**IS LAB ASSIGNMENT#05**

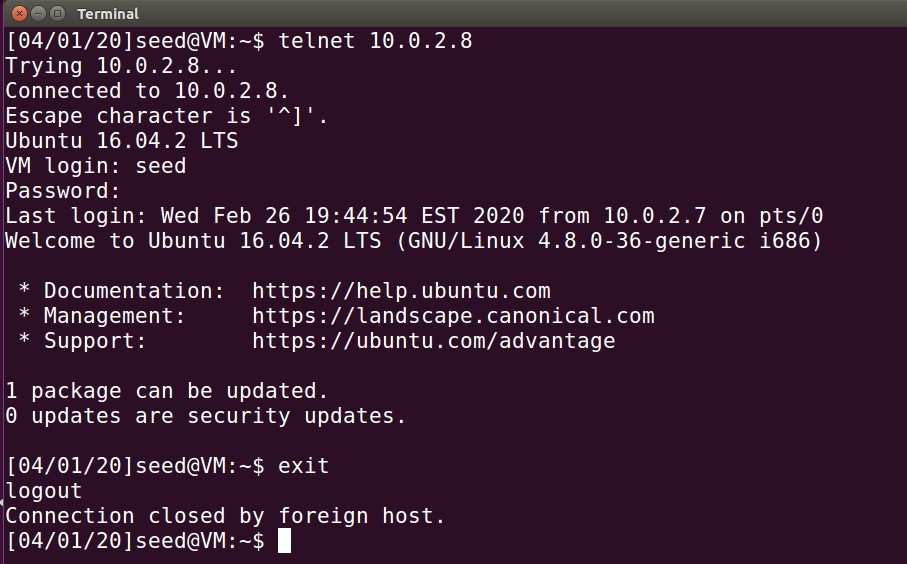
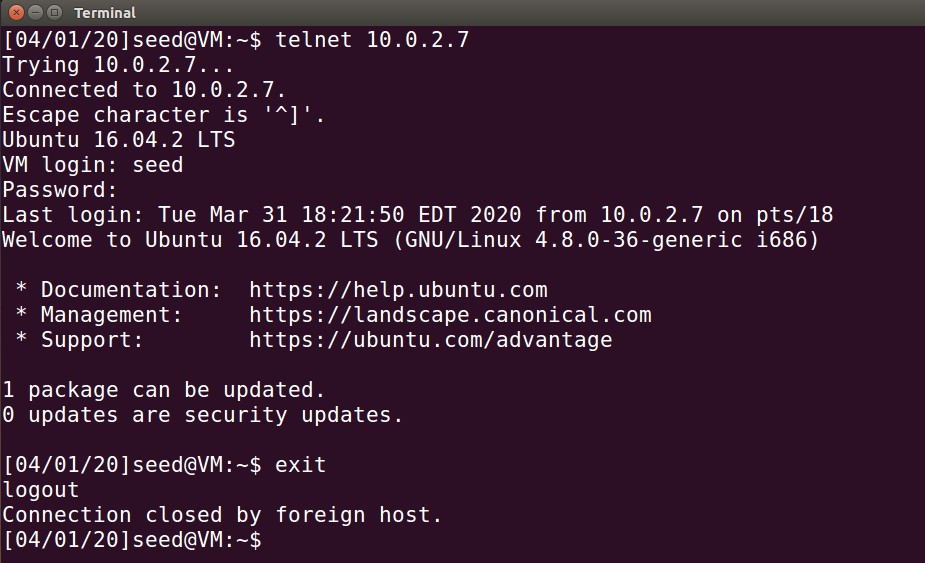
**Name: Saman Khan**

**ID: 19K-0354**

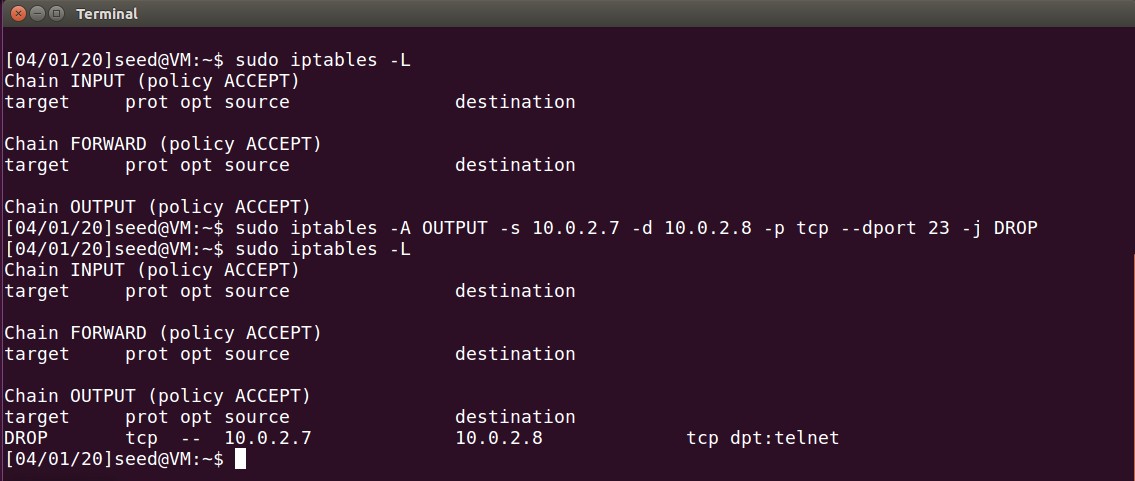
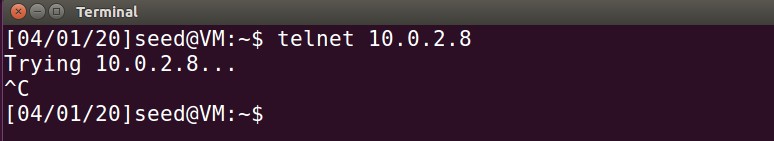
**Section: 7H**

**Task#01: Using Firewall**

In this task, we are going to set up some firewall policies and then observe how the system behaves both before and after the policies take effect. Machine A is the one that serves as the firewall. Before we install the firewall, we check to see if Machines A and B are able to telnet to one another in the following manner:

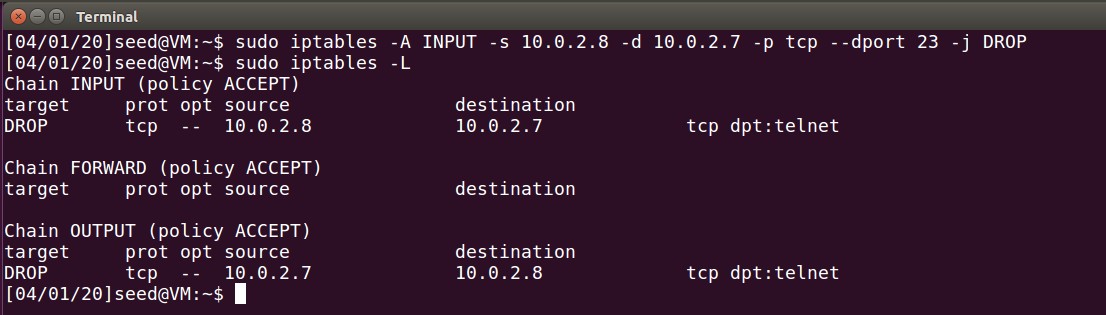
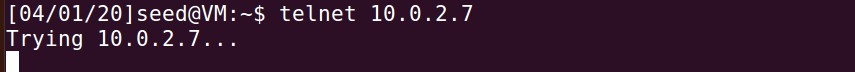
Machine A to Machine B  Machine B to Machine A

* **Preventing A from doing telnet to Machine B**

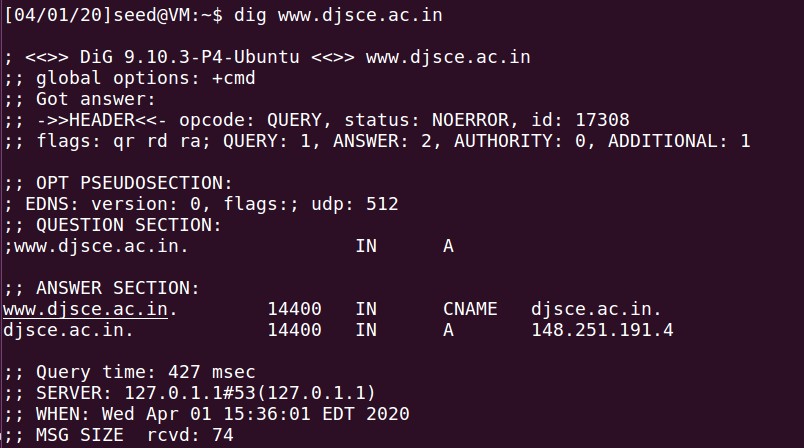
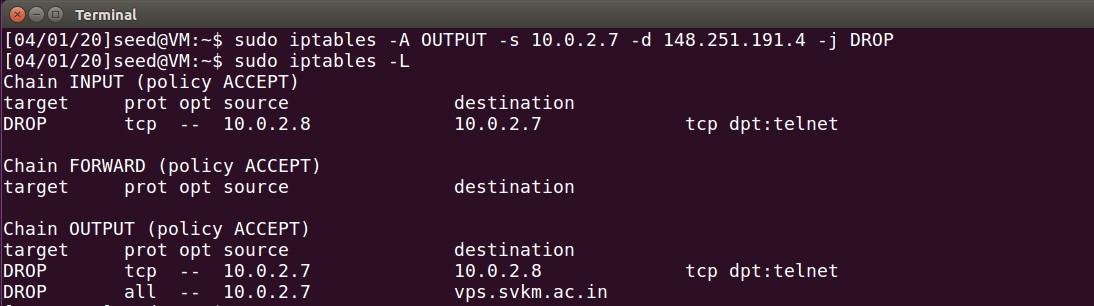
We use “iptables” to set up a personal firewall to prevent A from performing telnet. On trying to telnet to Machine B, we see that the telnet connection is unsuccessful.

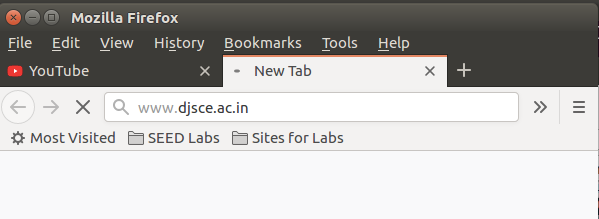
* **Preventing B from doing telnet to Machine A**

This time we perform the same operation and use “iptables” to set up a personal firewall to prevent B from performing telnet and then on trying to telnet to Machine A, we see that it was unsuccessful.



* **Preventing A from visiting an external website**

We will now create a rule in the firewall that will block Machine A from accessing a website hosted elsewhere. To better illustrate our point, please visit www.djsce.ac.in, the website that we have chosen to use. To begin, we have this website's unique IP address obtained. Now we will create a rule that will prevent any traffic from travelling to the IP address of the website. I decided to ban the IP address entirely rather than simply the HTTP and HTTPS traffic it generated. This assists in entirely banning the website from being accessed. It has come to

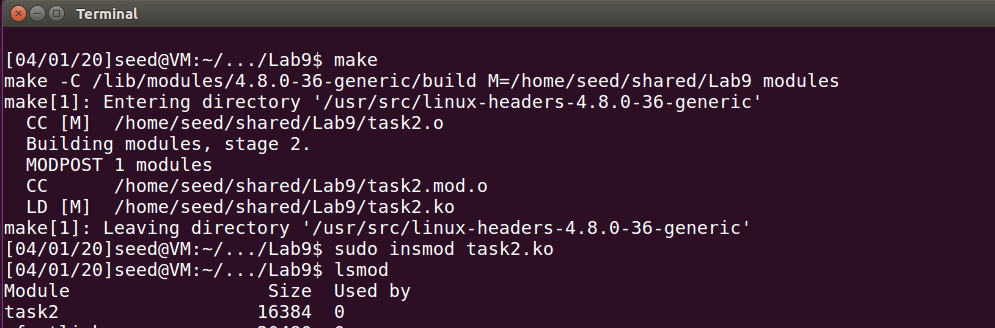
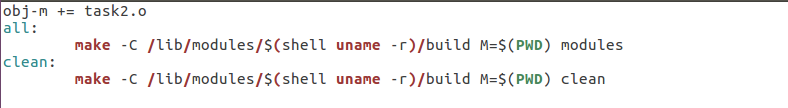


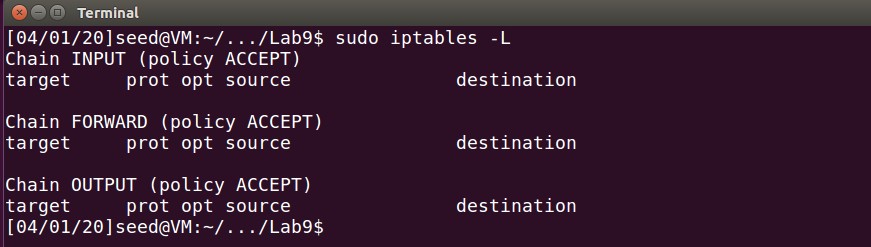
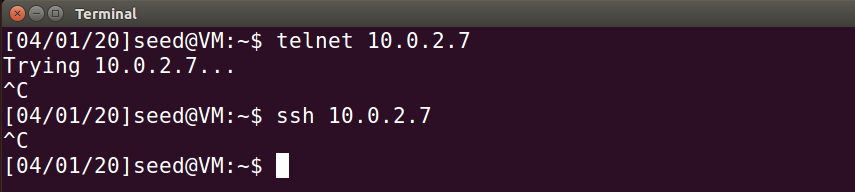
**Task#02: Implementing a Simple Firewall**

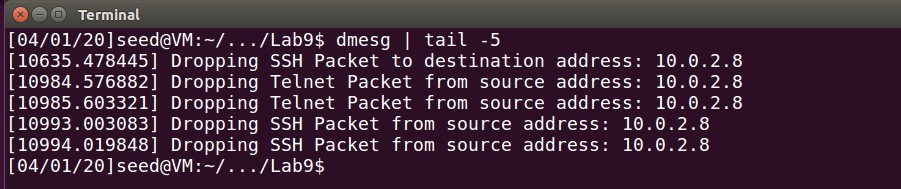
In this task, we will be implementing the packet filtering module by utilizing LKM and Netfilter. For the purpose of determining whether or not packets should be stopped, we add the following policies:

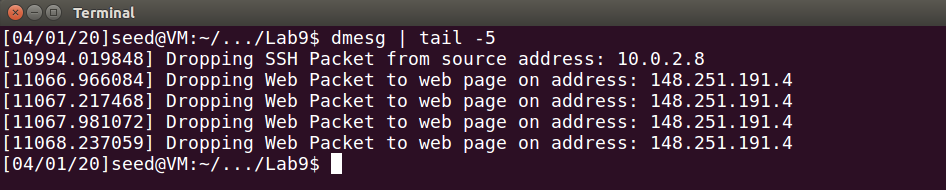
* Prevent telnet traffic from A to B.
* Prevent telnet traffic from B to A.
* Prevent access to external websites from within A
* Prevent SSH connections from A to B
* Prevent SSH connections from B to A

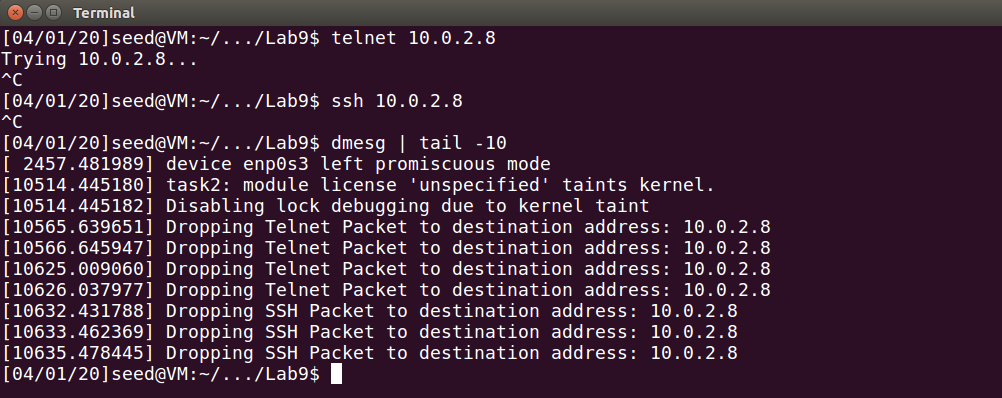
You'll find an attachment below containing both the C code for this task as well as the Makefile that builds the C programme into a loadable kernel module. It is clear to see that the module was compiled correctly and loaded into the kernel successfully. Before we continue, we will make sure that the firewall rules that were configured in Task 1 have been removed. This will allow the effect of the program's firewall rules to be seen. After that, we put the program's firewall policies through their paces by testing them. When we try to make a connection between Machine A and Machine B using telnet and SSH, we discover that the packets are lost, and we are unable to create a connection. When we try to connect Machine B to Machine A using either telnet or SSH, we get the same result both times. It has come to our attention that the connection was not successfully created on Machine B. When we looked through the logs in the kernel, we saw that both telnet and SSH packets were being dropped. This suggests that the application is preventing communication between Machine A and Machine B via telnet and SSH protocols. Now, in order to demonstrate that access to the website has been restricted, we make an effort to load the website using the Firefox browser; however, it does not load. In the process of checking through the kernel logs, we discovered that the packets destined for this website are also being dropped. The conclusion that can be drawn from this is that the firewall rules that were configured in the application are successful.

 ****

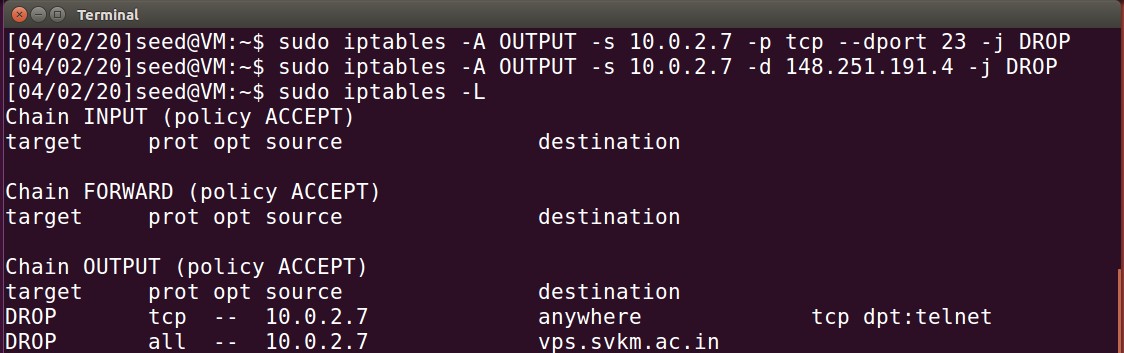




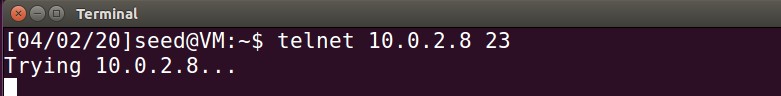
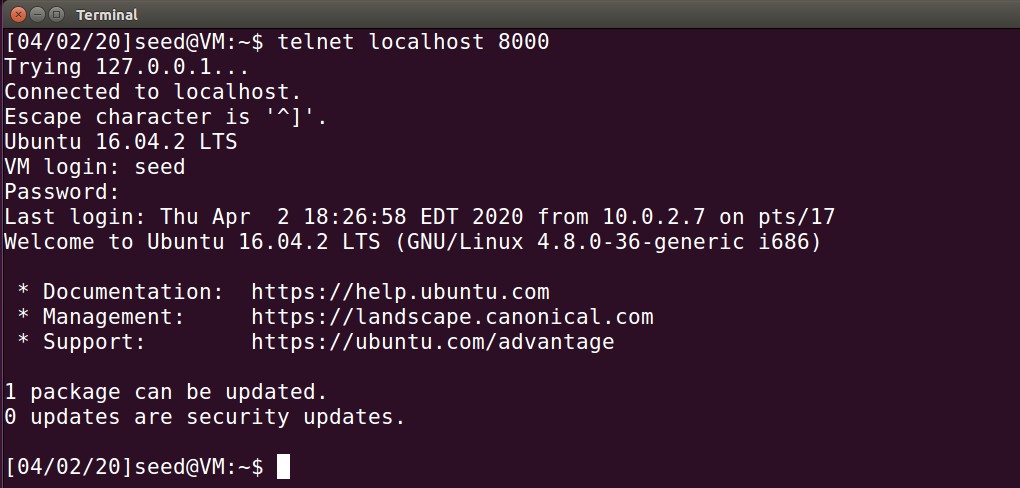


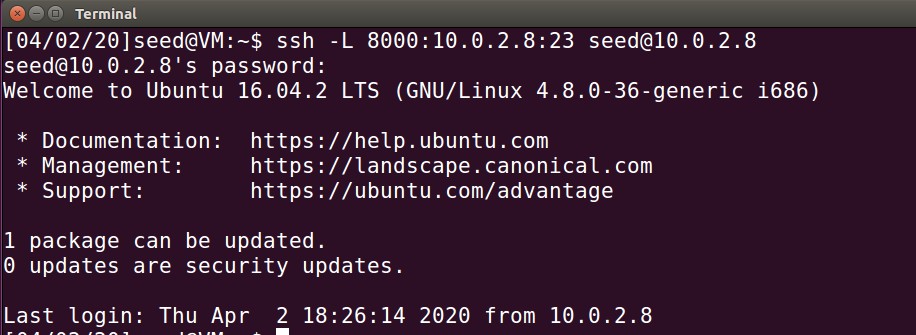
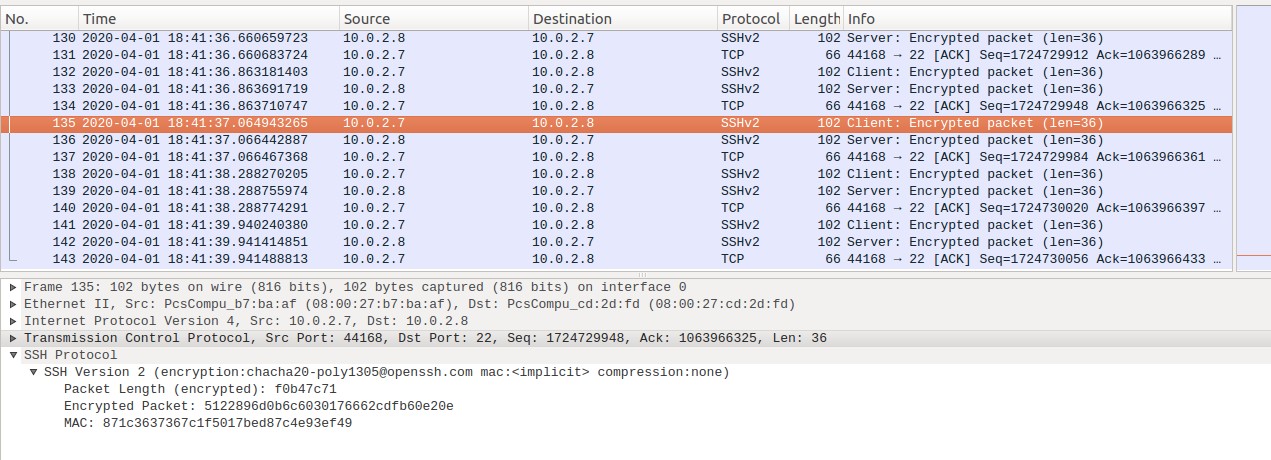


**Task#03: Evading Egress Filtering**

By utilizing SSH tunnels, we are able to get around the egress filtering that is being applied in this task. We have configured Machine A with the following rules, which will prevent access to the www.djsce.ac.in website as well as outbound traffic to external telnet servers.

**Task#03.A: Telnet to Machine B through the Firewall**

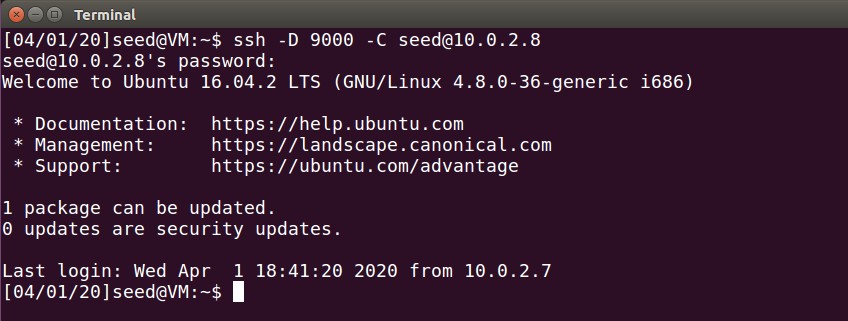
Upon attempting to telnet to Machine B, we observe that it fails, as expected. To circumvent the firewall, an SSH tunnel is created between Machine A and B. The following command creates an SSH tunnel between the localhost (port 8000) and Machine B (using the default port 22), and when packets leave Machine B, they are forwarded to port 23 on Machine B. The tunnel receives TCP packets from the telnet client on Machine A. Upon receiving this packet, the tunnel forwards TCP packets to port 22 on Machine B. Here, the received data is packed into a new TCP packet and sent to port 23 on Machine B. Since the firewall only sees SSH traffic and not telnet traffic, this SSH tunnel can be used to circumvent the firewall rule prohibiting telnet connections. The Wiring capture demonstrates the same. The packet capture reveals only SSH packets and no telnet packets, proving the point. Therefore, when we telnet to localhost 8000, we are essentially telnetting to host 10.0.2.8 – port 23 through the SSH tunnel. This circumvents the firewall's restriction on outgoing telnet connections.

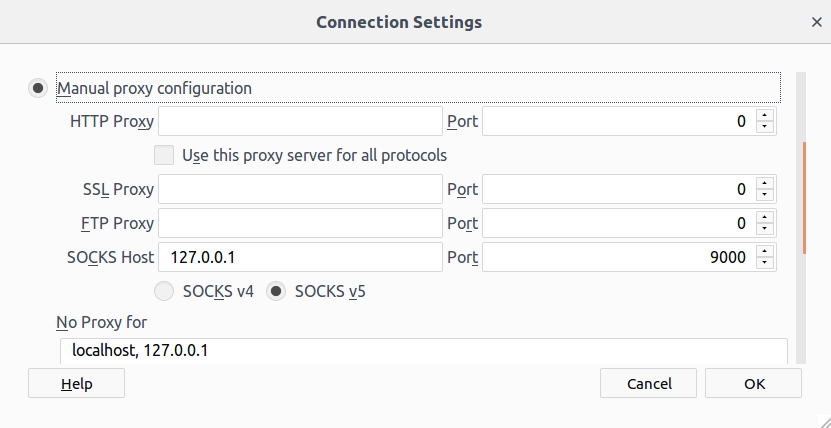
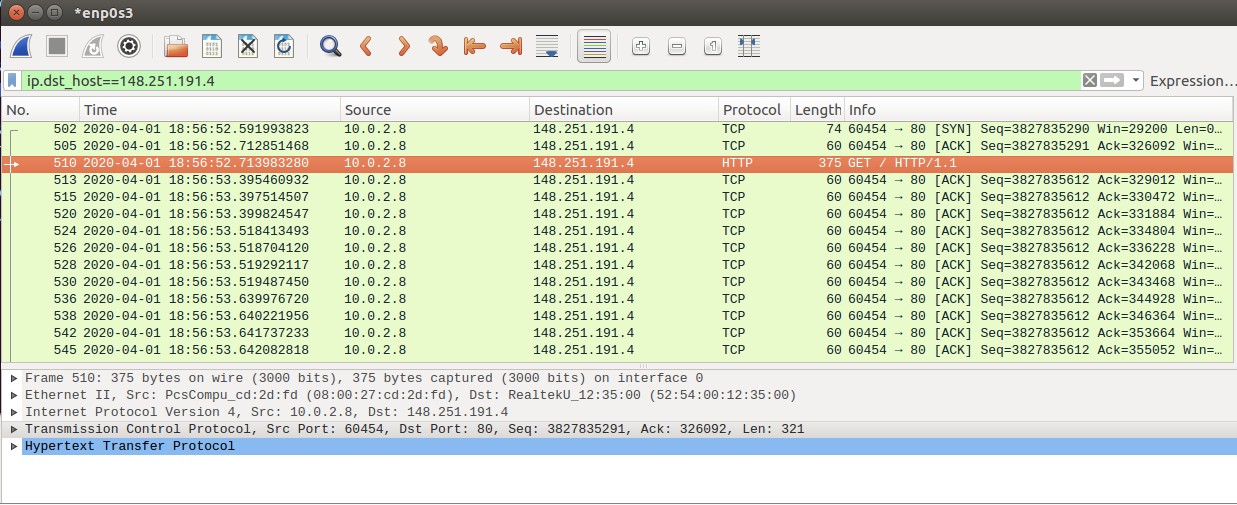


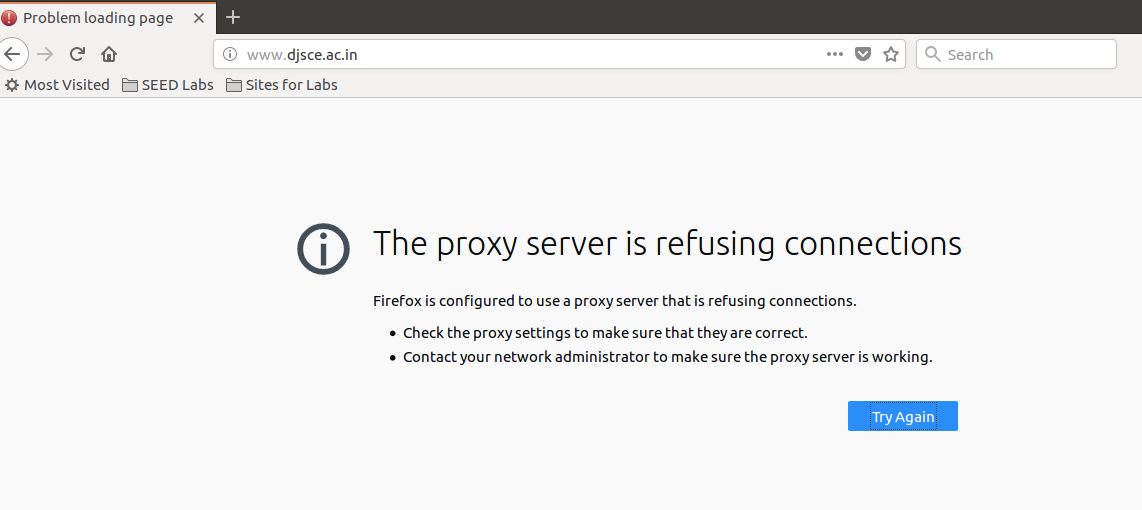
**Task#03.B: Connect to Facebook using SSH Tunnel**

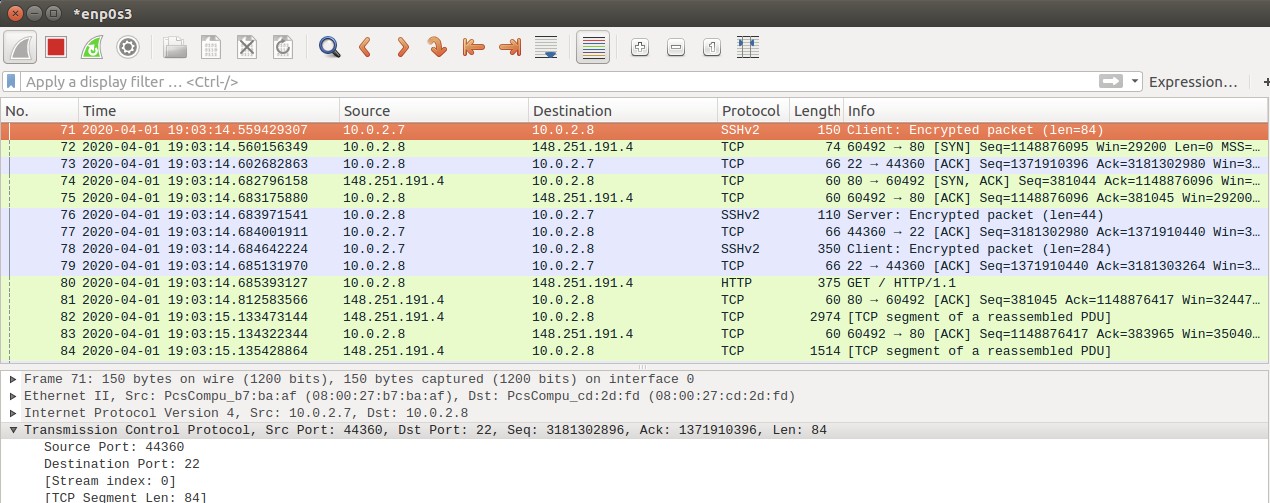
Utilizing Dynamic Port Forwarding as opposed to the Static Port Forwarding employed in the preceding task is another method for circumventing firewall rules. In this case, only the local port number and not the destination is specified. The packet is forwarded dynamically at the receiver's end based on the packet's destination information. This is accomplished through the use of a programme that creates an SSH tunnel between localhost port 9000 and Machine B port 22. Now, to bypass the firewall, we need the browser to connect to localhost:9000 whenever it wants to communicate with the web server. This ensures that traffic is routed through the SSH tunnel. To do this, we employ localhost: 9000 as Firefox's proxy and configure it as follows. We utilize the SOCKS proxy because, in order to dynamically forward traffic, we require the browser to inform the proxy about the destination, as this proxy is incapable of detecting the destination on its own. After this is configured, we visit the prohibited website and observe that it loads without issue. Upon examining the Wireshark traffic, we discover that the website traffic is routed through Machine B and not Machine A. This occurs because web traffic travels through the SSH tunnel between Machine A and Machine B, and then Machine B relays the traffic to its destination (web server). Since traffic now travels via SSH instead of HTTP, the firewall rule prohibiting access to the blacklisted domain is bypassed. Now, after breaking the SSH tunnel and restarting the page, we are unable to access the website. This error indicates that the proxy server is denying connections. Machines A and B are no longer connected through SSH, and this SSH tunnel served as the browser's proxy server. After re-establishing the SSH tunnel and reloading the web page, the page appears as before. Since the SSH tunnel operating as a proxy server has been built, the webpage has been reloaded.

In this task, the browser connects to the SSH proxy on the localhost at port 9000, and the SSH transfers the TCP data via the tunnel to Machine B, which then connects to the banned website (based on the destination). Since all traffic is SSH traffic and not web traffic, the firewall does not block anything. Wireshark reveals the same behavior, with traffic flowing from Machine A to Machine B over SSH and from Machine B to the web server via TCP/HTTP. This indicates that firewall restrictions can be circumvented by utilizing a different protocol, such as SSH, and performing the forbidden actions without the firewall's knowledge, unless deep packet inspection is employed. Using encryption-based protocols, such as SSH as opposed to telnet, it is possible to circumvent firewall regulations, even with deep packet inspection.

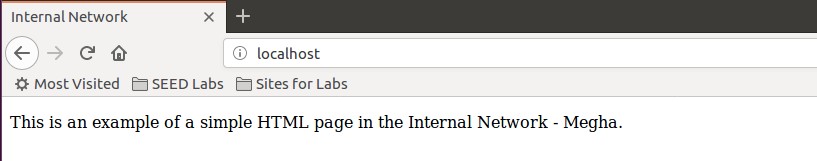








**Task#04: Evading the Ingress Filtering**

On Machine A, the web server is safeguarded by preventing HTTP/SSH traffic from the external network from entering (here just Machine B). Additionally, this machine can be set to restrict all SSH traffic from anywhere. The outcomes will be comparable. Below is a screenshot of the website and the firewall rules specified on Machine A. When attempting to visit this web server from the outside, it is discovered that http traffic has been banned. Additionally, we are unable to use the prior SSH tunnel approach for port forwarding, which would have granted us access to the web server. Since the firewall on Machine only restricts incoming SSH tunnels, we set up a reverse SSH tunnel on Machine A, which is not blocked. This SSH tunnel will allow access to the secured web server. After establishing the SSH tunnel described above, we can access the web server from the outside (Machine B - 10.0.2.8) by navigating to localhost: 8000 on the external Machine. This occurs because the established SSH tunnel passes the request to the SSH client on Machine A, which then forwards the request to port 80 on Machine A - 10.0.2.7, which is the web server. When the web server is successfully accessible from the outside, the results will be shown.

