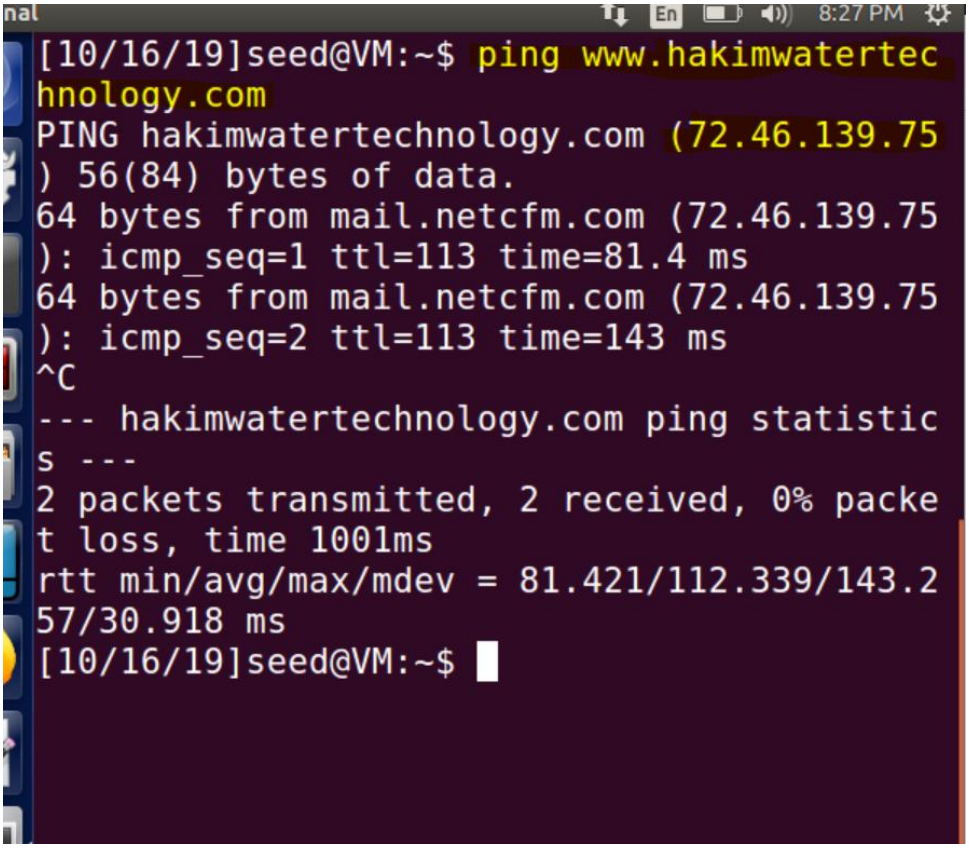
Testing our Firewall Policy

Telnet connection on port 23 is blocked



Wget connection on port 80 is Using terminal.

In this task we take a simple web page name www.hakimwatertechnology.com and ping and we were able to get the request and access the web page. Now our goal is to prevent this web page for which we deny the permission using *sudo ufw reject out to 72.46.139.75*
And we were able to block the web page from accessing as shown below:

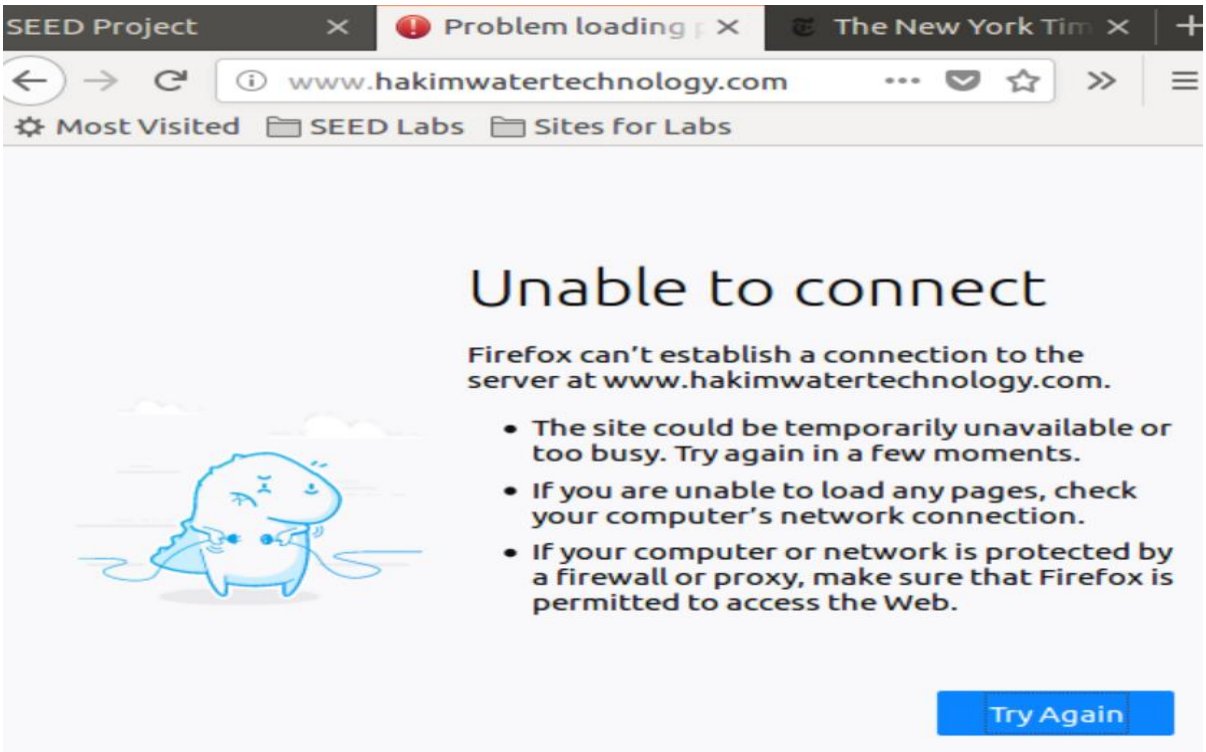


sudo ufw reject out to 72.46.139.75

```
[10/16/19]seed@VM:~$ sudo ufw reject out to 72.46.139.75
Rule added
[10/16/19]seed@VM:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
23                         DENY OUT    Anywhere
72.46.139.75               REJECT OUT  Anywhere
23 (v6)                    DENY OUT    Anywhere (v6)
```

```
[10/16/19]seed@VM:~$ ping www.hakimwatertechnology.com
PING hakimwatertechnology.com (72.46.139.75) 56(84) bytes of data.
From 10.0.2.9 icmp_seq=1 Destination Port Unreachable
From 10.0.2.9 icmp_seq=1 Destination Port Unreachable
From 10.0.2.9 icmp_seq=1 Destination Port Unreachable
From 10.0.2.9 icmp_seq=1 Destination Port Unreachable
From 10.0.2.9 icmp_seq=1 Destination Port Unreachable
From 10.0.2.9 icmp_seq=1 Destination Port Unreachable
```

```
From 10.0.2.9 icmp_seq=1 Destination Port Unreachable
From 10.0.2.9 icmp_seq=1 Destination Port Unreachable
^C

--- hakimwatertechnology.com ping statistics ---
0 packets transmitted, 0 received, +8354 errors
```



2.2 Task2: Implementing a Simple Firewall

Part 1: Loadable Kernel Module

In this task we will write a loadable kernel module for which we used the following code and compile the kernel module by using makefile:

```
#include <linux/module.h>
#include<linux/kernel.h>
#include<linux/init.h>


static int kmodule_init(void){
 printk(KERN_INFO "Initializing this module\n");
        return 0;

}

static void kmodule_exit(void){


printk(KERN_INFO "Module cleanup\n");

}


module_init(kmodule_init);
module_exit(kmodule_exit);


MODULE_LICENSE ("GPL");
```

kMod.c

```
obj-m += kMod.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules


clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Makefile

Now we execute the make command in the terminal which will change the directory and change back when finished.

```
[10/17/19]seed@VM:~/.../Lab6$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/see
d/Desktop/Lab6 modules
make[1]: Entering directory '/usr/src/linux-headers-4.8
.0-36-generic'
  CC [M]  /home/seed/Desktop/Lab6/kMod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Desktop/Lab6/kMod.mod.o
  LD [M]  /home/seed/Desktop/Lab6/kMod.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.
0-36-generic'
```

After the completion of building required module we install the kernel Modules, list the kernel module as shown below:

```
[10/17/19]seed@VM:~/.../Lab6$ sudo insmod kMod.ko
[10/17/19]seed@VM:~/.../Lab6$ lsmod | grep kMod
kMod                   16384  0
```

Finally, to verify that our module has been executed successfully we run the *dmesg* command and we are able to find the message printed by our module as highlighted below:

```
[39412.514463] Initializing this module
[41422.849579] Module cleanup
[10/17/19]seed@VM:~/.../Lab6$
```

Part 2: Net filter

In this task we will basically we will write a hardcore program and run in the kernel module which we used to block the telnet from machine A to machine B and vice versa versa.
The below code is used netfilter hooks. Where the function telnetFilter inspect the TCP header and check weather the destination port number is 23 or not as shown below:

```c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>

static struct nf_hook_ops telnetFilterHook;

unsigned int telnetFilter(void *priv, struct sk_buff *skb,
                const struct nf_hook_state *state)
{
  struct iphdr *iph;
  struct tcphdr *tcph;

  iph = ip_hdr(skb);
  tcph = (void *)iph+iph->ihl*4;

  if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)) {
    printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
        ((unsigned char *)&iph->daddr)[0],
        ((unsigned char *)&iph->daddr)[1],
        ((unsigned char *)&iph->daddr)[2],
        ((unsigned char *)&iph->daddr)[3]);
    return NF_DROP;
  } else {
    return NF_ACCEPT;
  }
}
```

```
    return NF_ACCEPT;
  }
}


int setUpFilter(void) {
        printk(KERN_INFO "Registering a Telnet filter.\n");
        telnetFilterHook.hook = telnetFilter;
        telnetFilterHook.hooknum = NF_INET_POST_ROUTING;
        telnetFilterHook.pf = PF_INET;
        telnetFilterHook.priority = NF_IP_PRI_FIRST;

        // Register the hook.
        nf_register_hook(&telnetFilterHook);
        return 0;
}

void removeFilter(void) {
        printk(KERN_INFO "Telnet filter is being removed.\n");
        nf_unregister_hook(&telnetFilterHook);
}

module_init(setUpFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

We also create makefile for the program and compile it in a kernel module so that we can initialize in our kernel module and we execute the module using terminal as shown below: And we were successful to block the telnet from machine A from machine B and finally we remove the module that we have initialized in the kernel and we were able to go to the initial phase i.e we were able to do telnet from Machine A from Machine B as shown below:

```
[10/17/19]seed@VM:~/Documents$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/see
d/Documents modules
make[1]: Entering directory '/usr/src/linux-headers-4.8
.0-36-generic'
  CC [M]  /home/seed/Documents/telnetFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Documents/telnetFilter.mod.o
  LD [M]  /home/seed/Documents/telnetFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.
0-36-generic'
```

```
[10/17/19]seed@VM:~/Documents$ sudo insmod telnetFilter
.ko
[10/17/19]seed@VM:~/Documents$ lsmod| grep telnetFilter
telnetFilter           16384  0
[10/17/19]seed@VM:~/Documents$ dmesg
[    0.000000] Linux version 4.8.0-36-generic (buildd@l
gw01-13) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubu
ntu1~16.04.4) ) #36~16.04.1-Ubuntu SMP Sun Feb 5 09:39:
41 UTC 2017 (Ubuntu 4.8.0-36.36~16.04.1-generic 4.8.11)
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   NSC Geode by NSC
[    0.000000]   Cyrix CyrixInstead
[    0.000000]   Centaur CentaurHauls
[    0.000000]   Transmeta GenuineTMx86
[    0.000000]   Transmeta TransmetaCPU
[    0.000000]   UMC UMC UMC UMC
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001:
```

```
[   14.781788] [UFW BLOCK] IN=enp0s3 OUT= MAC= SRC=fe80
:0000:0000:0000:738a:b1c9:0e6b:414d DST=ff02:0000:0000:
0000:0000:0000:0000:0001 LEN=64 TC=0 HOPLIMIT=1 FLOWLBL
=266484 PROTO=UDP SPT=8612 DPT=8612 LEN=24
[   14.781871] [UFW BLOCK] IN=enp0s3 OUT= MAC= SRC=fe80
:0000:0000:0000:738a:b1c9:0e6b:414d DST=ff02:0000:0000:
0000:0000:0000:0000:0001 LEN=64 TC=0 HOPLIMIT=1 FLOWLBL
=305709 PROTO=UDP SPT=8612 DPT=8610 LEN=24
[   14.792218] [UFW BLOCK] IN=enp0s3 OUT= MAC= SRC=fe80
:0000:0000:0000:738a:b1c9:0e6b:414d DST=ff02:0000:0000:
0000:0000:0000:0000:0001 LEN=64 TC=0 HOPLIMIT=1 FLOWLBL
=266484 PROTO=UDP SPT=8612 DPT=8612 LEN=24
[   14.792280] [UFW BLOCK] IN=enp0s3 OUT= MAC= SRC=fe80
:0000:0000:0000:738a:b1c9:0e6b:414d DST=ff02:0000:0000:
0000:0000:0000:0000:0001 LEN=64 TC=0 HOPLIMIT=1 FLOWLBL
=305709 PROTO=UDP SPT=8612 DPT=8610 LEN=24
[  788.391285] Registering a Telnet filter.
```

```
TCP SPT=40232 DPT=23 WINDOW=29200 RES=0x00 SYN URGP=0
  627.197940] hrtimer: interrupt took 6391360 ns
23315.419419] Registering a Telnet filter.
23353.299709] Dropping telnet packet to 10.0.2.6
23354.315849] Dropping telnet packet to 10.0.2.6
23356.331394] Dropping telnet packet to 10.0.2.6
23360.556055] Dropping telnet packet to 10.0.2.6
23368.748170] Dropping telnet packet to 10.0.2.6
23384.875874] Dropping telnet packet to 10.0.2.6
23418.923630] Dropping telnet packet to 10.0.2.6
[10/17/19]seed@VM:~/.../Lab6$
```

```
[10/18/19]seed@VM:~/.../Lab6$ telnet 10.0.2.6
Trying 10.0.2.6...
telnet: Unable to connect to remote host: Connection ti
med out
[10/18/19]seed@VM:~/.../Lab6$
```

```
[10/18/19]seed@VM:~/.../Lab6$ sudo rmmod telnetFilter
[10/18/19]seed@VM:~/.../Lab6$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login:
```

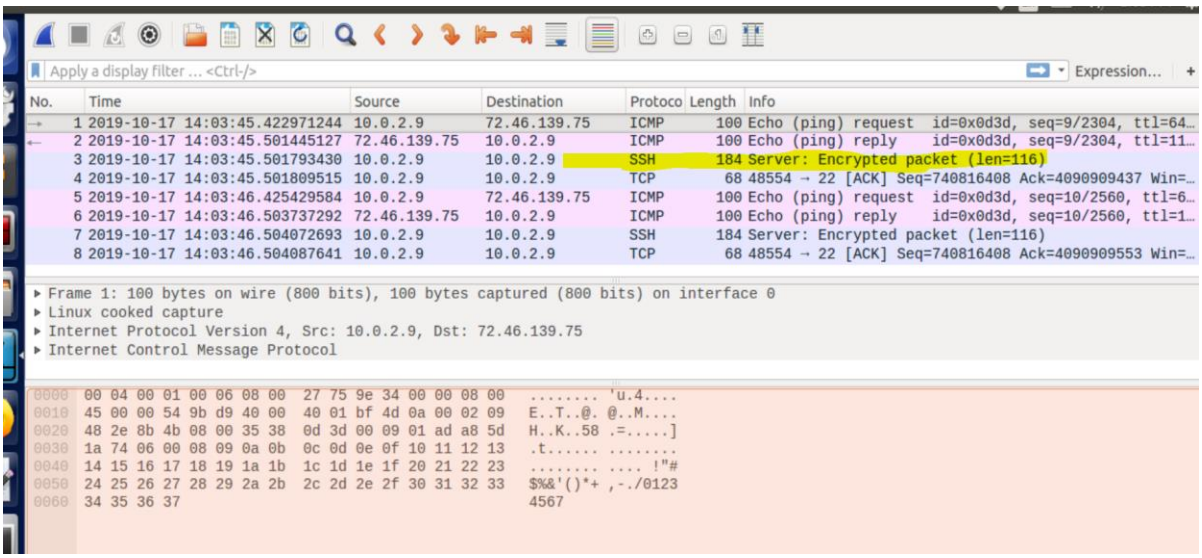
2.3 Task3: Evading Egress Filtering

Task 3.a: Telnet to Machine B through the firewall

```
[10/17/19]seed@VM:~$ ssh -L 8000:10.0.2.8:23 seed@10.0.2.9
The authenticity of host '10.0.2.9 (10.0.2.9)' can't be established.
ECDSA key fingerprint is SHA256:p1zAio6c1bI+8HDp5xa+eKRi561aFDaPE1/xq1eYzCI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.9' (ECDSA) to the list of known hosts.
seed@10.0.2.9's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

Last login: Thu Oct 17 13:48:04 2019 from 10.0.2.9
```

Since we are using two machines to perform telnet to machine B through the firewall, we use this: ssh -L 8000:10.0.2.8:23 seed@10.0.2.9 command to establish an SSH tunnel. So, if we point the machine to localhost 8000, it gets forwarded to 10.0.2.9:23 which is the telnet port of second machine through after going to 10.0.2.8:22 which is the SSH port of first machine. This is known as port forwarding.

```
[10/17/19]seed@VM:~$ telnet localhost 8000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Oct 17 16:08:06 EDT 2019 from 10.0.2.8 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.
```



Afterwards, to verify that we have created a secure channel or not, we use wireshark to monitor all communication between those two ports are encrypted or not. And we successfully found out that all communication remains encrypted.

Task 3.b: Connect to Facebook using SSH Tunnel

In order to establish a new tunnel between our localhost:port and machine B, we need to use the following command : ssh -D 9000 seed@10.0.2.9 and after that from the advance network settings we have to change the connection settings into Manual proxy configuration where we are going to manually input 127.0.0.1 as our SOCKS Host and the port number will be 9000. This will help us to connect and use the Facebook through localhost.

```
[10/17/19]seed@VM:~$ ssh -D 9000 seed@10.0
.2.9
seed@10.0.2.9's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4
.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canon
ical.com
 * Support:        https://ubuntu.com/adva
ntage

1 package can be updated.
0 updates are security updates.

Last login: Thu Oct 17 16:36:04 2019 from
localhost
```

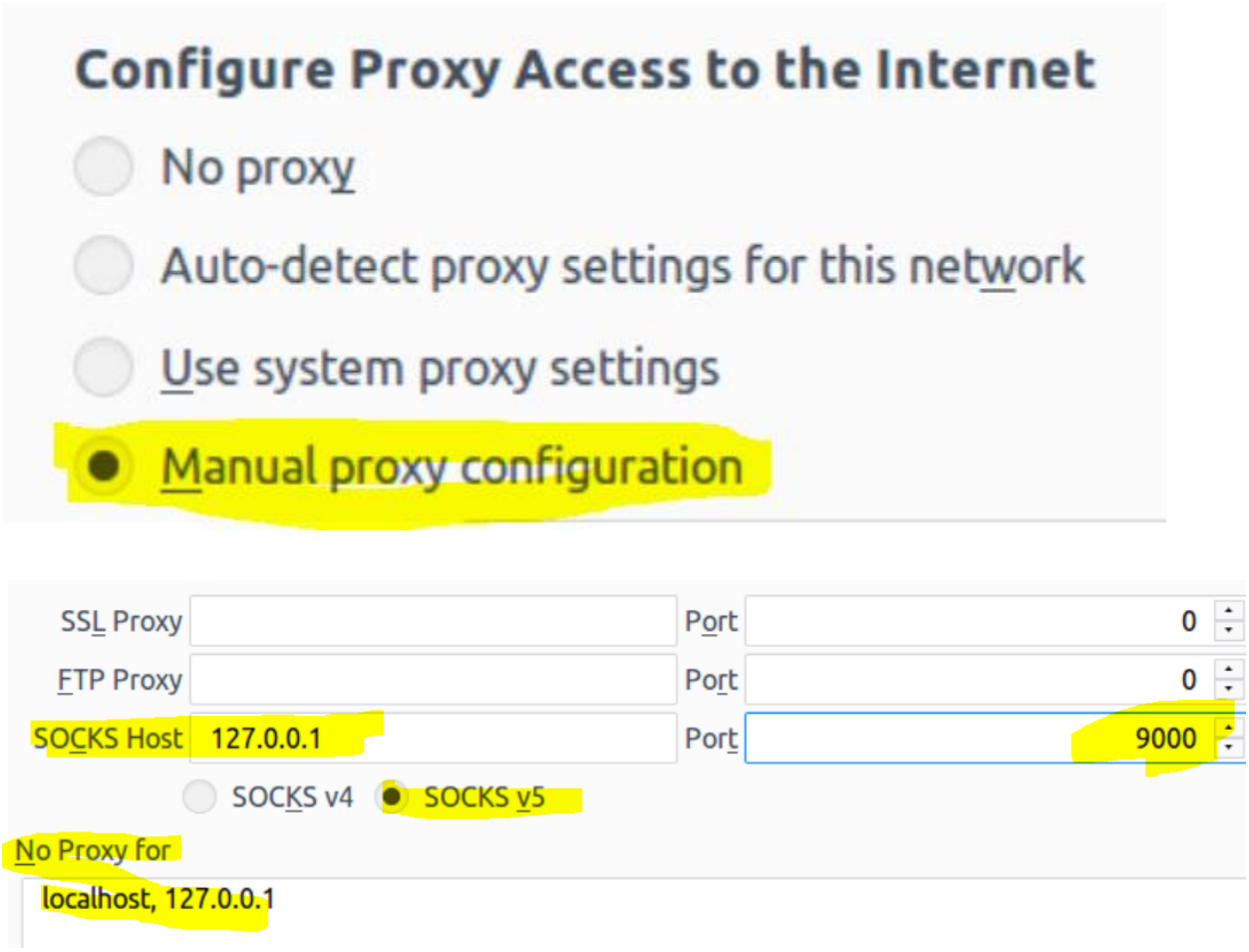


Fig: Configuring the SOCKS proxy.

Closed all ip address of Facebook:

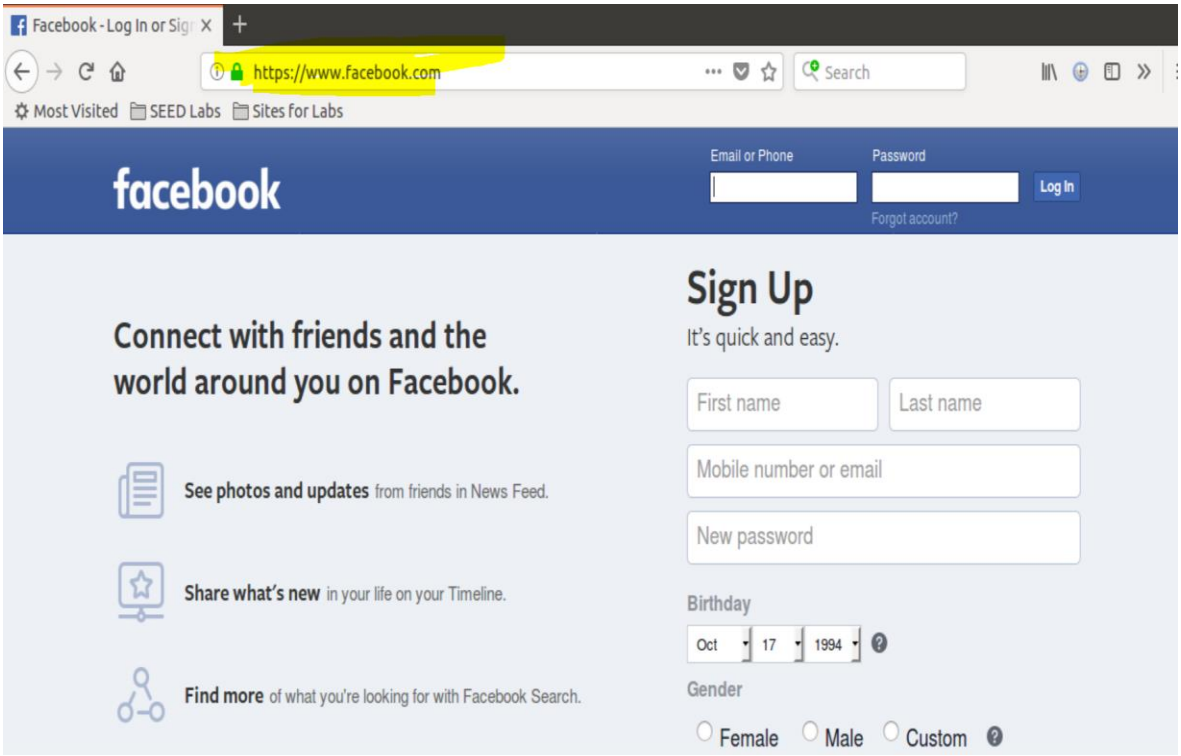
```
[10/17/19]seed@VM:~$ sudo ufw reject out from 66.220.144.0 to 66.220.159.255
Rules updated
[10/17/19]seed@VM:~$ sudo ufw reject out from 69.63.176.0 to 69.63.191.255
Rules updated
[10/17/19]seed@VM:~$ sudo ufw reject out from 204.15.20.0 to 204.15.23.255
Rules updated
```

```
[10/17/19]seed@VM:~$ telnet localhost 9000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
www.facebook.com
Connection closed by foreign host.
[10/17/19]seed@VM:~$
```

1. Run Firefox and go visit the Facebook page. Can you see the Facebook page? Please describe your observation.

Ans:

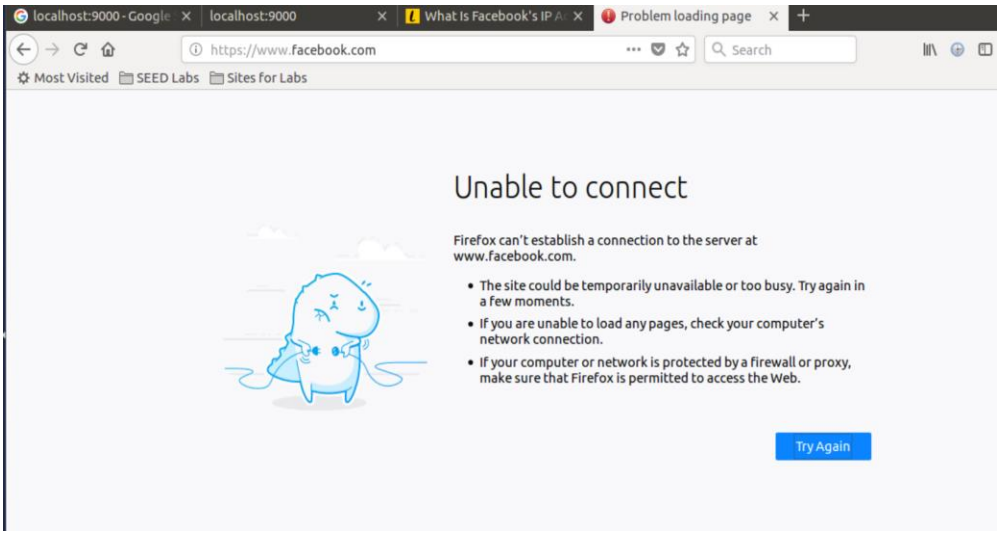
Yes , we were able to go into facebook through SSH tunnel.



2. After you get the facebook page, break the SSH tunnel, clear the Firefox cache, and try the connection again. Please describe your observation.

Ans:

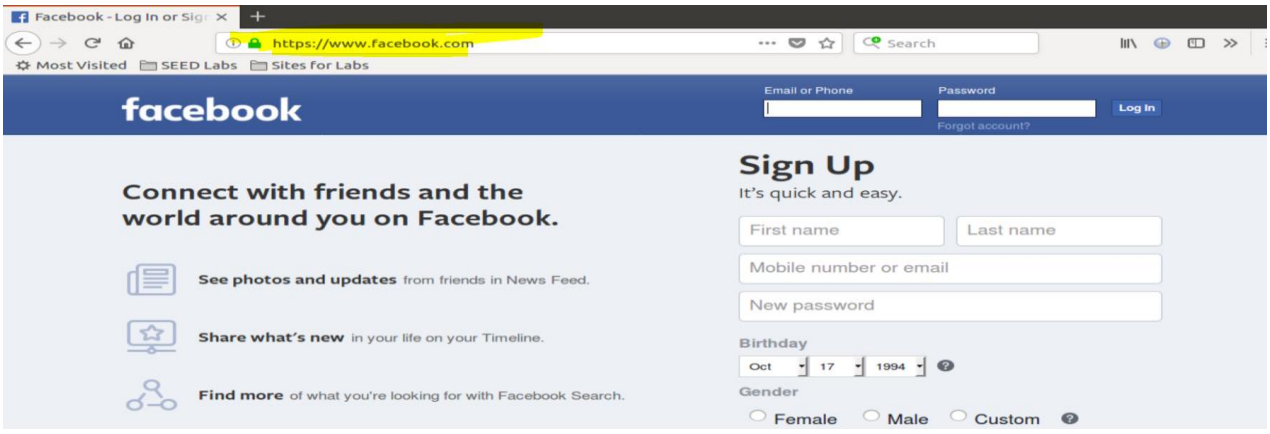
Actually, right after we break the tunnel, the facebook page did not show up which is why we were unable to connect into facebook via our machine B, because we have closed all the connection.



3. Establish the SSH tunnel again and connect to Facebook. Describe your observation.

Ans:

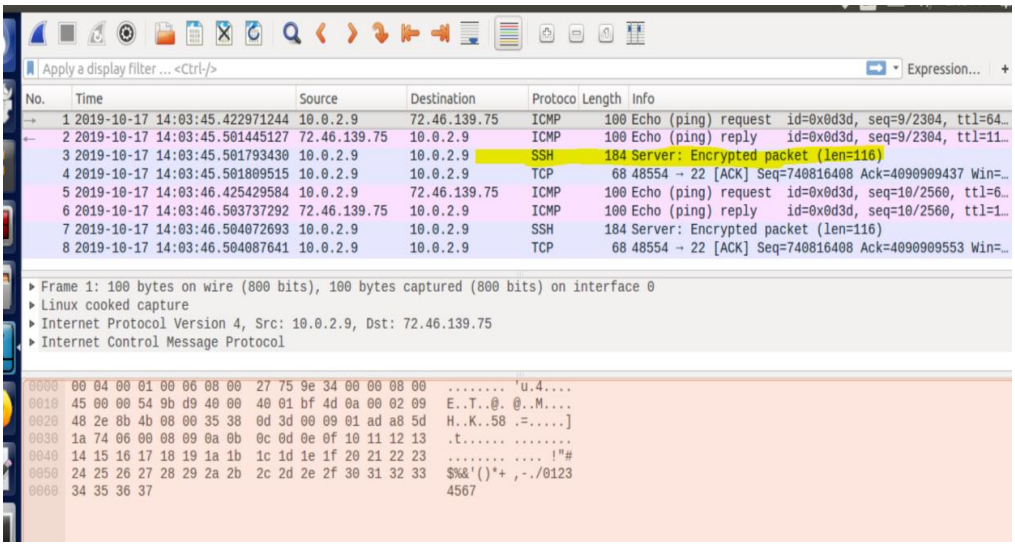
Right after establishing the SSH tunnel, we were certainly able to gain access into facebook again.



4. Please explain what you have observed, especially on why the SSH tunnel can help bypass the egress filtering. You should use Wireshark to see what exactly is happening on the wire. Please describe your observations and explain them using the packets that you have captured.

Ans:

Most significantly, SSH tunnel can help bypass the firewall and make an encrypted secure connection which assists us to take control of our machine .It was a great experience how SSH tunnel makes our life easy even if any social media is blocked into our workplace, we will still be able to use those websites by establishing connection to our localhost.



2.4 Task4: Evading Ingress Filtering

Ingress filtering is a method used by ISPs (Internet Service Provider) to filter suspicious traffic from entering a network.

In this task we have two machines with the following IP addresses:

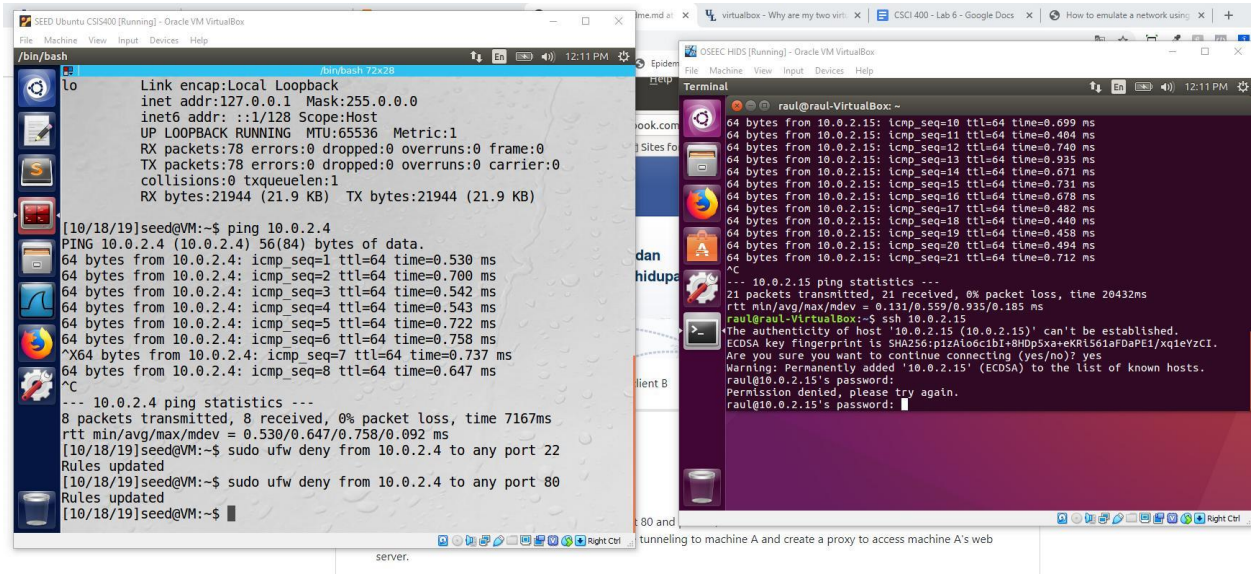
- Machine A: 10.0.2.15
- Machine B: 10.0.2.4

In the following images we ping machine A from machine B, and it successfully communicates.

After we use the following commands to restrict access to machine A using the following

sudo ufw deny from 10.0.2.4 to any port 22
sudo ufw deny from 10.0.2.4 to any port 80,

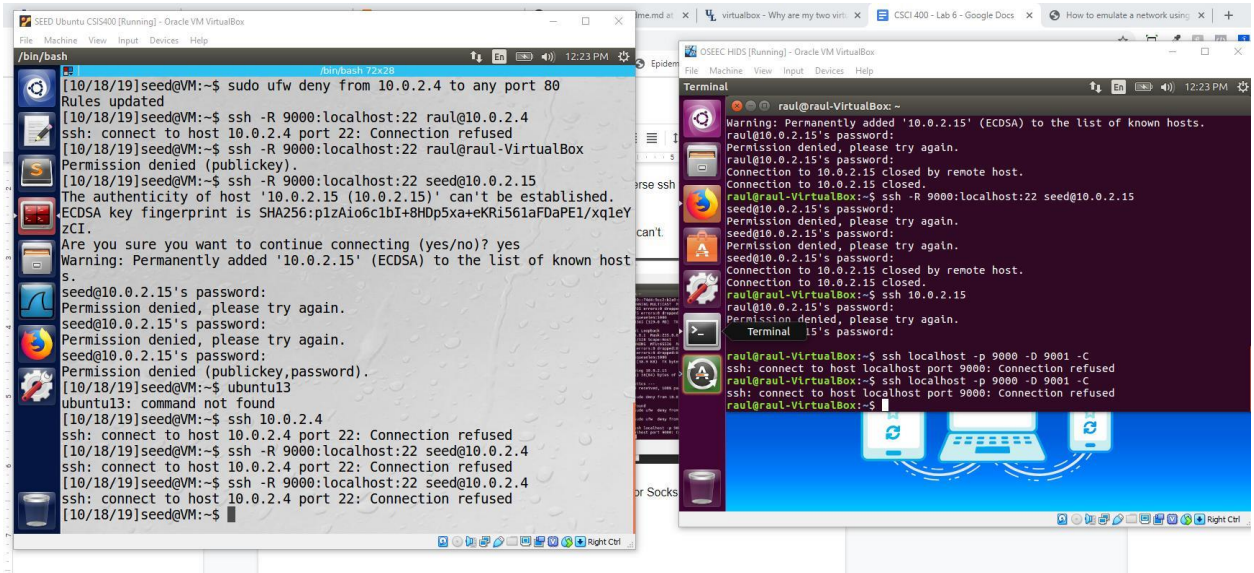
And we can see that trying to ssh into machine A it is not successful since Permission is denied.



Next we try to do a reverse ssh tunneling to machine A with the -R argument as follows

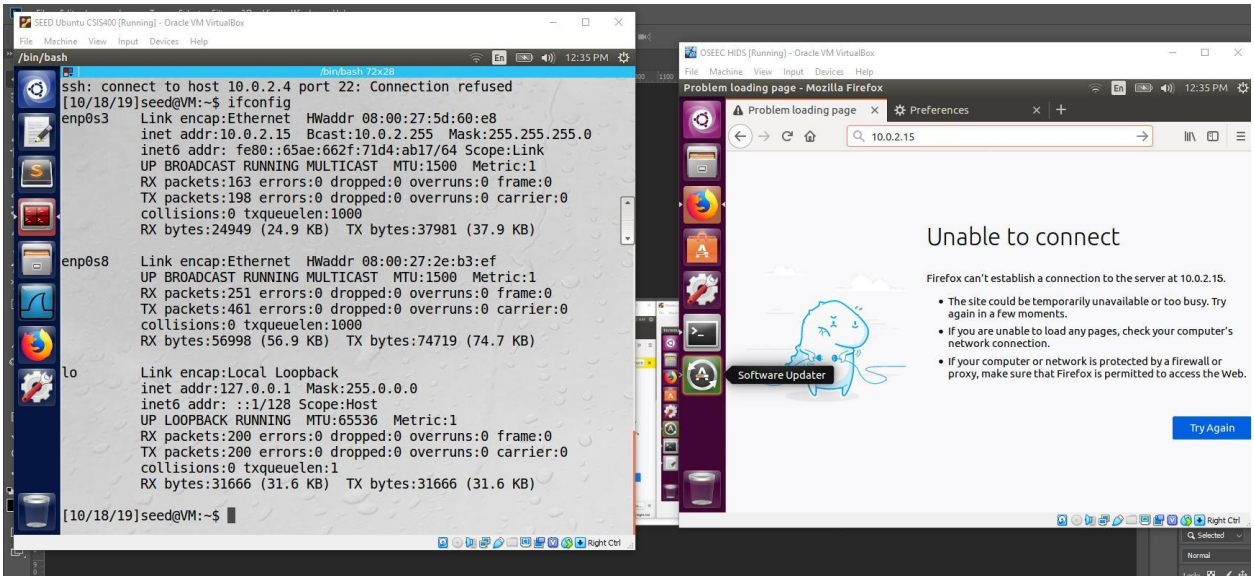
ssh -R localhost:22 seed@10.0.2.4

and again, we get a Permission Denied.

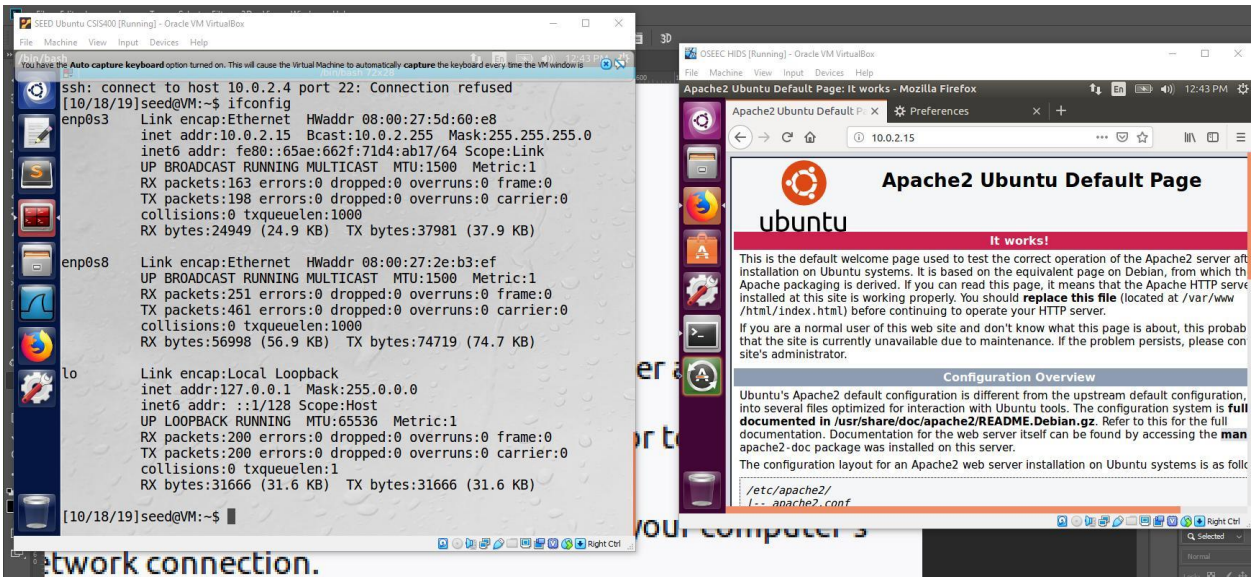


We tunnel to machine A through port 9000, and we do a reverse ssh tunnel to port 9001.

Now we try to access the machine A without a proxy, but we can't.



Then we manually change the proxy in Firefox preferences for Socks host as localhost and Port to 9001 and now we are able to access machine A.



Local DNS Attack Lab

DNS which stands for Domain Name System which is a very important infrastructure in the entire Internet. This is because DNS basically acts as a phone book which converts domain names into the IP address and enables the communication between two or more than two devices in the network. The conversion activities take place behind the scenes but sometimes the attacker will manipulate and mislead to the alternate domain containing malicious and hack the system. This type of attack is called a DNS attack. In this lab we will look at how to perform the local DNS attack and observe how attackers manipulate or trick the DNS system behind the scene.

Setting Up a Local DNS Server

The following IP addresses are assigned respectively to those with the given name in the same network to perform the successful attack task in our VM.

User Machine 10.0.2.6

Attacker machine 10.0.2.7

Local DNS Server 10.0.2.8

Task 1: Configure the User Machine


```
[05/11/20]seed@VM:.../resolv.conf.d$ dig 10.0.2.8

; <<>> DiG 9.10.3-P4-Ubuntu <<>> 10.0.2.8
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 244
45
;; flags: qr aa rd ra ad; QUERY: 1, ANSWER: 1, AUTHORIT
Y: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;10.0.2.8.                      IN      A

;; ANSWER SECTION:
10.0.2.8.               0       IN      A       10.0.2.
8

;; Query time: 0 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Mon May 11 13:59:20 EDT 2020
```

In this task our purpose is to make 10.0.2.8 our local DNS server for user machine **10.0.2.6** which was successful after we configured the user machine as shown above.

Task 2: Set up a Local DNS Server

For this setup we have the server program called **BIND** and is pre-build in our VM. So, we will just configure the server and turn off some countermeasures as shown below:

```
[05/11/20]seed@VM:.../bind$ vim named.conf.options
[05/11/20]seed@VM:.../bind$ sudo rndc dumdb -cahce
rndc: 'dumdb' failed: unknown command
[05/11/20]seed@VM:.../bind$ sudo rndc dumpdb -cahce
[05/11/20]seed@VM:.../bind$ sudo rndc flush
[05/11/20]seed@VM:.../bind$ sudo service bind9 restart
```

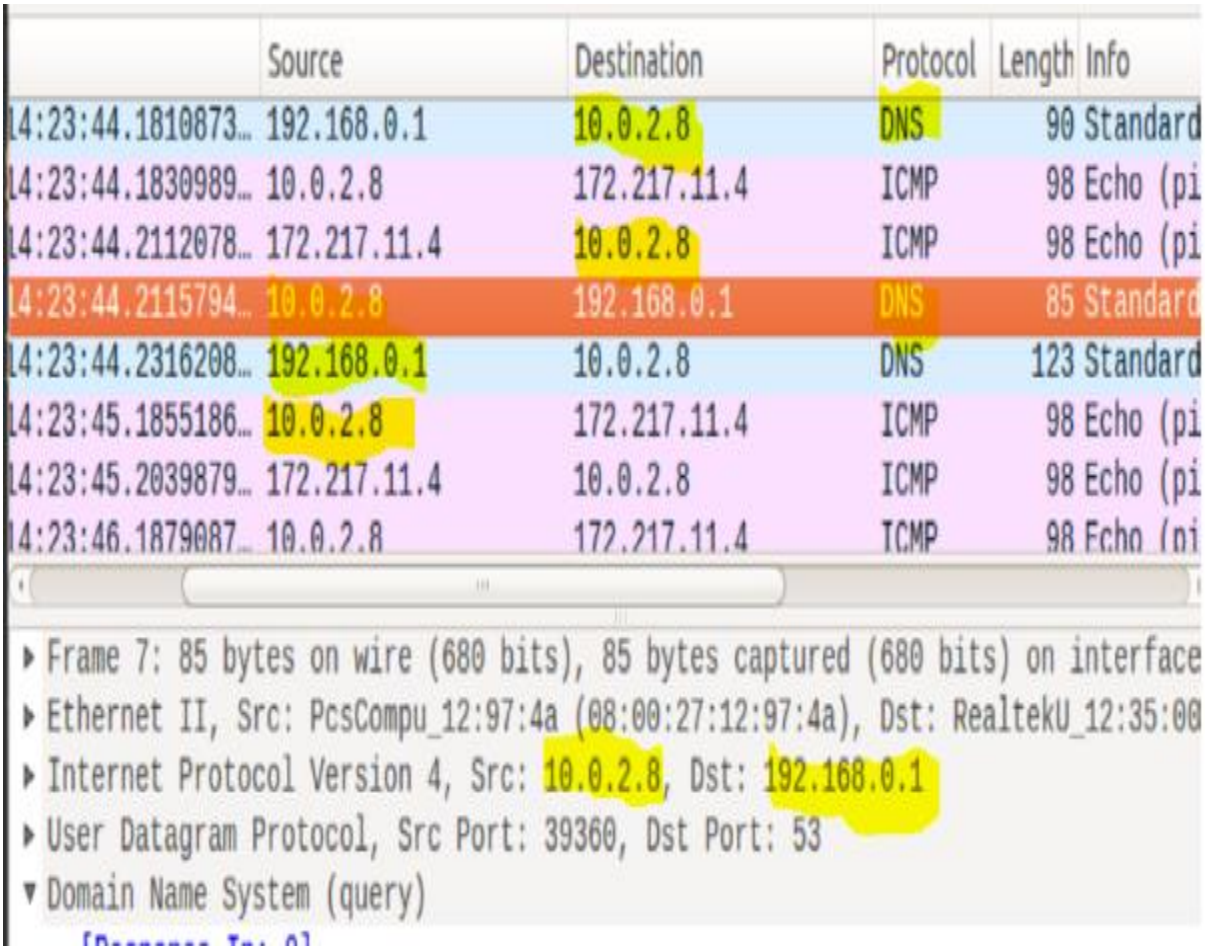
Task 3:Use the DNS server

To see weather out Local DNS server has been set properly or not we use user machine **10.0.2.6** to ping the google as shown below:

```
[05/11/20]seed@VM:~$ ping www.google.com
PING www.google.com (172.217.11.4) 56(84) bytes of data
.
64 bytes from lga25s60-in-f4.1e100.net (172.217.11.4):
icmp_seq=1 ttl=54 time=28.1 ms
```

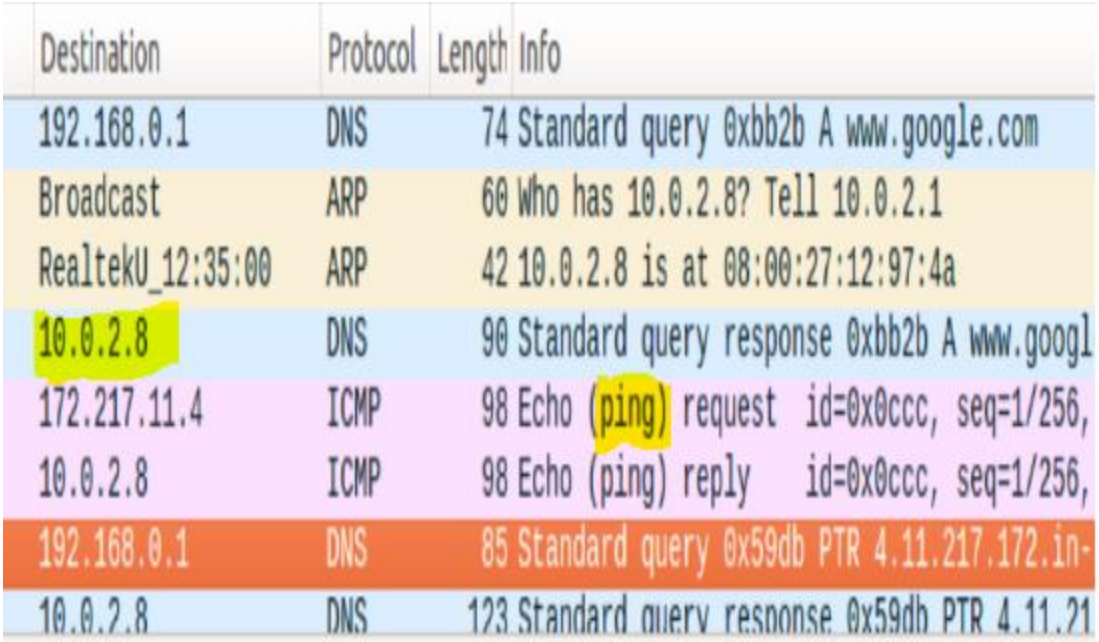
And to observe the communication we used Wireshark where we can see the DNS as **10.0.2.8**

As local DNS send the **ping** request to www.google.com and get a ping reply back as we can see below.



	Source	Destination	Protocol	Length	Info
14:23:44.1810873…	192.168.0.1	10.0.2.8	DNS	90	Standard
14:23:44.1830989…	10.0.2.8	172.217.11.4	ICMP	98	Echo (pi
14:23:44.2112078…	172.217.11.4	10.0.2.8	ICMP	98	Echo (pi
14:23:44.2115794…	10.0.2.8	192.168.0.1	DNS	85	Standard
14:23:44.2316208…	192.168.0.1	10.0.2.8	DNS	123	Standard
14:23:45.1855186…	10.0.2.8	172.217.11.4	ICMP	98	Echo (pi
14:23:45.2039879…	172.217.11.4	10.0.2.8	ICMP	98	Echo (pi
14:23:46.1879087…	10.0.2.8	172.217.11.4	ICMP	98	Echo (pi

```
▸ Frame 7: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface
▸ Ethernet II, Src: PcsCompu_12:97:4a (08:00:27:12:97:4a), Dst: RealtekU_12:35:00
▸ Internet Protocol Version 4, Src: 10.0.2.8, Dst: 192.168.0.1
▸ User Datagram Protocol, Src Port: 39360, Dst Port: 53
▾ Domain Name System (query)
```



Destination	Protocol	Length	Info
192.168.0.1	DNS	74	Standard query 0xbb2b A www.google.com
Broadcast	ARP	60	Who has 10.0.2.8? Tell 10.0.2.1
RealtekU_12:35:00	ARP	42	10.0.2.8 is at 08:00:27:12:97:4a
10.0.2.8	DNS	90	Standard query response 0xbb2b A www.googl
172.217.11.4	ICMP	98	Echo (ping) request id=0x0ccc, seq=1/256,
10.0.2.8	ICMP	98	Echo (ping) reply id=0x0ccc, seq=1/256,
192.168.0.1	DNS	85	Standard query 0x59db PTR 4.11.217.172.in-
10.0.2.8	DNS	123	Standard query response 0x59db PTR 4.11.21

Summary:

In a nutshell, without DNS an Internet cannot work, so attackers always have a main target on the DNS server. Basically, an attacker can perform two types of attack DoS Denial-of- Service attack and DNS spoofing attack. DNS traffic is not encrypted because it is not a securely designed protocol. So, attackers are able to perform a DNS spoofing attack and cause cache poisoning.

Remote DNS Attack Lab

There are some limitations on local DNS attack for instance, both the attacker and the victim were in the same LAN to sniff the victims DNS query. In the real world the attacks are performed remotely, and the remote attackers are having difficulty launching the attack because they cannot sniff the Victim DNS query easily as like in local DNS attacks. To gain the victim DNS query the attacker will send the request to the targeted DNS server with the intended DNS query and launch the spoof attack without waiting until the attacker are succeed during this period the attacker will be compromising cache but after Kaminsky attack was discover the attacker can launch the spoof attack without waiting. In this lab we will be going through how Kaminsky DNS attack works by setting up the lab environment.

Lab Environment

In this task we setup the VMs with three machines where:

User Machine 10.0.2.6

Attacker machine 10.0.2.7

Local DNS Server Apollo 10.0.2.8

Configure the Local DNS Server:

In this configuration we will be working with a local DNS server to install the DNS server software which is known as BIND.

As we set up the lab in local DNS attack in the same way the lab environment has been setup in 3 easy steps:

Step1: We Configure the BIND 9 server

Step 2: We Turn off DNSSEC

Step3: Fix the source Ports.

```
options{
        // dnssec-validation auto;
        dnssec-enable no;

        dump-file "/var/cache/bind/dump.db";
        auth-nxdomain no;    # conform to RFC1035

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

User Machine 10.0.2.6:

In this task we setup the 10.0.2.6 user machine where we will configure the 10.0.2.8 as the local DNS server as shown below:

```
erated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WIL
L BE OVERWRITTEN
nameserver 10.0.2.8
```

Kaminsky Attacker:

To conduct this attack first the victim DNS is set up with adding the following entry in the file named.conf.default-zones and observe the following.

```
zone "ns.dnslabattack.net"{
        type master;
        file "/etc/bind/db.attacker";
        };
~
-- INSERT --                              34,1          Bot
```

Observation:

Even if the attack was not successful, we can see how the successful attack would look like in the given result. If the attack was successful, the NS record for example.com becomes ns.dnslabattack.net which can be verified when we use dig tool for the given domain name.

```
[05/11/20]root@VM:~# dig twysw.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> twysw.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 61
079
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY:
1, ADDITIONAL: 1

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

;; AUTHORITY SECTION:
example.com.            3600    IN      SOA     ns.ican
n.org. noc.dns.icann.org. 2019121388 7200 3600 1209600
3600

;; Query time: 43 msec
```

```
[05/11/20]root@VM:~# dig twysw.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> twysw.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 266
30
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY:
0, ADDITIONAL: 0

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

;; Query time: 12 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Mon May 11 20:13:03 EDT 2020
;; MSG SIZE  rcvd: 35
```

Summary:

In a nutshell, the remote DNS attack the attacker needs to accomplish three main target tasks which include first trigger the targeted DNS server to send the DNS queries, second spoof the reply and third negotiate the cache effect. To prevent DNS cache poisoning attack the DNSSEC was developed whose main idea was to use authentic digital signature but most of the DNS server does not support this mechanism till today which makes it easier to conduct the attack.