UE18CS335 – COMPUTER NETWORK SECURITY

Lab – 3 LINUX FIREWALL EXPLORATION LAB

Date: 21/02/2021

By:

Nitish S

PES2201800368

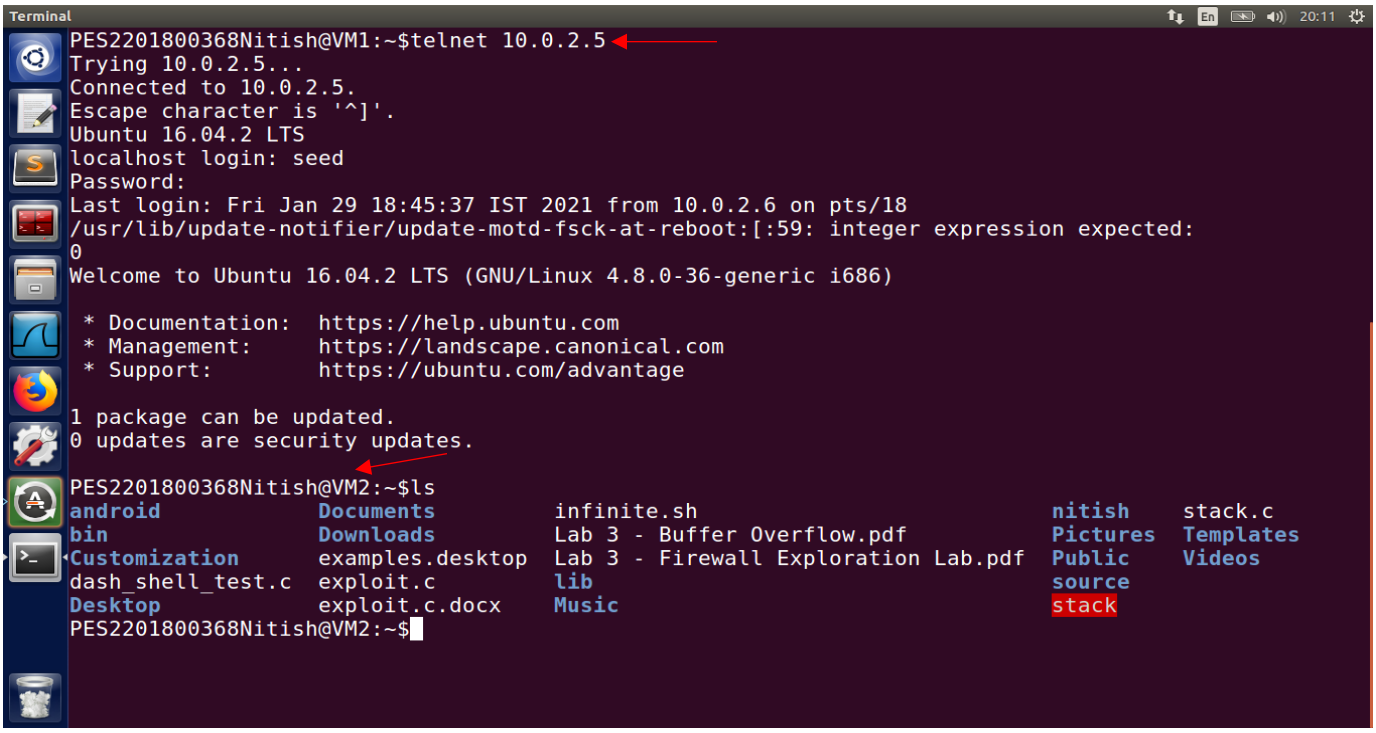
6 ‘A’

TASK 1: USING FIREWALL

Setup: - VM1 = 10.0.2.4 ; VM2 = 10.0.2.5

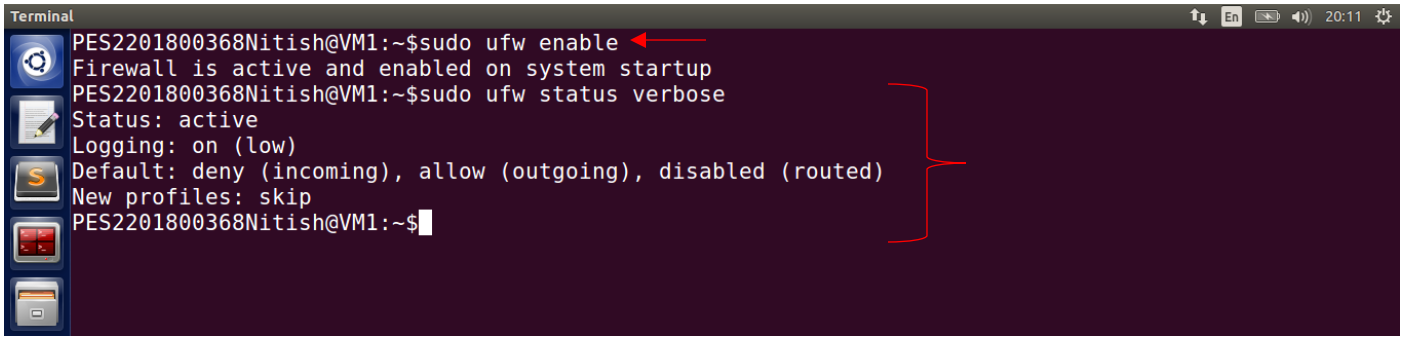
PART-1:

Telnet to VM2 from VM1 to check if it is a success or not.



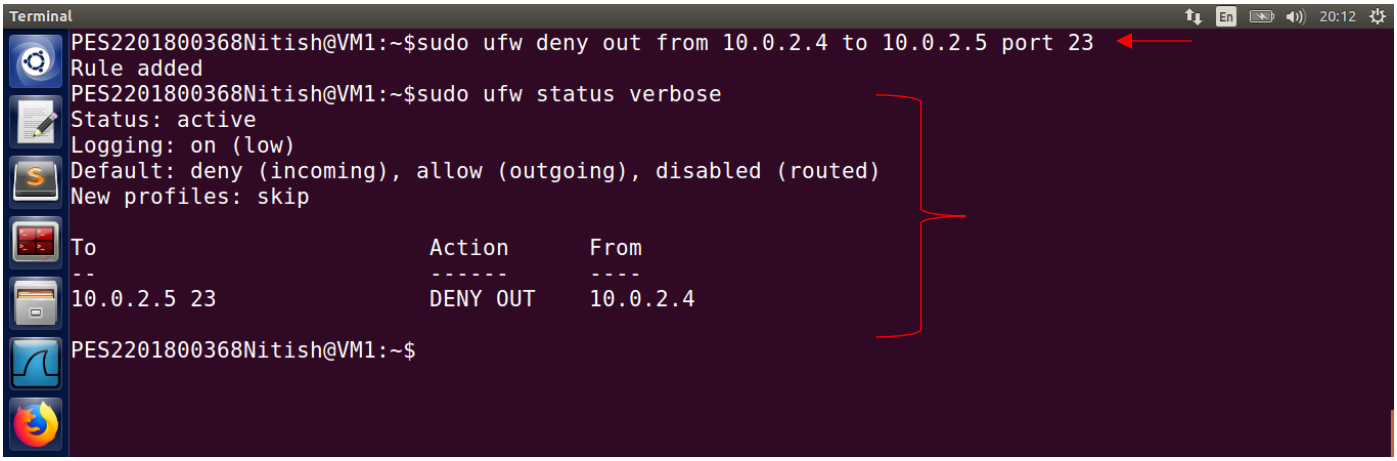
SCREENSHOT SHOWING SUCCESSFUL TELNET CONNECTION FROM VM1 TO VM2

Enabling the firewall and checking the default set rules



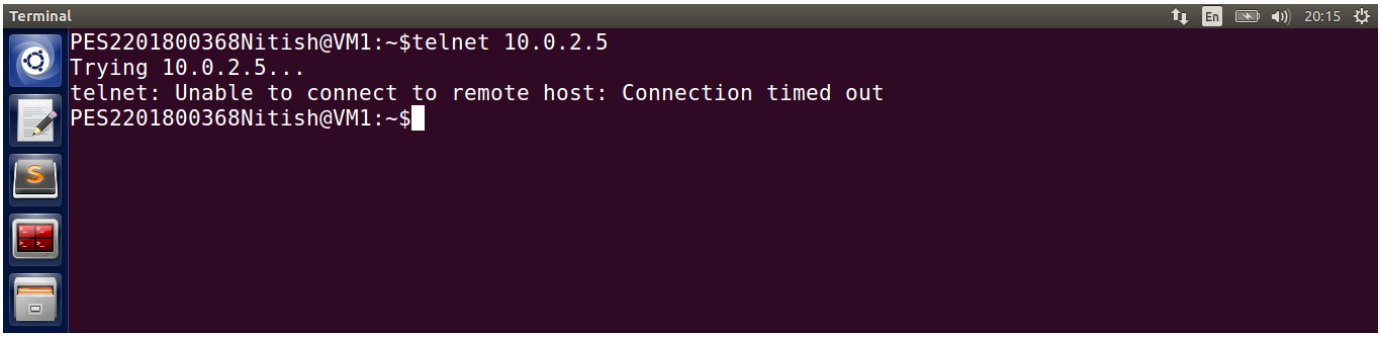
SCREENSHOT SHOWING THE DEFAULT FIREWALL RULES

Manually configuring the firewall to deny connection to VM2 using port 23



SCREENSHOT SHOWING THE RULE ADDED TO FIREWALL

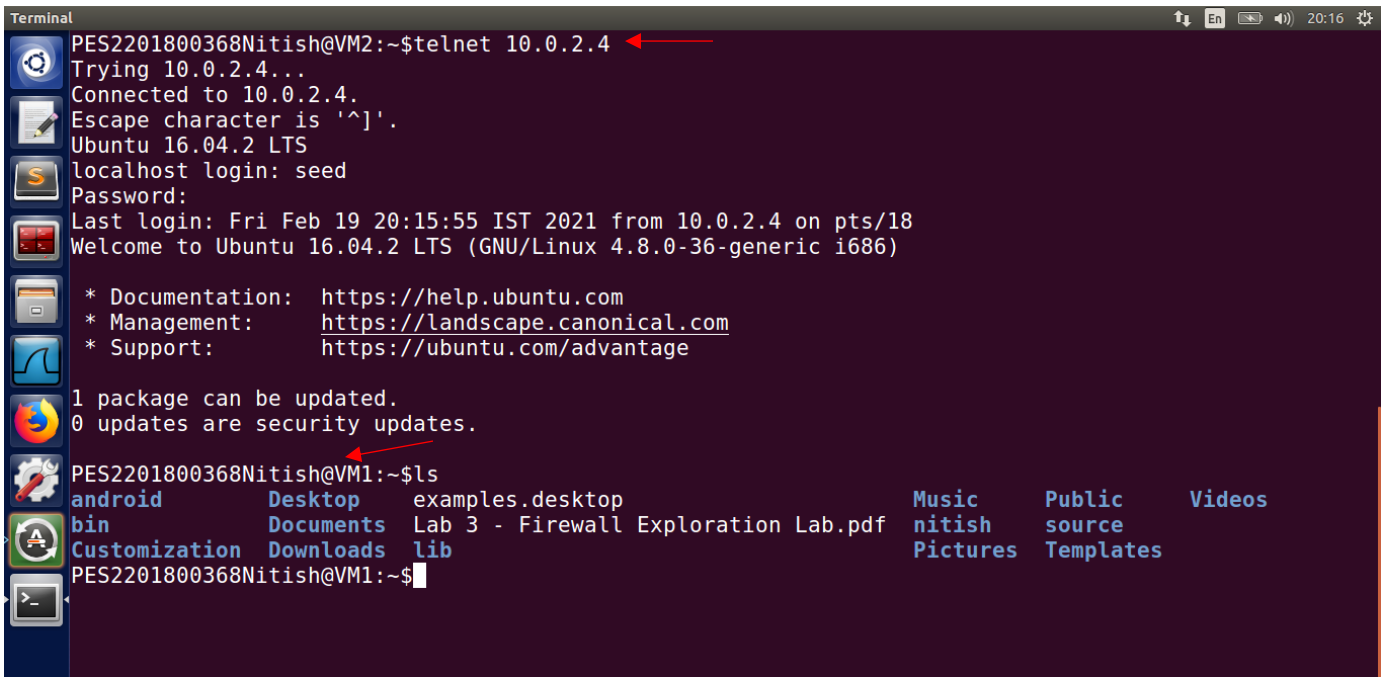
Verification of the added rule



SCREENSHOT SHOWING THE FAILURE OF TELNET CONNECTION FROM VM1 TO VM2 DUE TO THE NEWLY ADDED FIREWALL RULE

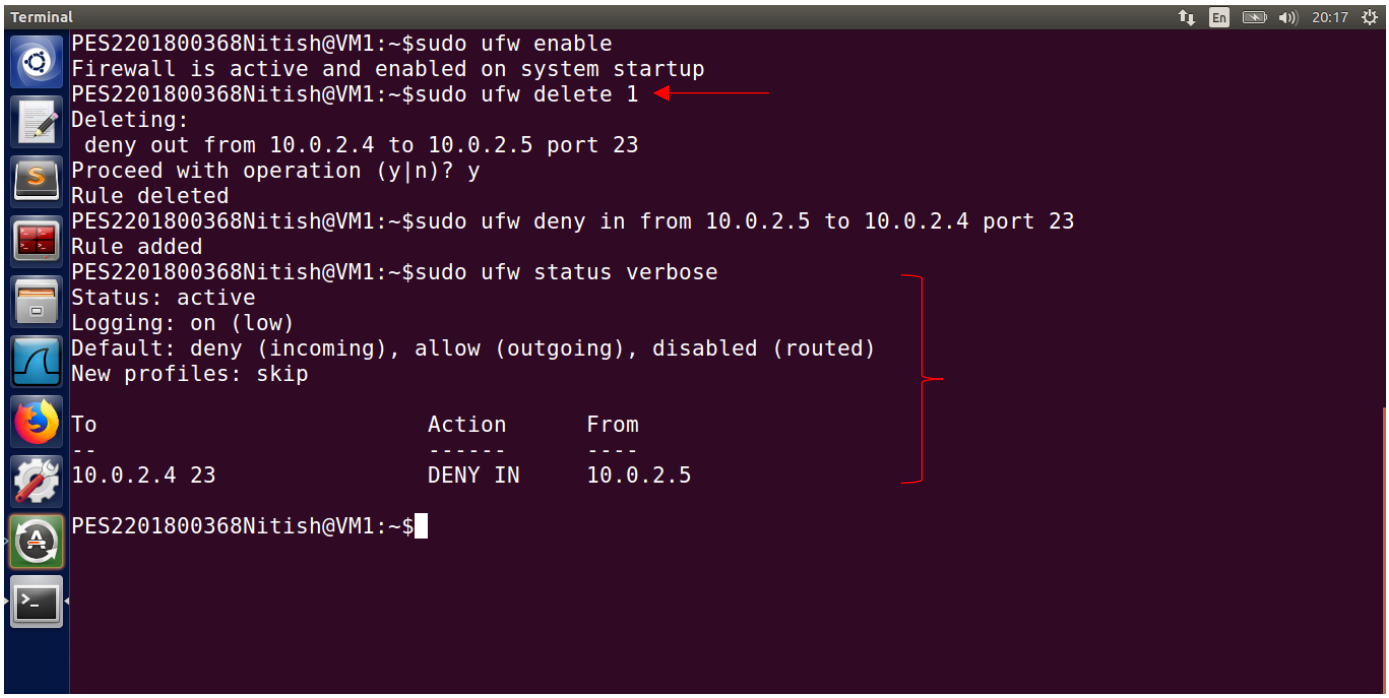
PART – 2:

Telnet to VM1 from VM2 to check if it's a success or not



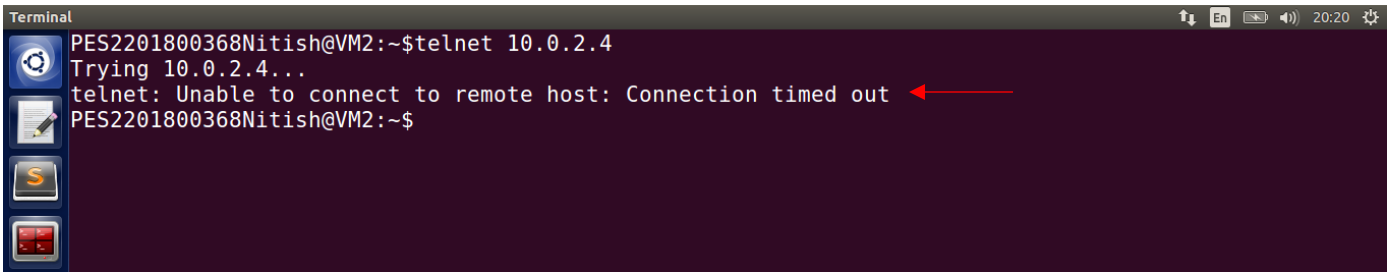
SCREENSHOT SHOWING SUCCESSFUL TELNET CONNECTION FROM VM2 TO VM1

Deleting the existing rules and configuring a new rule to prevent telnet connection from VM2 to VM1



SCREENSHOT SHOWING THE NEWLY ADDED FIREWALL RULE

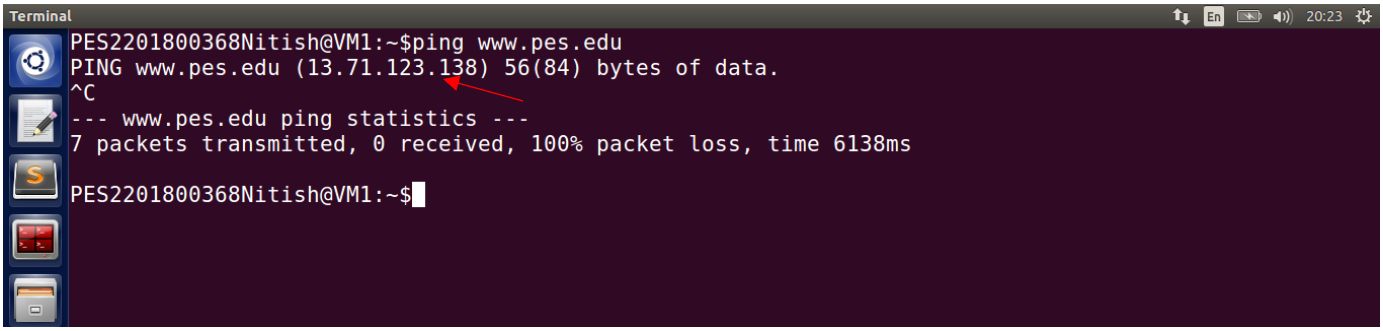
Testing the added firewall rule



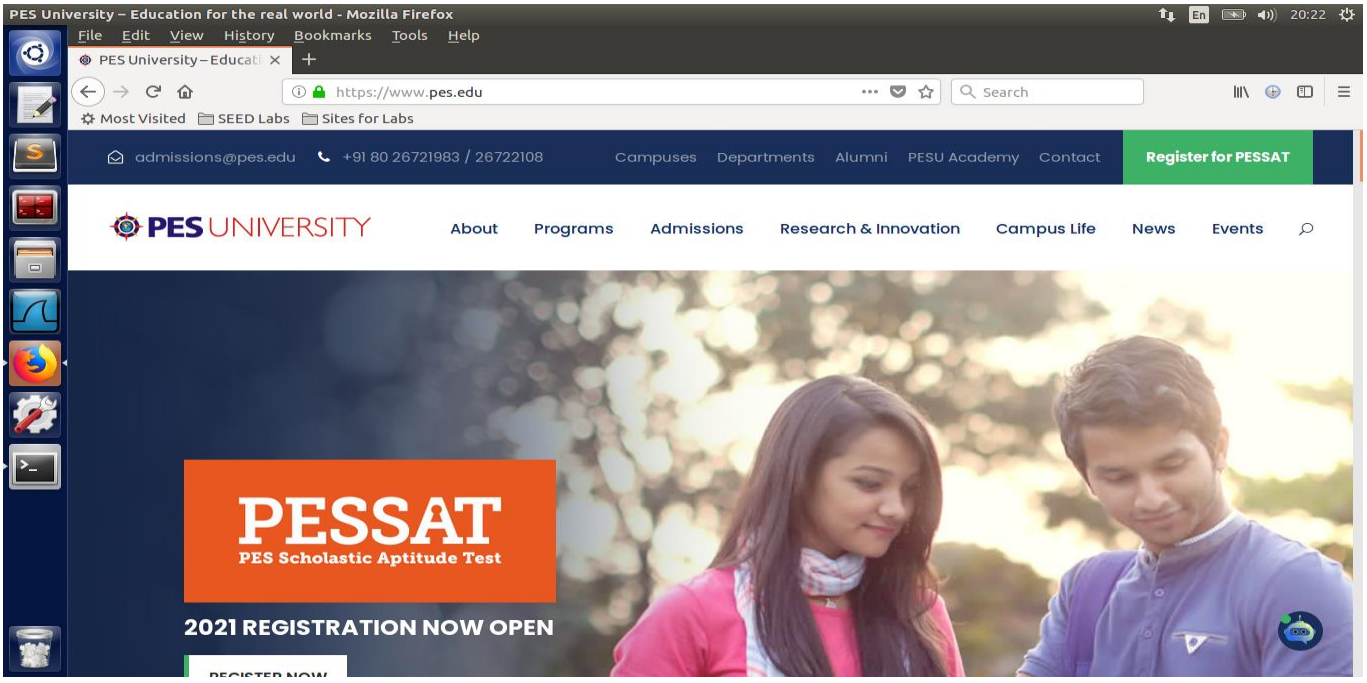
SCREENSHOT SHOWING THE FAILURE OF TELNET FROM VM2 TO VM1 DUE TO THE NEWLY CONFIGURED FIREWALL RULE

PART – 3:

To obtain the IP address of www.pes.edu using ping command

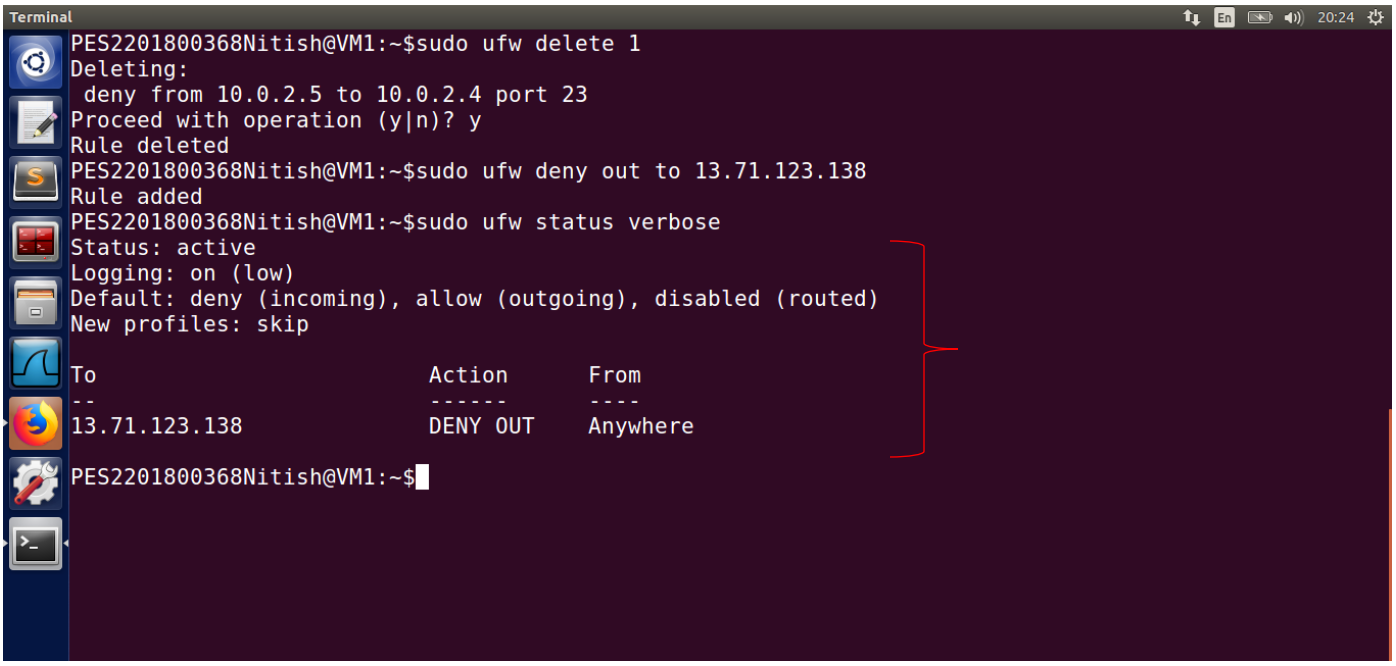


SCREENSHOT SHOWING THE IP ADDRESS OF www.pes.edu



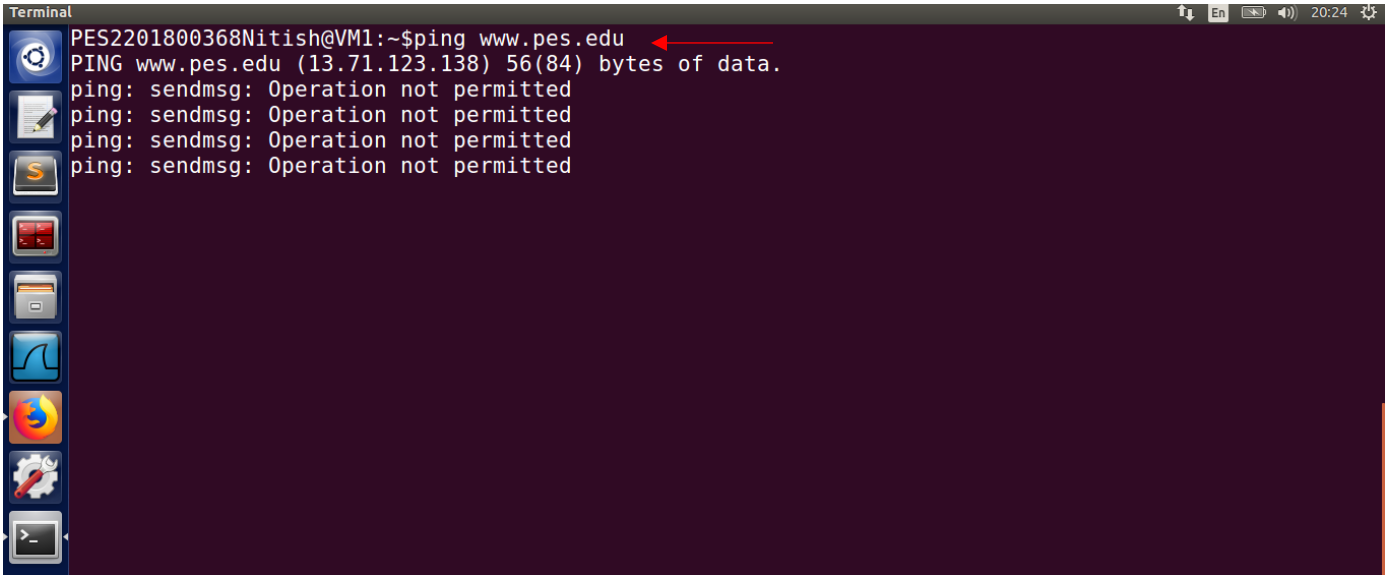
SCREENSHOT SHOWING www.pes.edu ACCESSIBLE FROM THE BROWSER

Adding a new rule to the firewall to deny access to www.pes.edu website

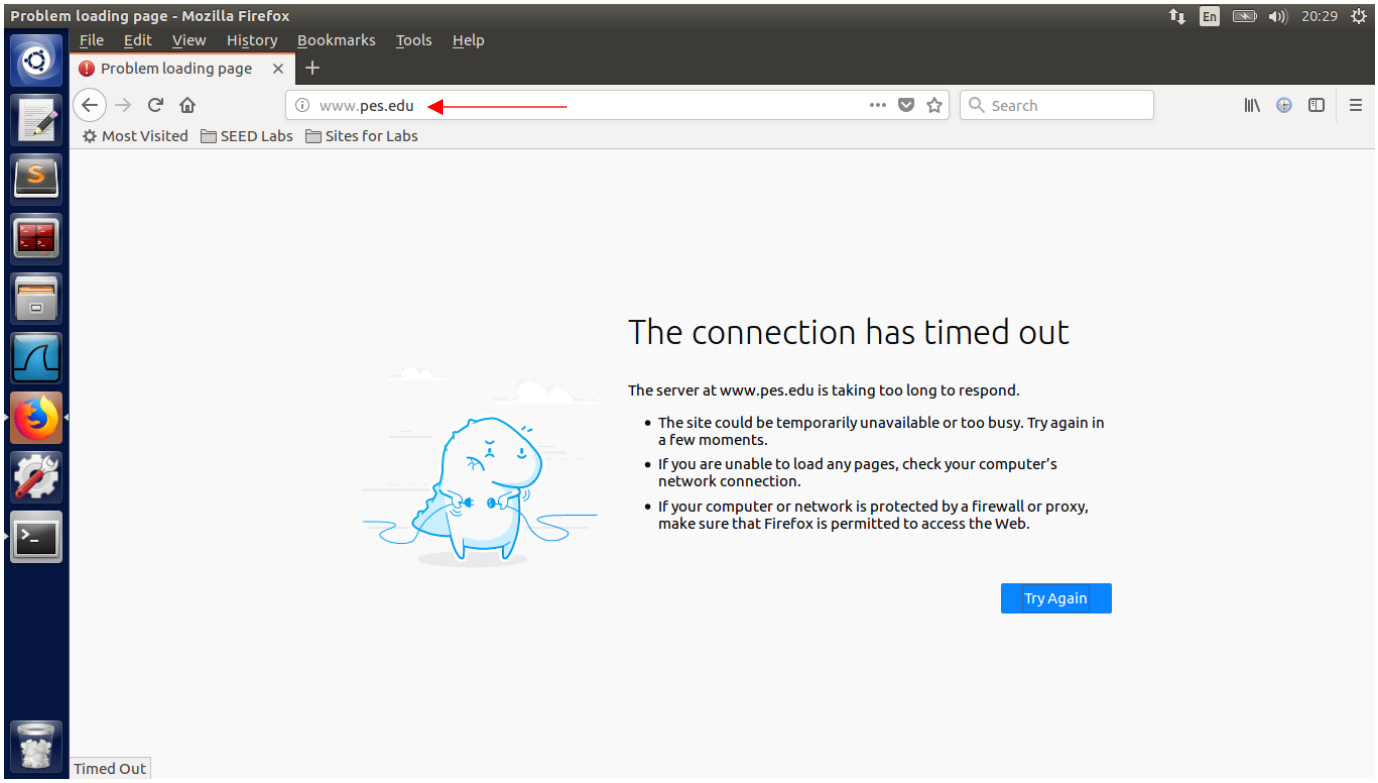


SCREENSHOT SHOWING THE NEWLY CONFIGURED RULE

Testing the newly added rule



SCREENSHOT SHOWING THE FAILURE OF PING COMMAND TO THE BLOCKED WEBSITE



SCREENSHOT SHOWING INACCESSIBILITY OF THE WEBSITE FROM THE BROWSER

Observations from Task 1:

Firewall is a part of a computer system designed to stop unauthorized traffic flowing from one network to another and is used for the purpose of filtering data and protecting against attacks. In the above task, we perform filtering of incoming or outgoing traffic by configuring the allow/deny rules in the firewall. Denying all incoming and outgoing telnet connections to the system i.e. to port 23 was the first part and the same was tested and verified when a connection timed out message was obtained when tried to use telnet. In the second part, we deny an outgoing connection to a website by using its IP address of the website and its verified using the operation failed message when the ping command was run and connection timed out message when the website was accessed in the browser.

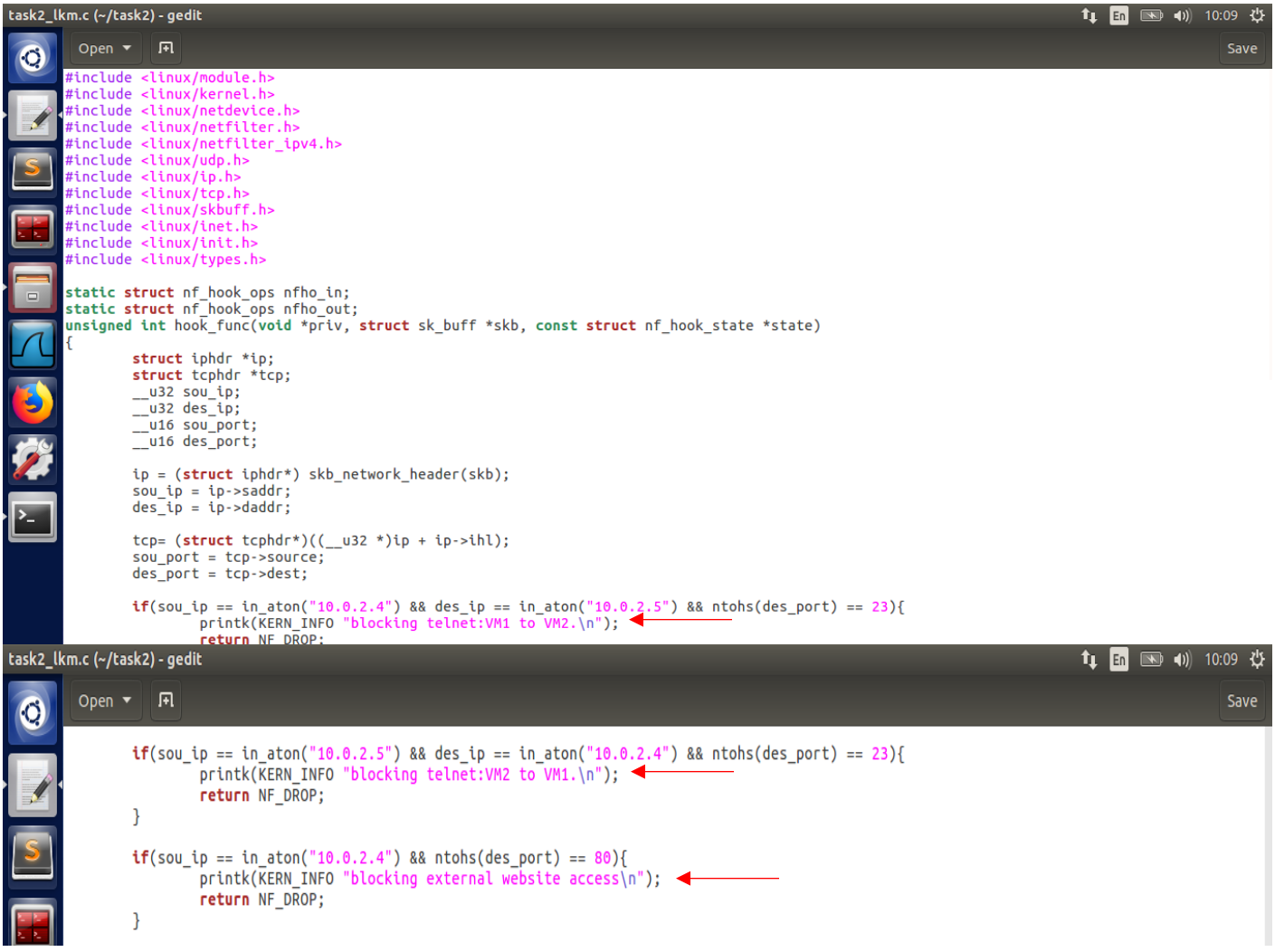
TASK 2: HOW FIREWALL WORKS

In this task, we will develop a firewall using netfilter and LKM. We implement five rules in this firewall:

- Block telnet from VM1 to VM2
- Block telnet from VM2 to VM1
- Block external website access from VM1
- Block ssh from VM1 to VM2
- Block ssh from VM2 to VM1

Setup: VM1 = 10.0.2.4 ; VM2 = 10.0.2.5

task2_lkm.c



```
        if(sou_ip == in_aton("10.0.2.4") && des_ip == in_aton("10.0.2.5") && ntohs(des_port) == 22){
                printk(KERN_INFO "blocking ssh: VM1 to VM2.\n");
        return NF_DROP;
        }

        if(sou_ip == in_aton("10.0.2.5") && des_ip == in_aton("10.0.2.4") && ntohs(des_port) == 22){
                printk(KERN_INFO "blocking ssh:VM2 to VM1.\n");
                return NF_DROP;
        }

        printk(KERN_INFO "allow packet.\n");
        return NF_ACCEPT;
}


int init_module()
{
        printk(KERN_INFO "HELLO NETFILTER!\n");
        nfho_in.hook = hook_func;
        nfho_in.hooknum = NF_INET_PRE_ROUTING;
        nfho_in.pf = PF_INET;
        nfho_in.priority = NF_IP_PRI_FIRST;
        nf_register_hook(&nfho_in);
        nfho_out.hook = hook_func;
        nfho_out.hooknum = NF_INET_POST_ROUTING;
        nfho_out.pf = PF_INET;
        nfho_out.priority = NF_IP_PRI_FIRST;
        nf_register_hook(&nfho_out);
        return 0;
```

C ▾ Tab Width: 8 ▾ Ln 80, Col 40 ▾ INS

```
}
void cleanup_module()
{
        printk(KERN_INFO "BYE NETFILTER!\n");
        nf_unregister_hook(&nfho_in);
        nf_unregister_hook(&nfho_out);
}
```

C ▾ Tab Width: 8 ▾ Ln 80, Col 40 ▾ INS

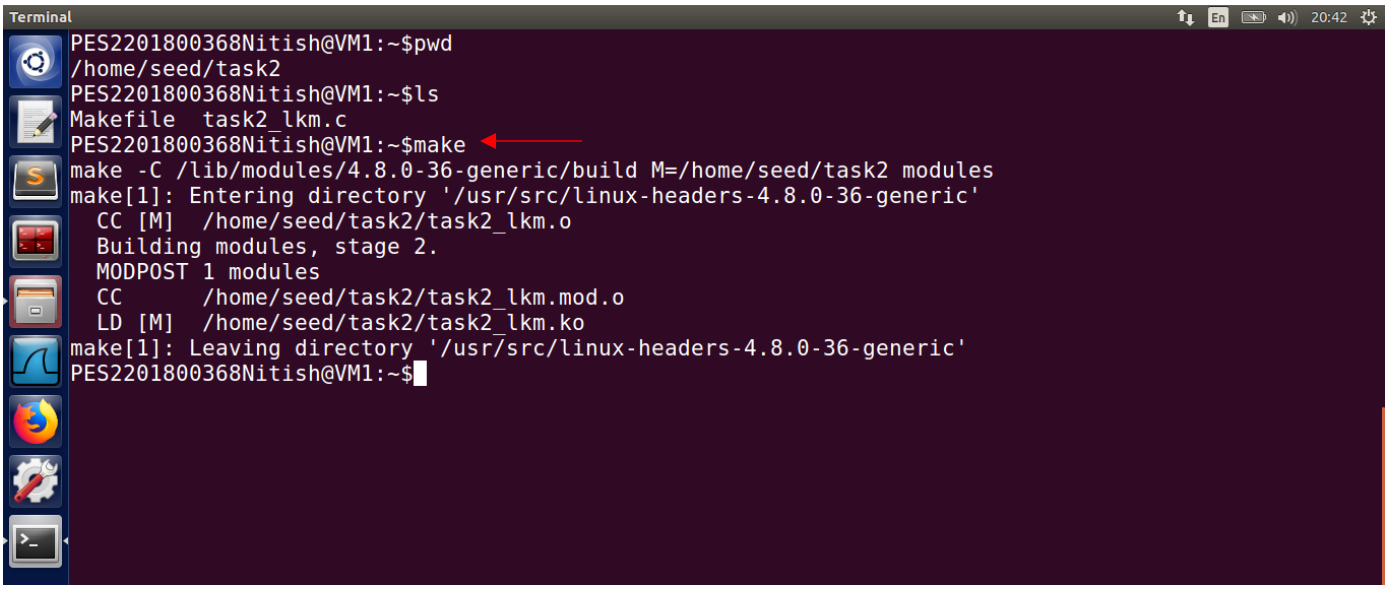
makefile

File Edit View Search Tools Documents Help 10:20

Open ▾ Save

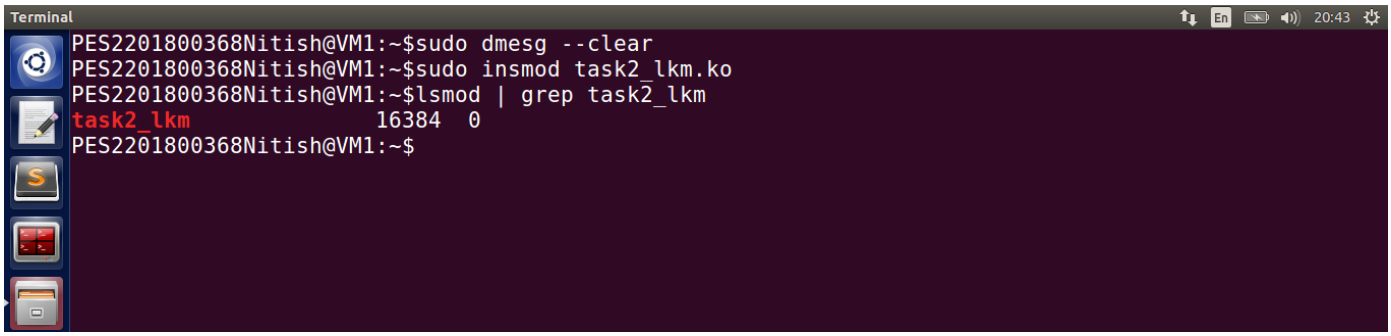
```
obj-m += task2_lkm.o
all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

The program can be compiled using the ***make*** command. (Note: put lkm.c & Makefile into a folder. Open a terminal by right clicking inside that folder. Type ‘make’).



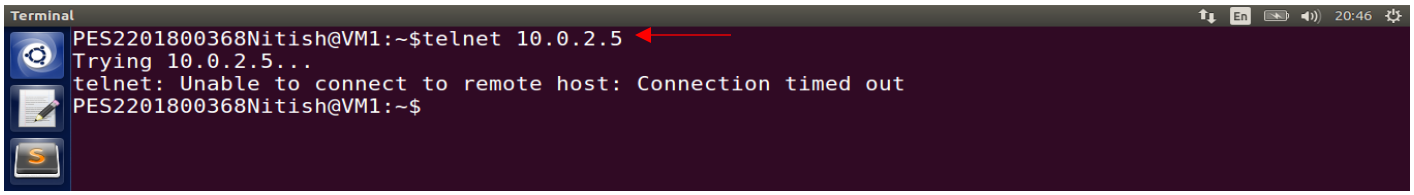
SCREENSHOT SHOWING THE make COMMAND EXECUTION

The compiled kernel module (task2_lkm.ko) can be inserted using insmod:

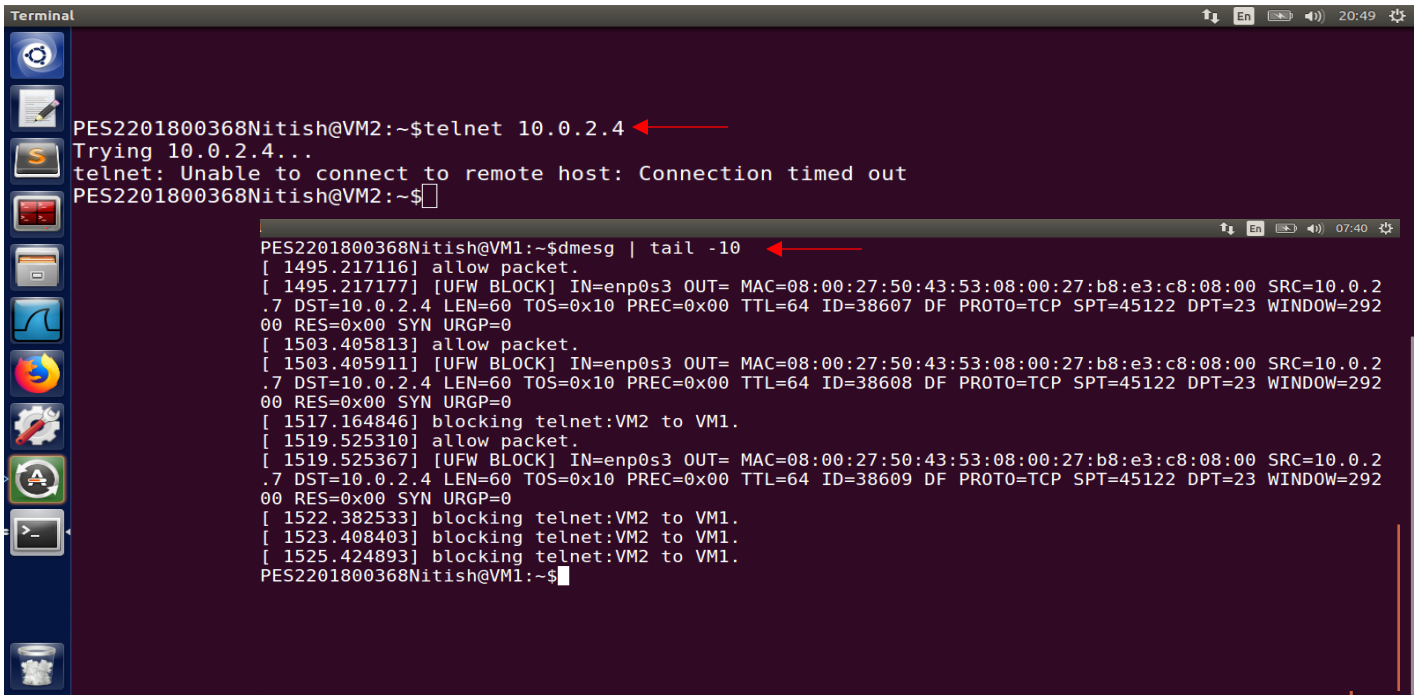


Now we test all the added firewall rules:

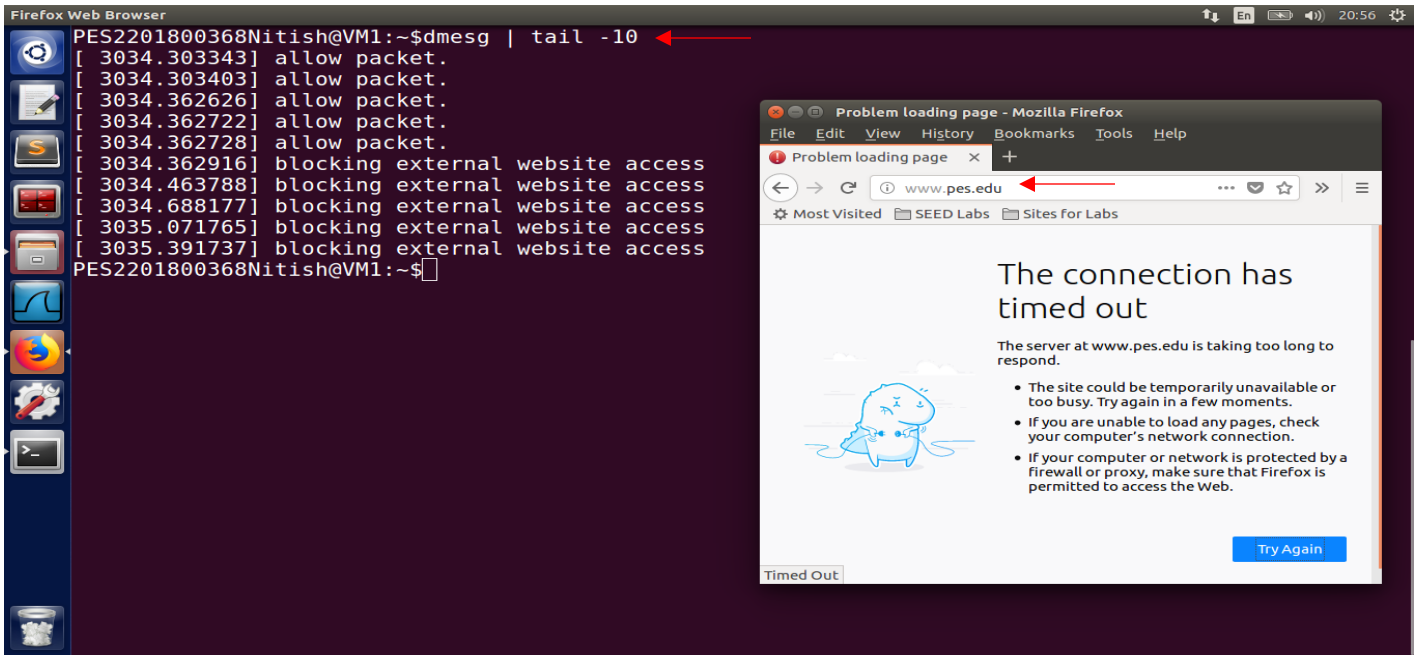
- Block telnet from VM1 to VM2
- Block telnet from VM2 to VM1
- Block external website access from VM1
- Block ssh from VM1 to VM2
- Block ssh from VM2 to VM1



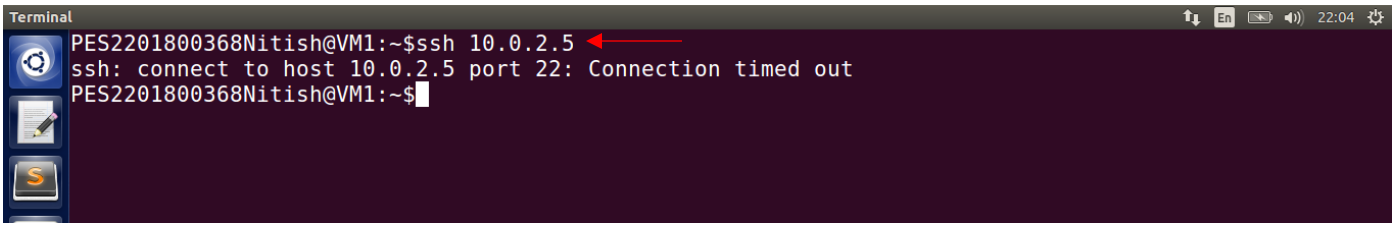
SCREENSHOT SHOWING TELNET FROM VM1 to VM2 BLOCKED



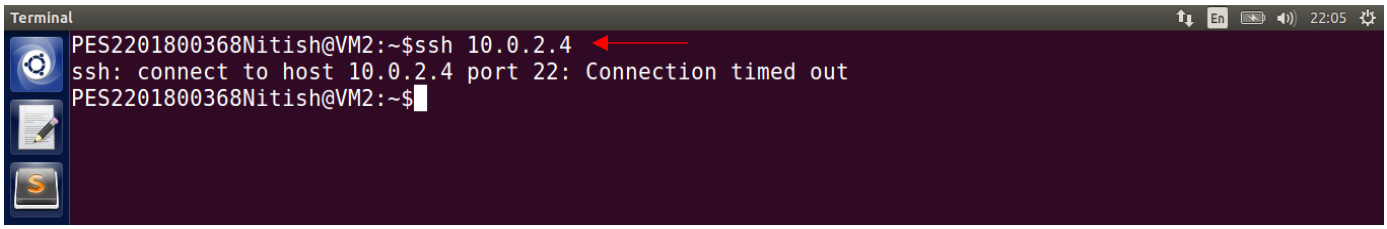
SCREENSHOT SHOWING TELNET FROM VM2 to VM1 BLOCKED AND THE COMMAND RUN TO CHECK THE DROPPING OF PACKETS



SCREENSHOT SHOWING ACCESS TO www.pes.edu BLOCKED



SCREENSHOT SHOWING THE SSH FROM VM1 to VM2 BLOCKED



SCREENSHOT SHOWING THE SSH FROM VM2 to VM1 BLOCKED

Observation from Task2:

The main aim of this task was to implement additional features in the form of kernel modules in Linux and load them dynamically. Kernel Modules are pieces of code that can be loaded and unloaded on-demand at runtime. We write such a kernel module named task2_lkm.c, compile it and install the module using makefile and build the same using the commands used as shown in the above screenshots. The kernel module contains the various firewall rules and when the module is loaded and run, we see that the rules get implemented and the firewall is active thereby filtering all the packets and denying connections as mentioned in the rules which is evident from the screenshots of the executions.

TASK 3: EVADING EGRESS FILTERING

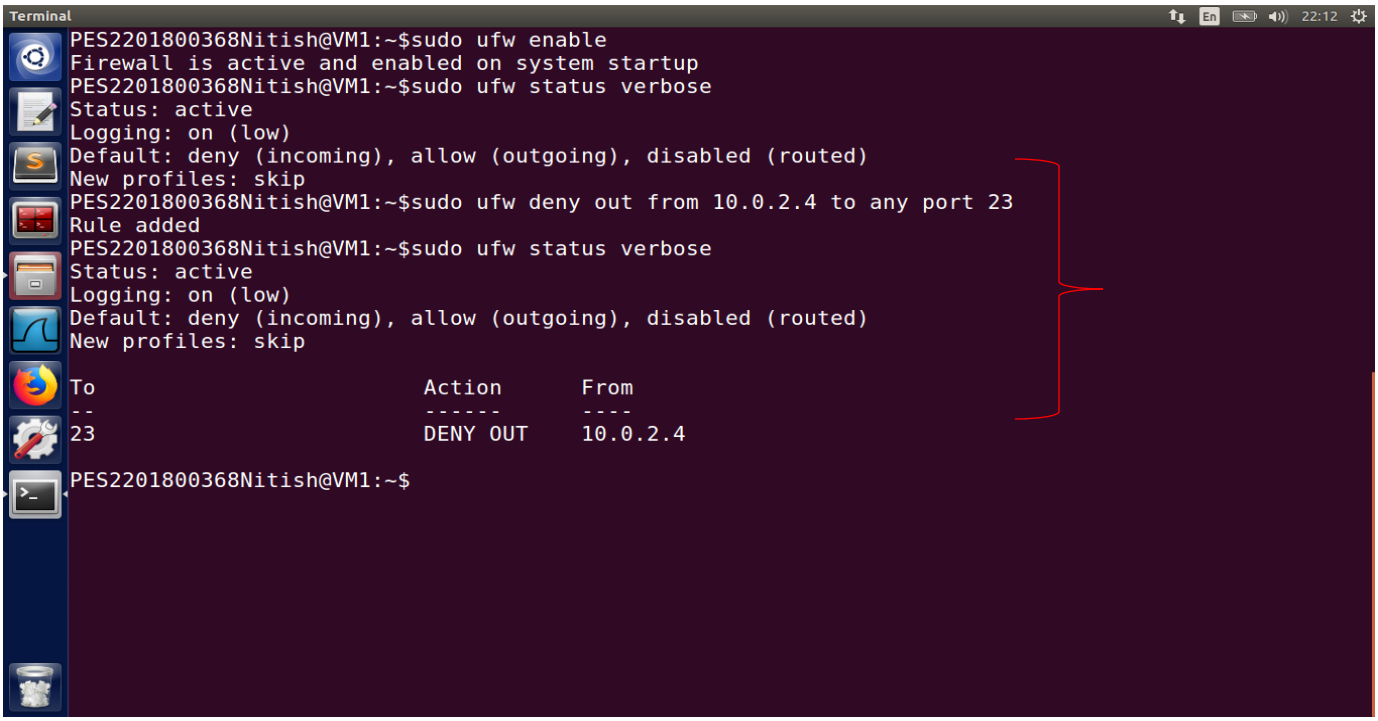
In this task, we will be using a ***ssh*** tunnel to evade egress filtering.

Setup: VM1 = 10.0.2.4 ; VM2 = 10.0.2.5 ; VM3 = 10.0.2.7

TASK 3.A: TELNET TO MACHINE B THROUGH THE FIREWALL

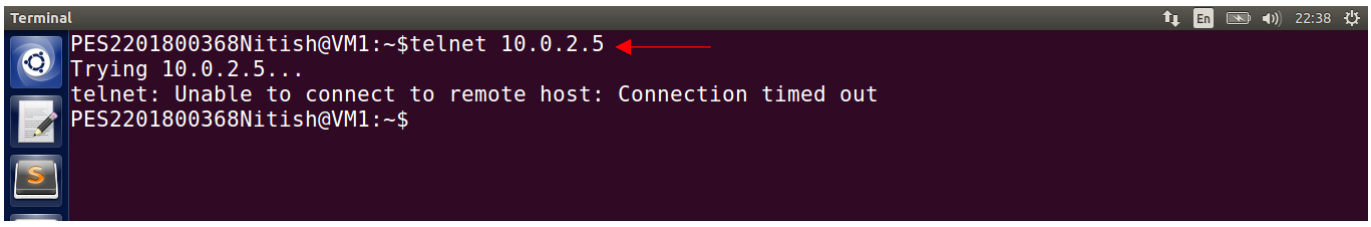
Aim: In this lab, we will use three VMs. VM1 will be blocked from being able to telnet to VM2. We will utilize a ***ssh*** tunnel to allow VM1 to telnet to VM2 via VM3. VM1 is analogous to 'home machine', VM2 to 'work machine' and VM3 to 'apollo machine'.

We first delete all the existing firewall rules on VM1 and add a new rule to prevent VM1 from being able to telnet to any other machine.



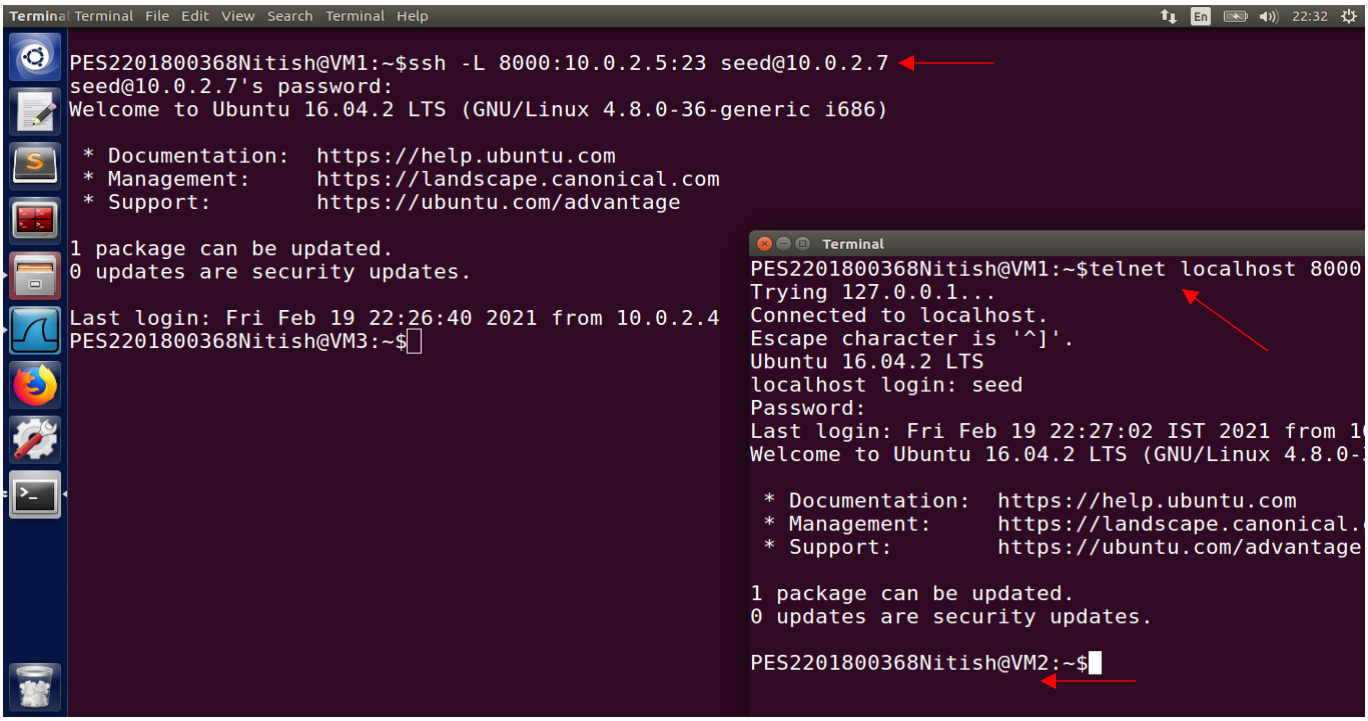
SCREENSHOT SHOWING THE NEWLY ADDED RULES

To verify the effect of the added rule:

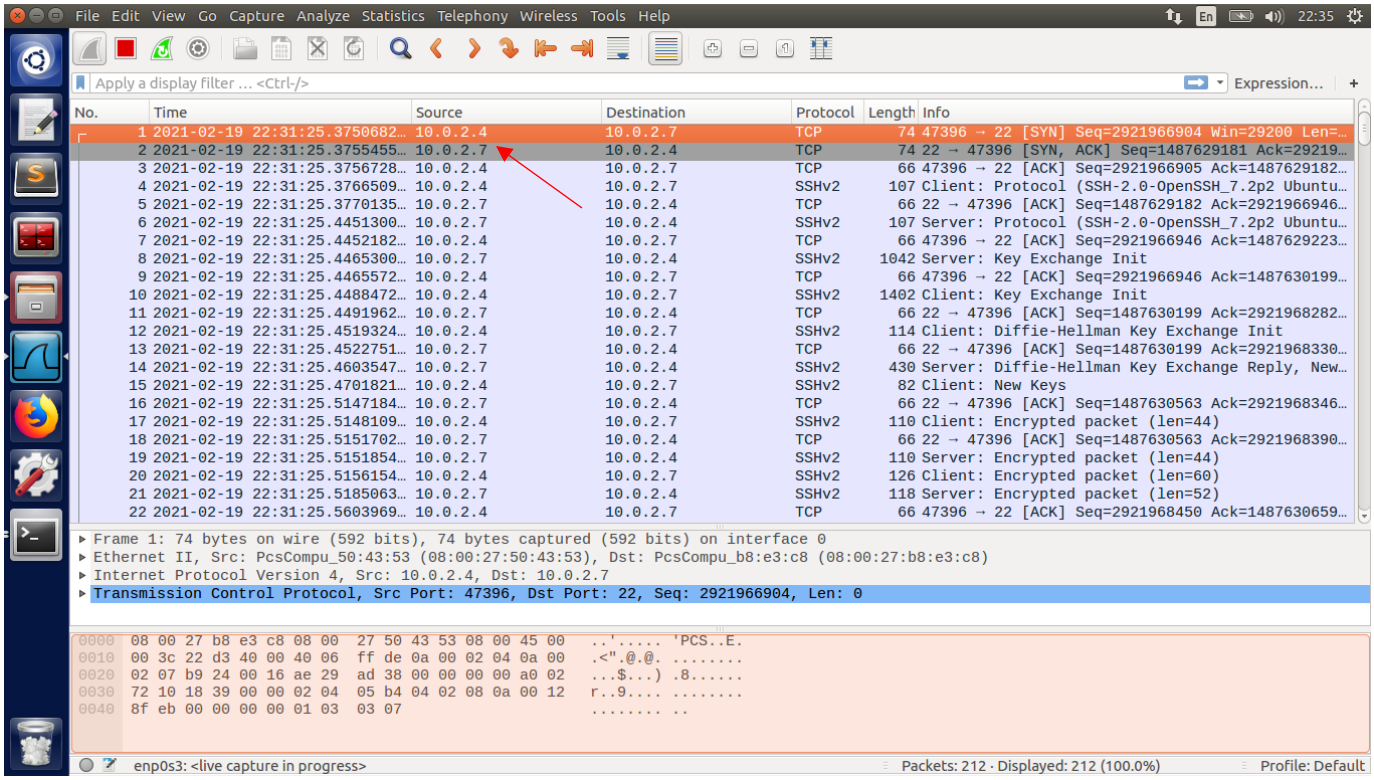


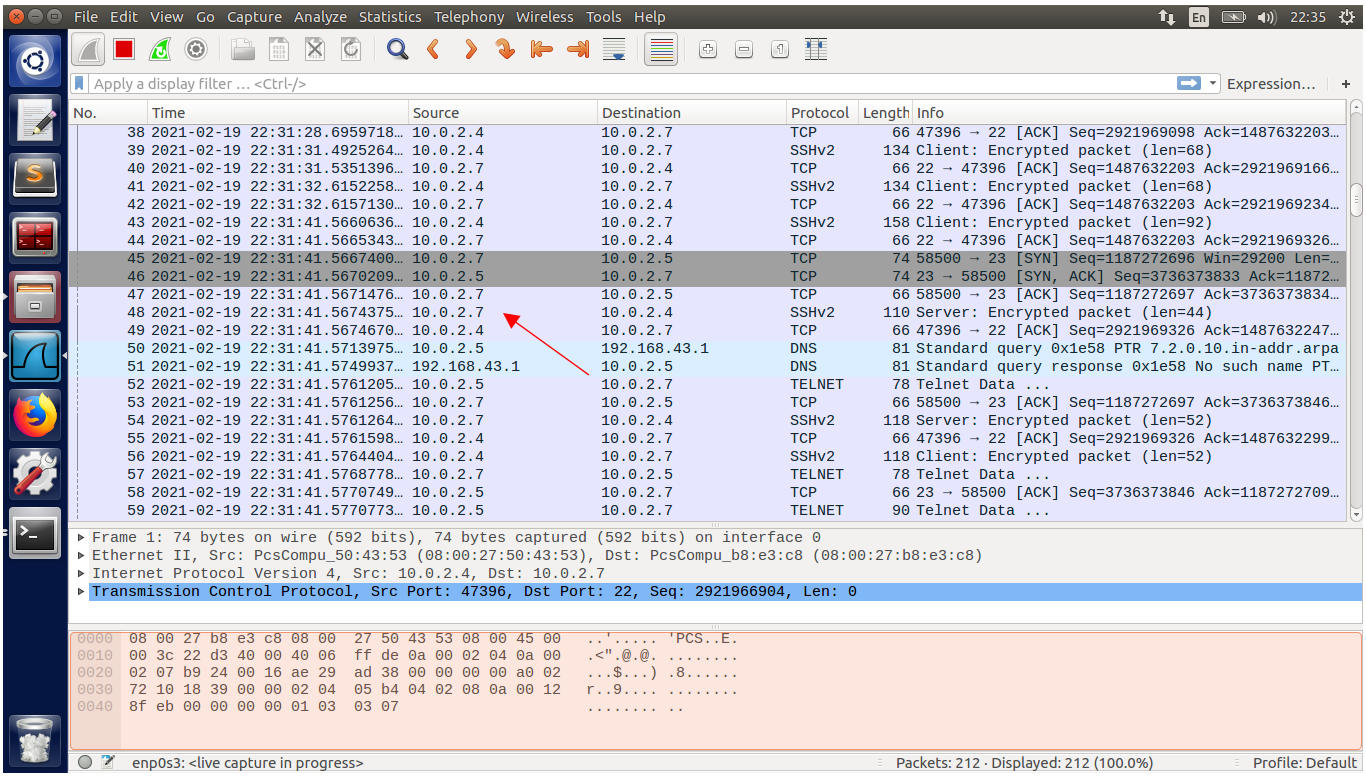
SCREENSHOT SHOWING THE FAILURE OF TELNET FROM VM1 to VM2

We next setup a ***ssh*** tunnel between VM1 and VM2 to allow VM1 to telnet via VM2, evading the firewall on VM1. The ***ssh*** command below allows VM1 to use its local port 8000 to ***telnet*** to VM3 via VM2.



SCREENSHOT SHOWING THE SETTING UP OF ssh TUNNEL and HENCE OBTAINING THE TELNET ACCESS TO VM2 VIA THE ESTABLISHED TUNNEL OF VM3





WIRESHARK CAPTURE

Observations from Task 3a:

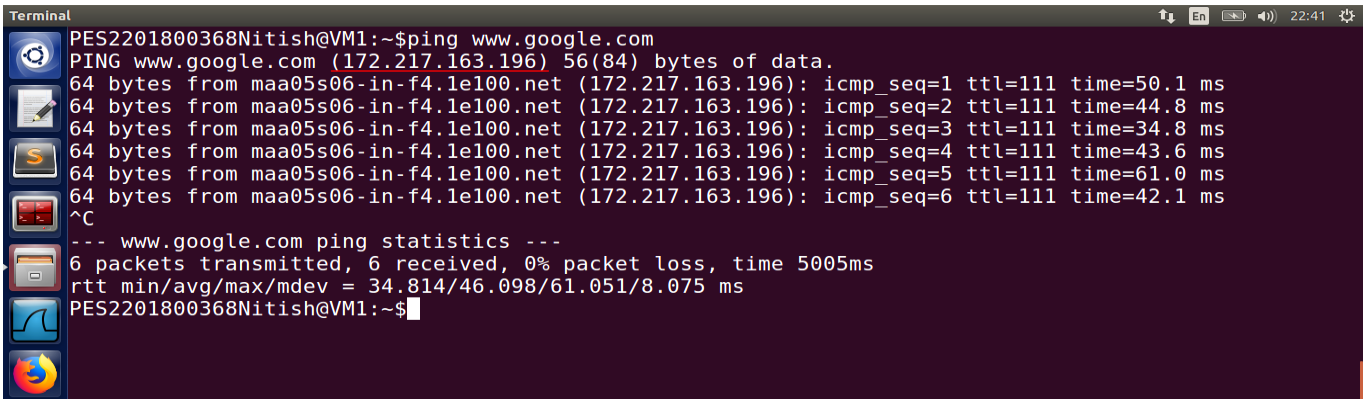
The aim of this task to evade firewalls using SSH tunneling. If there exists a firewall to deny telnet connection from 'home' machine to 'work', then we can build ssh tunnel from 'home' to another machine 'apollo'. This tunnel will forward the TCP data received on port 8000 on 'home' to port 23 on 'work' thereby evading the firewall from 'home' to 'work' and establishing successful telnet connection. This is what is done in the above task and is evidently seen in the screenshots and wireshark capture.

TASK 3.B: CONNECTING TO GOOGLE USING SSH TUNNEL

In this task, we will setup a firewall rule to block VM1 from visiting www.google.com but then leverage dynamic forwarding via ssh tunnel to visit www.google.com from VM1 via VM2.

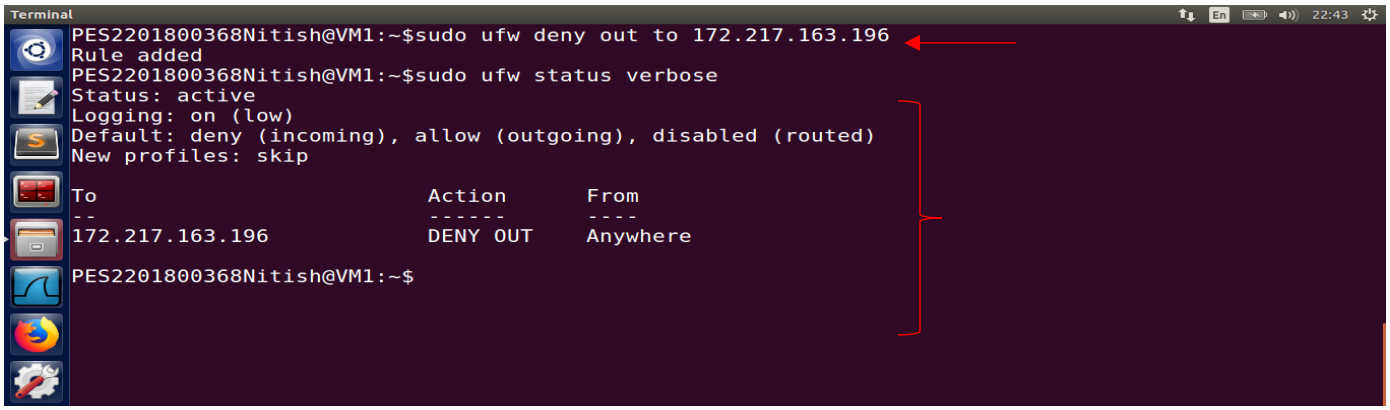
All existing firewall rules are deleted using the 'sudo ufw delete <rule_no>' command.

Obtaining the IP address of www.google.com using ping command



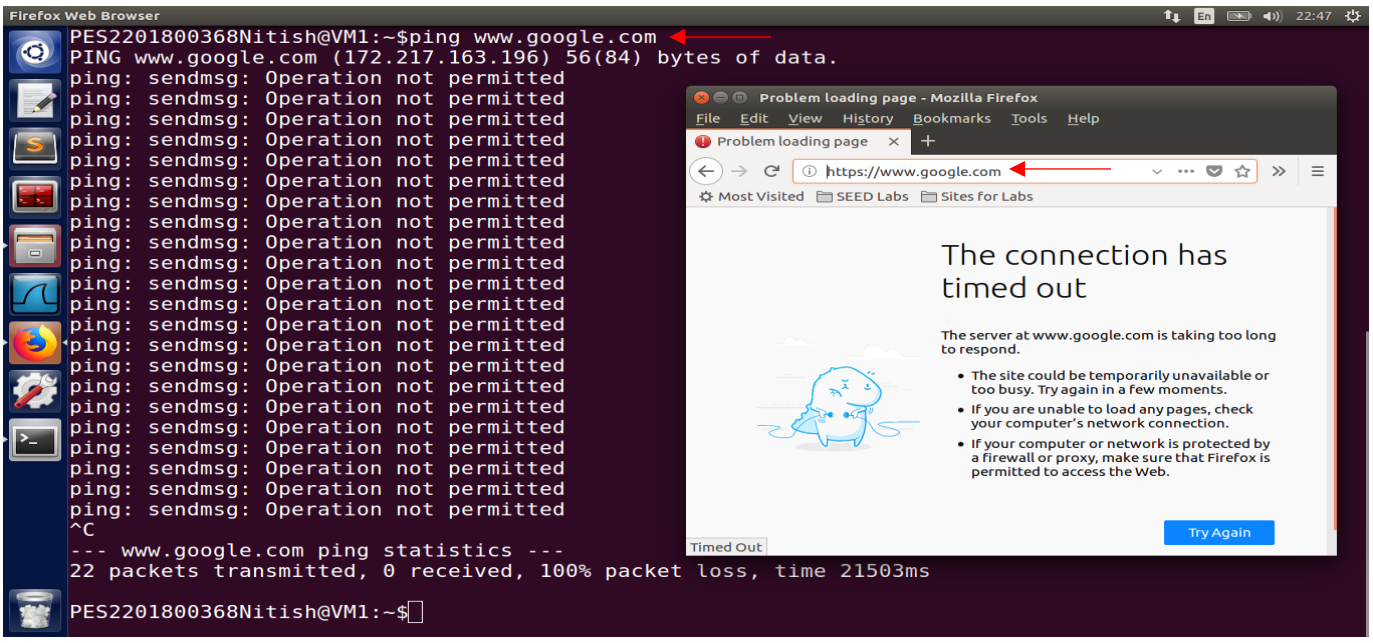
SCREENSHOT SHOWING THE IP ADDRESS OF www.google.com

We can setup a firewall rule to block traffic to that IP.



SCREENSHOT SHOWING THE RULE ADDED

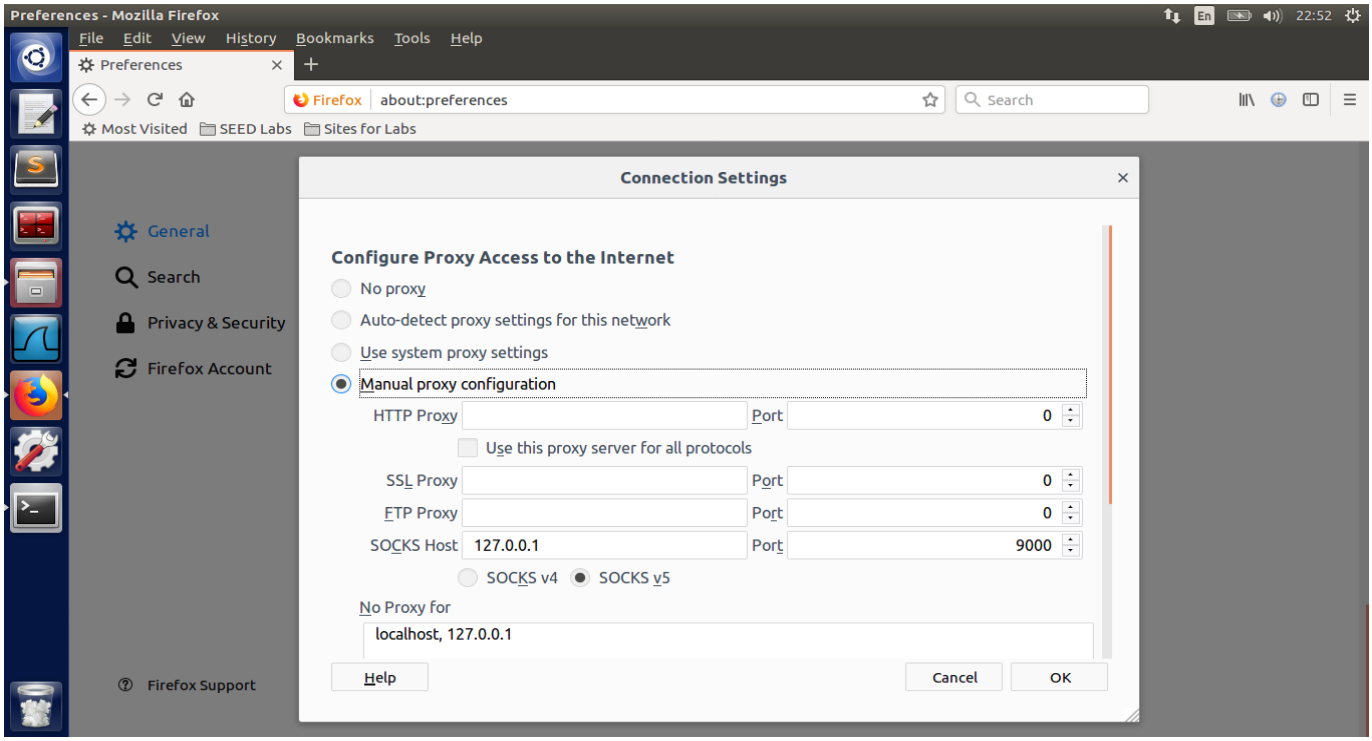
Verifying the added rule



SCREENSHOT SHOWING PING AND BROWSER ACCESS TO www.google.com FAILED

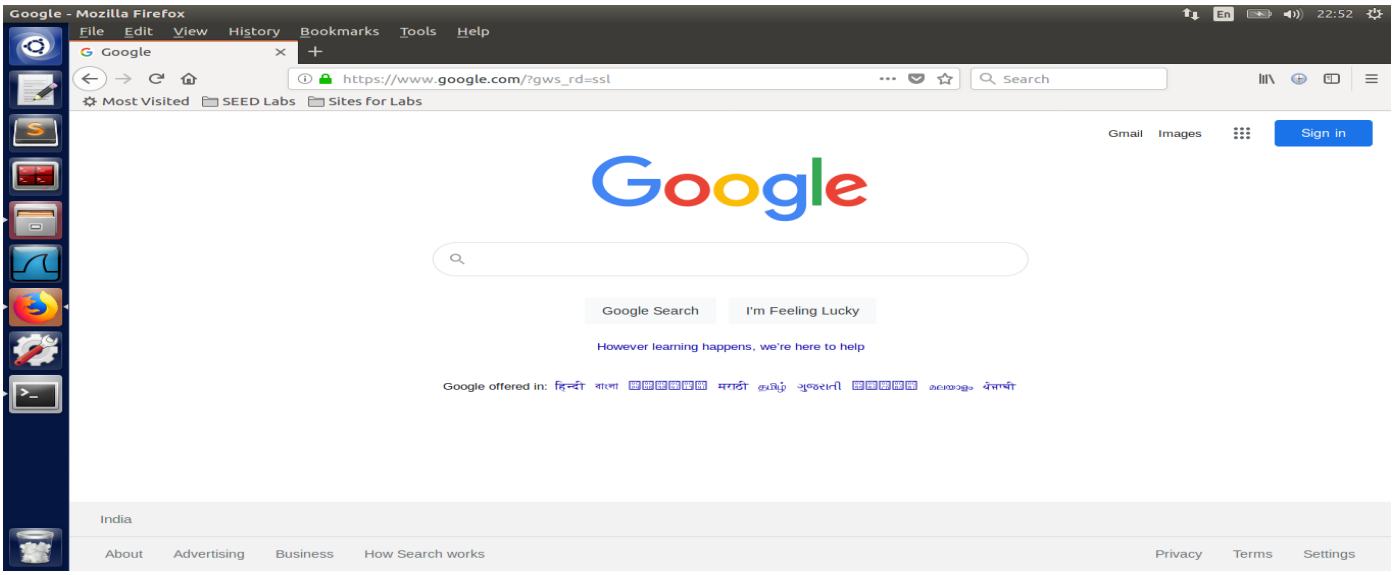
We now setup a ssh tunnel with dynamic port forwarding between VM1 and VM2. With this tunnel setup, VM1 will be able to use its local port 9000 to send a request to www.google.com via VM2.

SCREENSHOT SHOWING THE ESTABLISHING OF TUNNEL TO EVADE THE FIREWALL

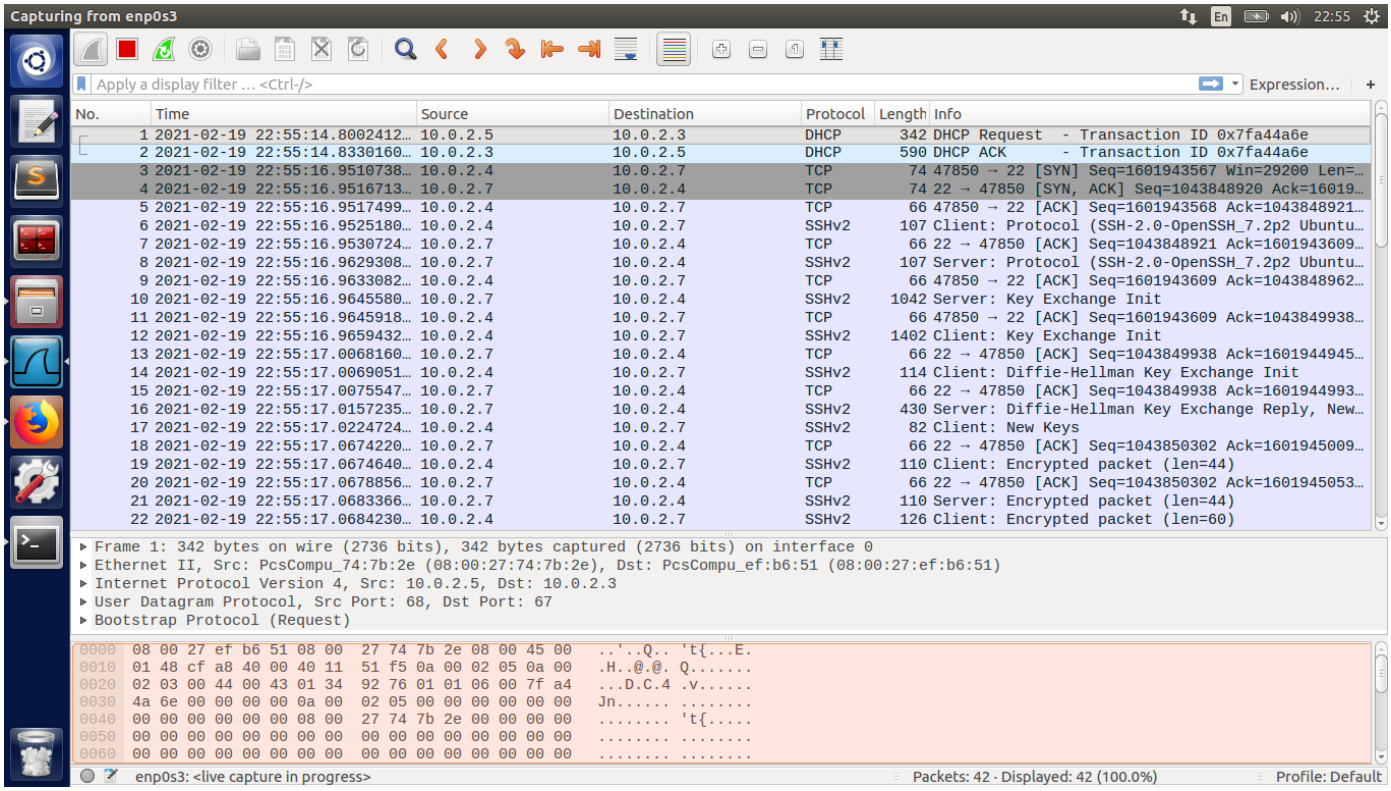


SCREENSHOT SHOWING THE PROXY SETTINGS CONFIGURED TO USE THE ESTABLISHED TUNNEL

Testing the established tunnel



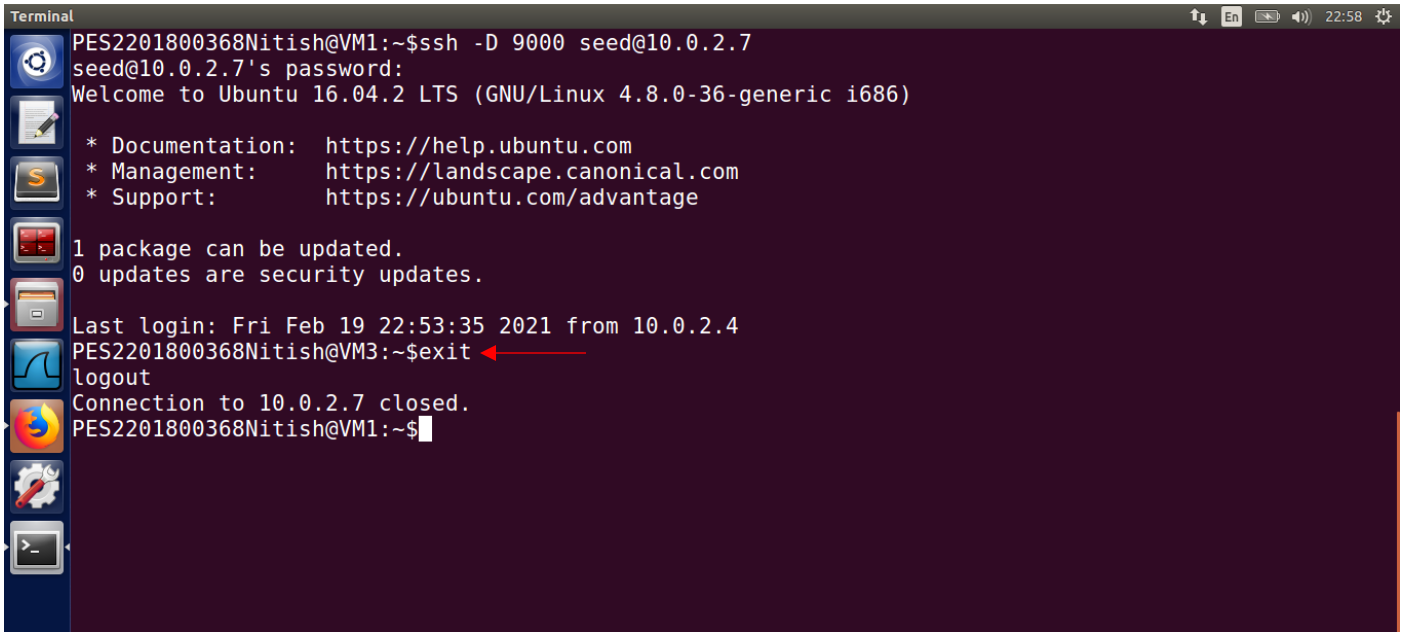
SCREENSHOT SHOWING SUCCESSFUL ACCESS TO www.google.com ON CONFIGURING THE PROXY SETTINGS AND ESTABLISHING A TUNNEL TO EVADE THE FIREWALL



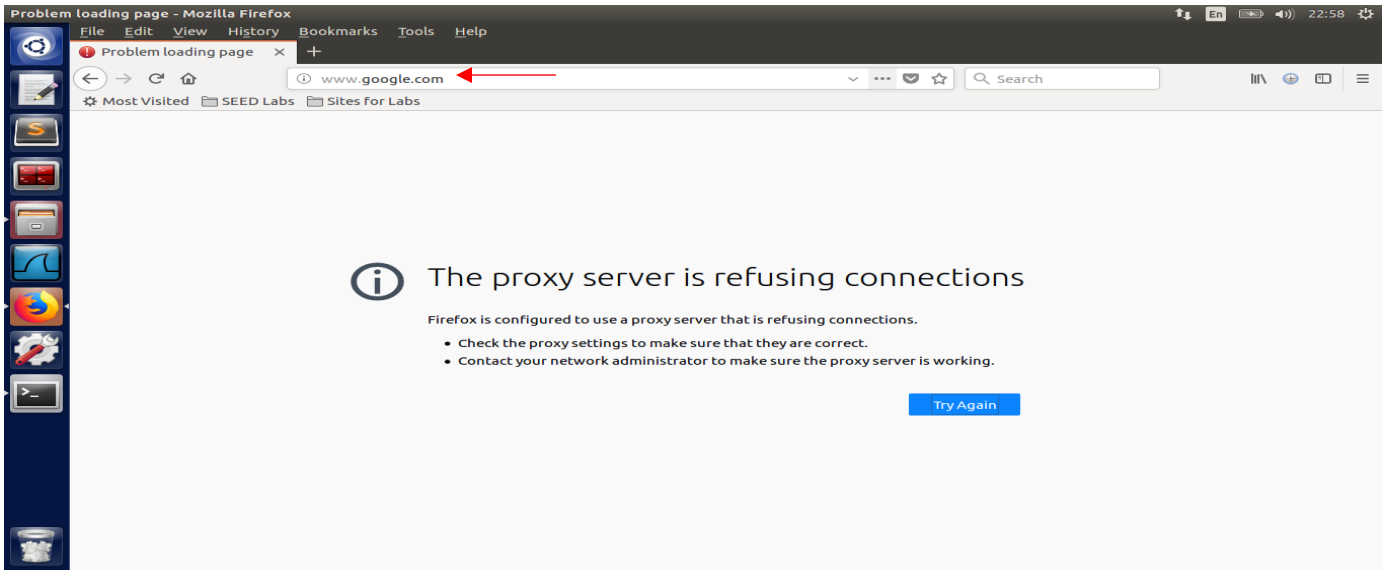
WIRESHARK CAPTURE OF THE SAME

We next disable the tunnel to see its effect. Provide a screenshot of your observations.
We also clear the browser cache to make sure firefox does not show the webpage it just loaded.

If we try to visit www.google.com again, we observe that the browser informs the proxy is refusing connections. The browser is still configured to use proxy, but with the tunnel not running any more, the browser cannot use local port 9000 to get the result.



SCREENSHOT SHOWING THE DISABLING OF THE TUNNEL



SCREENSHOT SHOWING THE RESULT OF DISABLING THE TUNNEL AND RETAINING THE PROXY SETTINGS

On re-enabling the tunnel and retaining the proxy settings, we once again obtain the www.google.com page. If the tunnel is established but the proxy settings are not configured (i.e. left to the default option which is “use system proxy settings”), we once again get a connection timed out message and the webpage isn’t obtained.

Observation from Task 3b:

The above task is another example of evading egress firewall. We have shown above a situation where a firewall rule is configured to prevent access to a website www.google.com from the machine. But in order to evade this firewall and access the website, we establish a ssh tunnel using the command used above and also configure the proxy settings in the browser to use the tunnel. We observe from the screenshots that only when both the tunnel is established and the proxy settings are configured, will the access to browser happen and either one of them not done, will lead to failure of connecting to the website. This method is helpful when the companies or organizations block access to certain websites within the limits using a firewall and this technique can be used to evade the firewall and ensure access to the website.

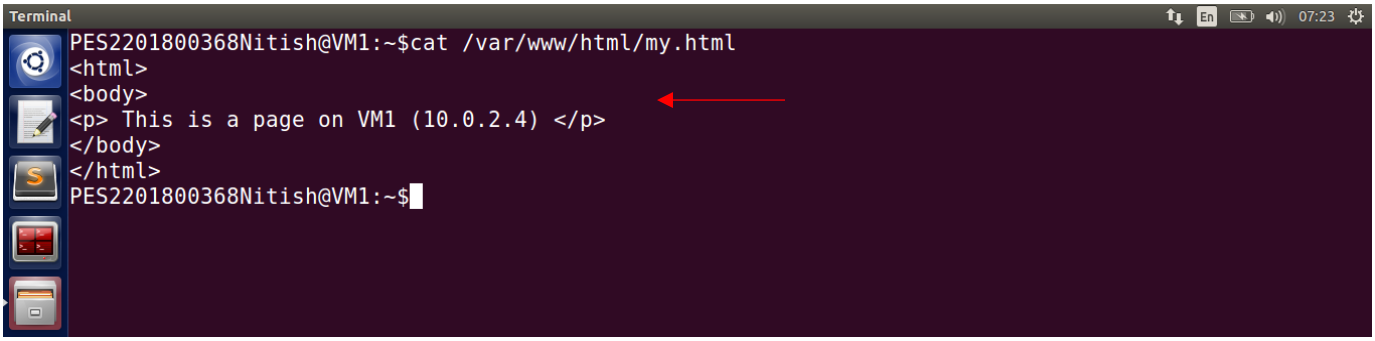
TASK 4: EVADE INGRESS FILTERING

In this task, we will block incoming port 80 and port 22 on VM1, but still access a web page on the web server in VM1 from VM3 by using a reverse ***ssh*** tunnel.

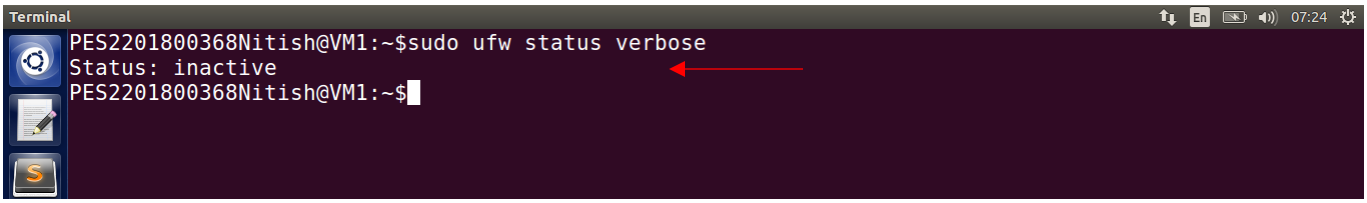
Setup: VM1 = 10.0.2.4 ; VM3 = 10.0.2.7

We firstly delete all the previously configured firewall rules.

Our goal is to access a secret page on VM1 (my.html) from VM3. The content of the page is shown below:

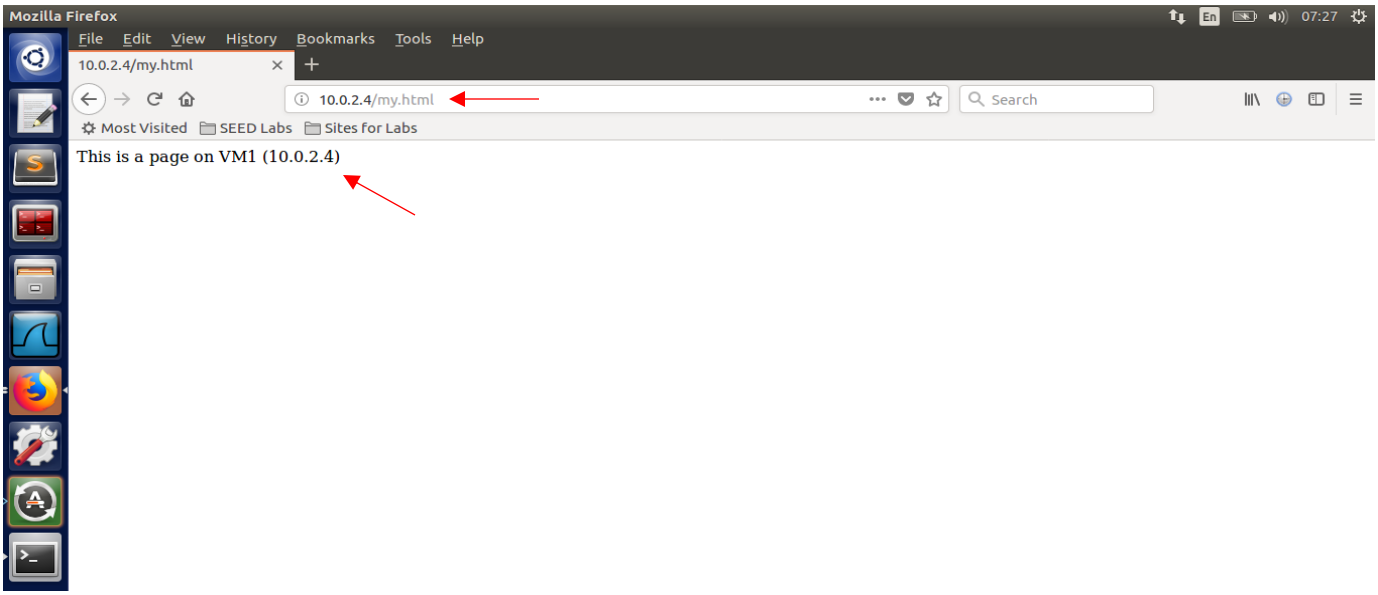


SCREENSHOT SHOWING THE my.html WEBPAGE



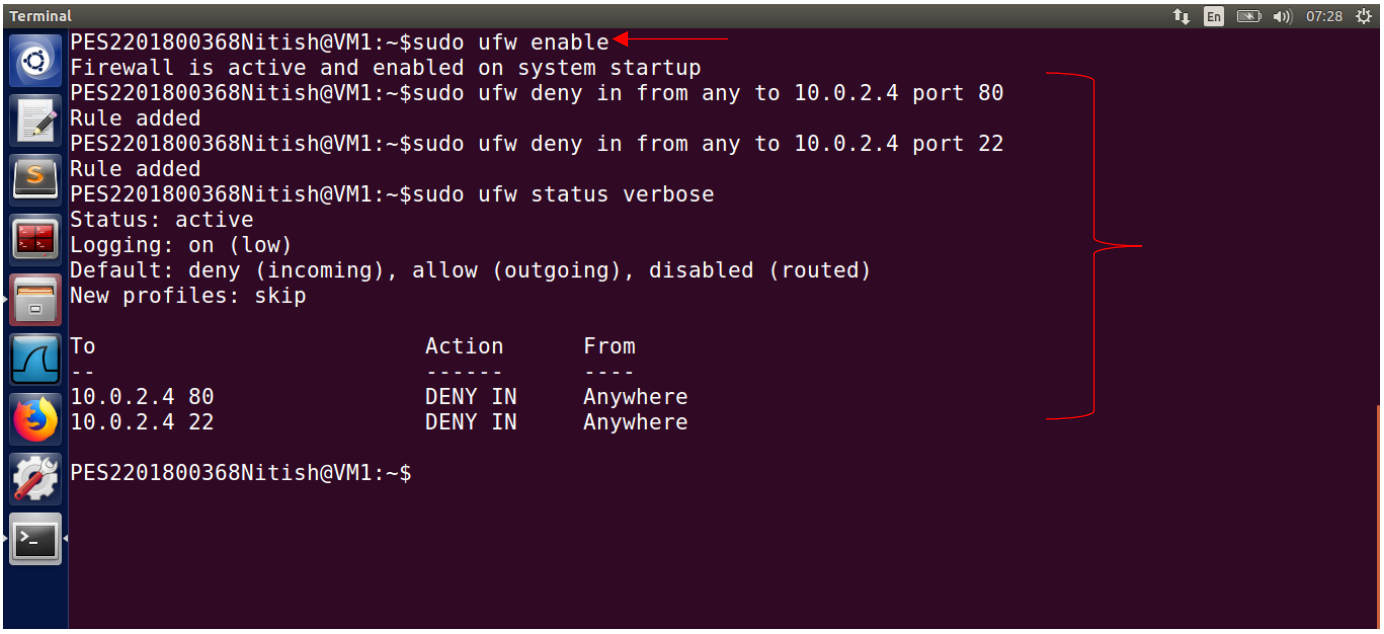
SCREENSHOT SHOWING NO RULES CONFIGURED IN THE FIREWALL

We now verify that the website is accessible from VM3 when there are no rules configured on the firewall.



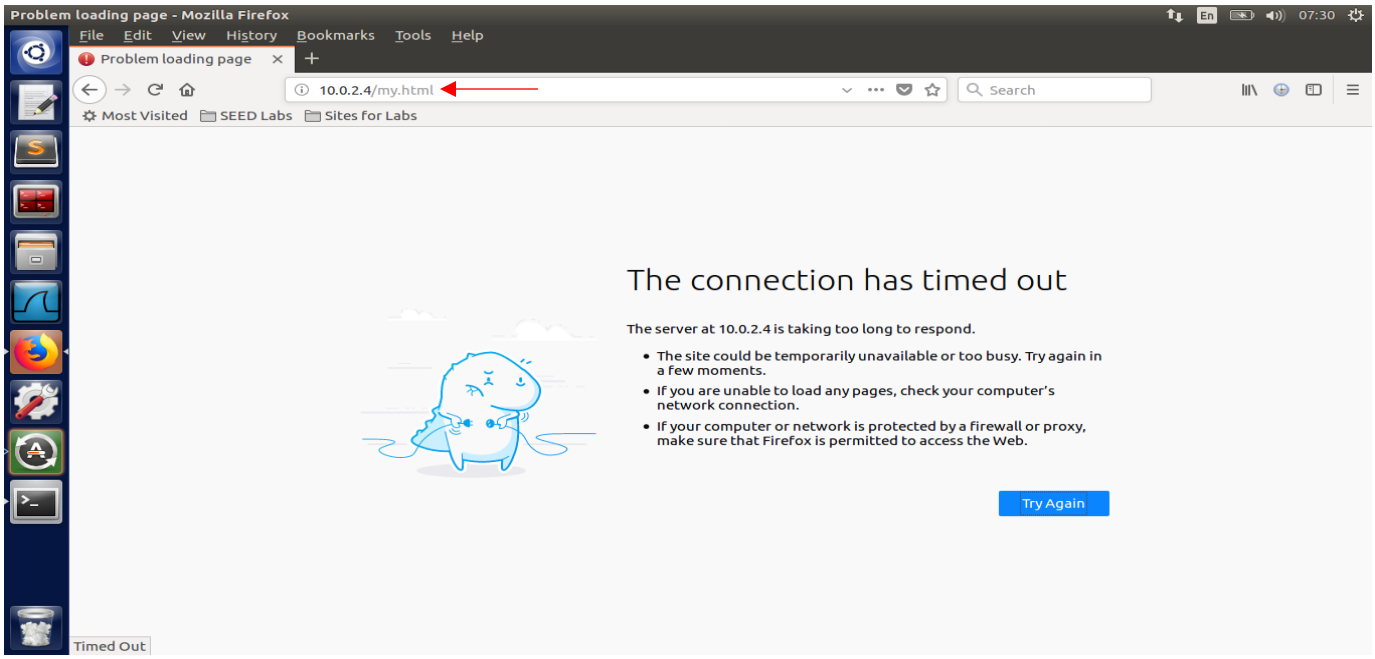
SCREENSHOT SHOWING THE WEBPAGE FROM VM1 ACCESSIBLE FROM VM3

We next block incoming requests on port 80 and port 22 on VM1.



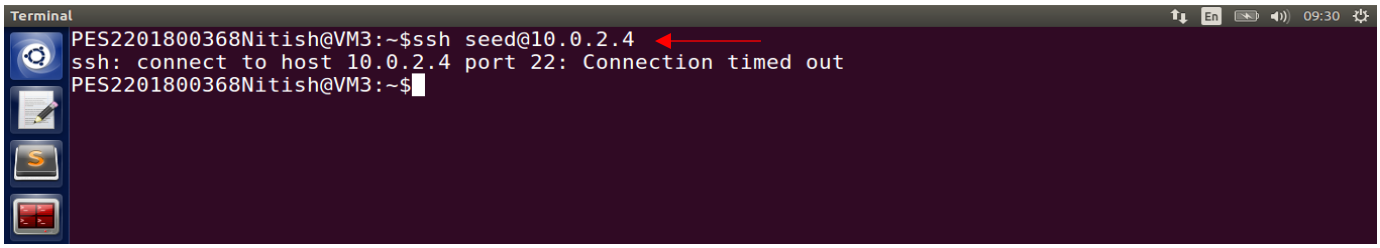
SCREENSHOT SHOWING THE NEWLY ADDED RULES

Given the firewall rules in place on VM1, we next check if the page is still accessible. We clear the browser cache.
If we try to access the page again from VM3, the page is no longer accessible.



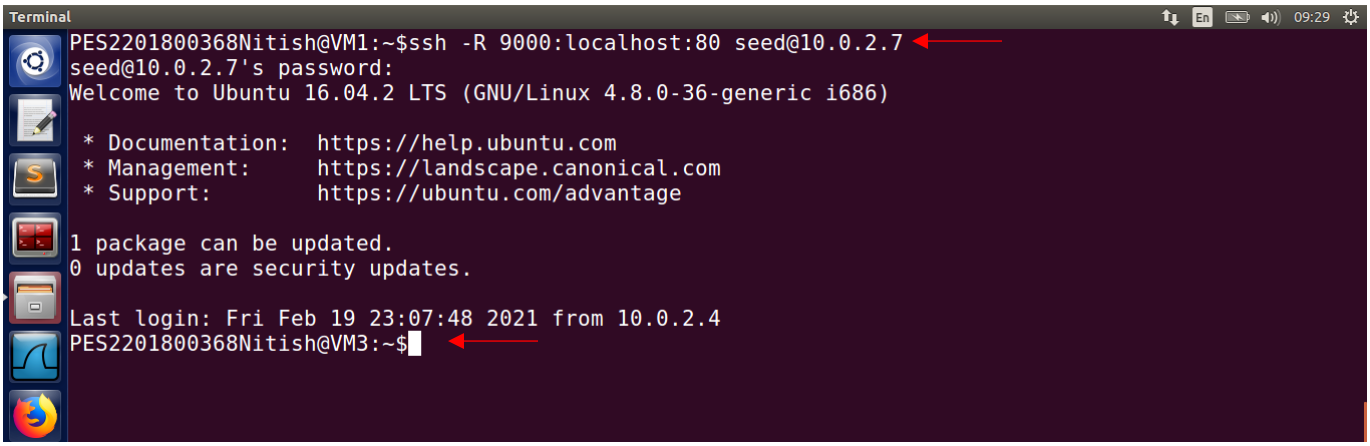
SCREENSHOT SHOWING CONNECTION TIMED OUT ERROR WHEN THE WEBPAGE IS ACCESSED

Also, because port 22 is blocked, VM3 cannot ***ssh*** to connect to VM1.



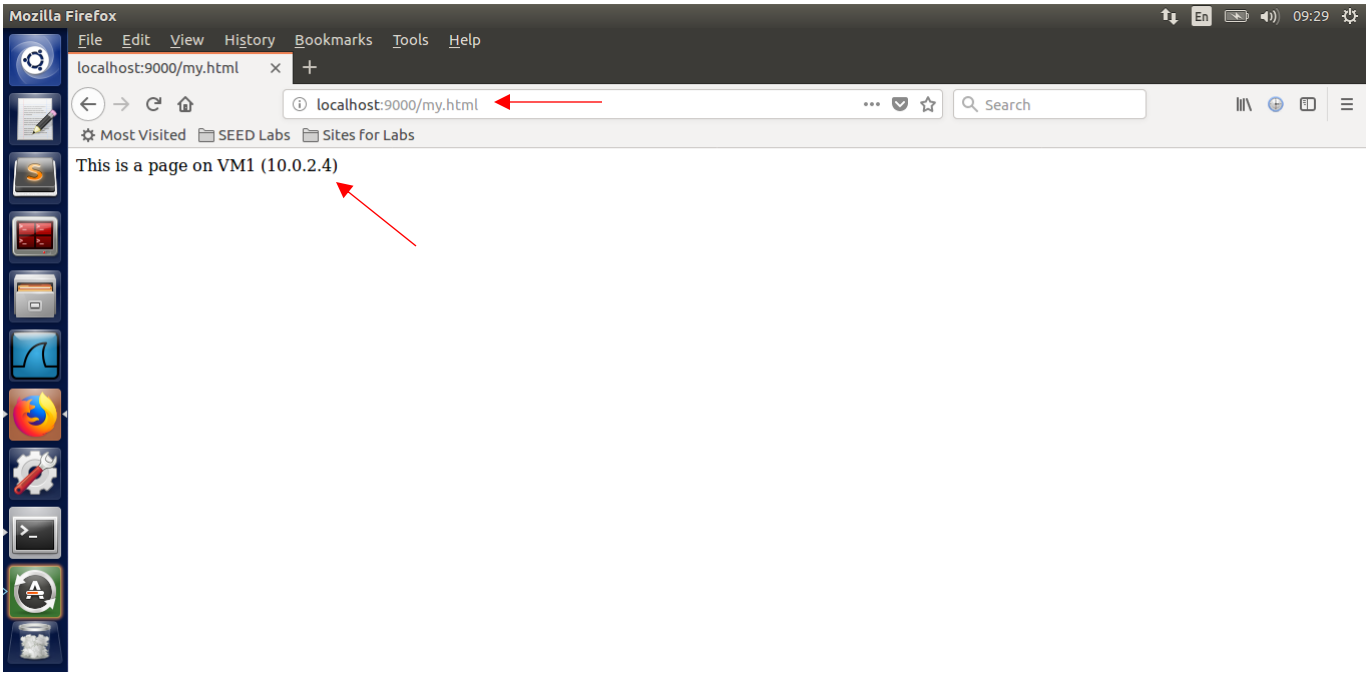
SCREENSHOT SHOWING FAILED ssh CONNECTION TO VM1 DUE TO FIREWALL RULE

We next set up a reverse tunnel. Using this VM3 can use its local port 9000 to access port 80 on VM1.



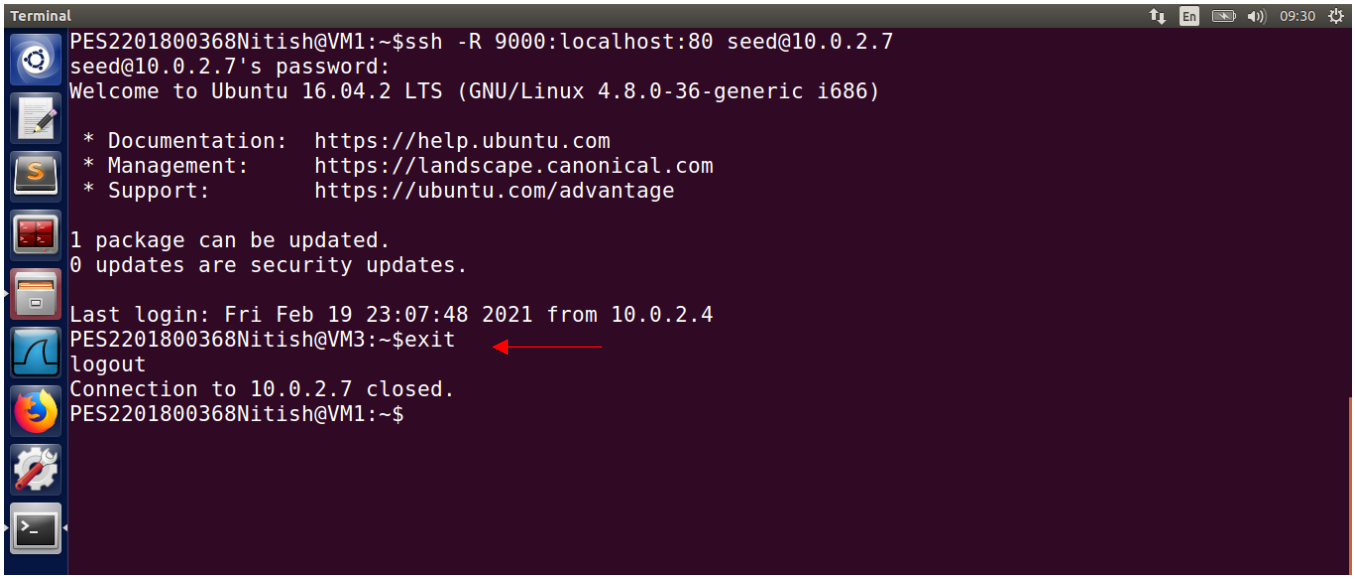
SCREENSHOT SHOWING THE ESTABLISHED REVERSE TUNNEL

With the tunnel setup, we test whether we can access the webpage using port 9000 on VM3.



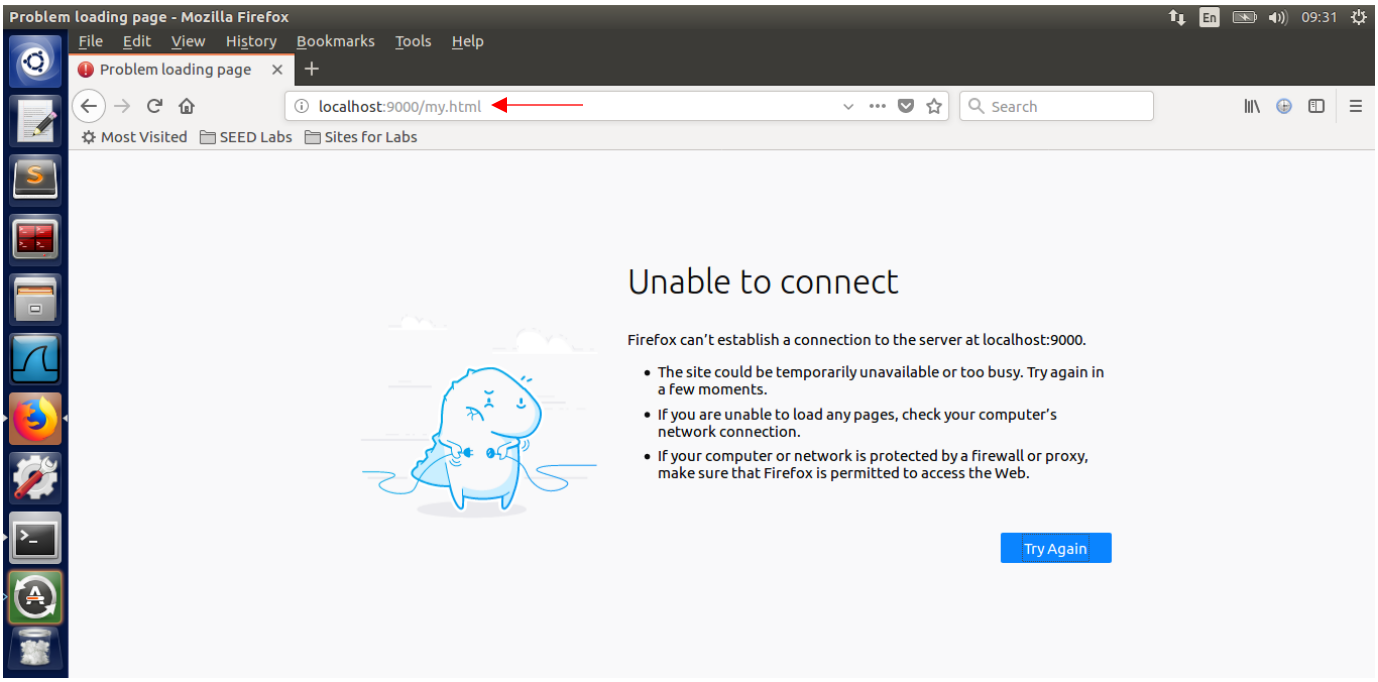
SCREENSHOT SHOWING THE WEBPAGE ACCESS AVAILABLE ON PORT 9000 DUE TO THE ESTABLISHED REVERSE TUNNEL

We next break the tunnel to see its effect (on VM1)



SCREENSHOT SHOWING THE BREAKING OF THE ESTABLISHED REVERSE TUNNEL

Testing the effect of breaking the reverse tunnel



SCREENSHOT SHOWING FAILURE TO ACCESS THE WEBPAGE FROM VM3 DUE TO BREAKING OF THE REVERSE TUNNEL

Observation from Task4:

The aim of the above task was to evade the firewall preventing access to an internal website for which even incoming ssh traffic is blocked. Therefore in order to evade this firewall we create a ssh tunnel from outside and use this tunnel to do the port forwarding. We create a special type of ssh tunnel called the reverse tunnel, so when users from outside send a HTTP request to port 9000 from their browsers, the ssh tunnel will forward the request to the ssh client, which further forwards the request to port 80 of the web-server machine having the webpage and thereby the webpage can be access by outsiders.

Without this technique of establishing a reverse tunnel, we cannot access internal webpages of an organization protected by firewall.