MH4921
Supervised Independent Study II

Linux Firewall Lab

Brandon Goh Wen Heng

Academic Year 2018/19

Contents

1 Introduction **1**

2 Overview **1**

3 Exploration **1**
3.1 Virtual Machine (VM) Configuration 1
3.2 Using Firewalls 2
3.3 Optional: Implementing Packet Filtering Module 6
3.4 Evading Egress Filtering 8
3.4.1 `Telnet` Bypass 9
3.4.2 Facebook Bypass 10
3.5 Web Proxy – Application Firewall 13
3.5.1 `squid` Server Setup 13
3.5.2 Firewall Evasion 16
3.6 URL Rewriting/Redirection 17
3.6.1 `myprog.pl` 17
3.6.2 Replace Facebook Pages With Stop Sign 19
3.6.3 Replacing All Images In A Page 19

4 Appendix A **21**
4.1 Packet Filtering Module – pfm.c 21
4.2 Makefile 22

5 Appendix B **23**
5.1 `myprog.pl` 23
5.2 `myprog.pl` – Stop Sign 23
5.3 `myprog.pl` – Image Manipulation 24

1 Introduction

Firewalls are commonly used to block traffic, as evident in enterprises and academic institutions where applications and sites are blocked to prevent distractions or illegal traffic from occurring within the networks. There are multiple types of firewall but this lab will focus on two types, the *packet filter* and the application firewall.

Packet filters act by inspecting the packets and if it matches any of the firewall rules, the packets are forwarded or dropped. Packet filters are mostly *stateless*, in that the packets are filtered based on the information encoded in the individual packets and not based on the data stream. Packet filters may use a combination of the packet's source and destination address, protocol and port numbers among many fields.

Application firewalls such as web proxies work at the application layer where the data of the packet is analysed. This method is primarily used for egress filtering of web traffic.

2 Overview

This lab will aim to bring insights on how firewalls work by making use of a firewall software and a simplified packet filtering firewall. The first three sections will look at how the packet filtering firewall `ufw` works and the counter measures that could be taken to evade this type of filtering.

The last two sections of this lab will detail how access to websites can be controlled via a web proxy named `squid`, a type of application firewall. It will show how this type of firewall can be used to circumvent packet filtering firewalls. Also, `squid` can modify the contents of the displayed web page, using a method known as URL rewriting/redirection. The demonstration defaces the NTU home page on the local computer, elaborated in great detail at a later chapter of this report.

3 Exploration

3.1 Virtual Machine (VM) Configuration

The firewall that will be used later needs to have its configuration modified as its default firewall policy drops all incoming packets. To change it, the default policy file `/etc/default/ufw` is opened and the following line is modified.

`DEFAULT_INPUT_POLICY = 'DROP'` $\rightarrow$ `DEFAULT_INPUT_POLICY = 'ACCEPT'`

3.2 Using Firewalls

Linux distributions have a built in kernel firewall tool named `iptables`. The front-end alternative that is more user-friendly is `ufw`. By using `ufw`, it can be used to create a personal firewall for the system but alternative means are more efficient when configuring firewalls for larger networks. For this task, the following are attempted.

- Prevent Machine A from executing `Telnet` to Machine B
- Prevent Machine B from executing `Telnet` to Machine A
- Prevent Machine A from visiting an external website. (Note that some websites have multiple IP addresses)

The manipulation of the firewall is performed on machine A while the firewall on machine B is untouched and left at the default setting.

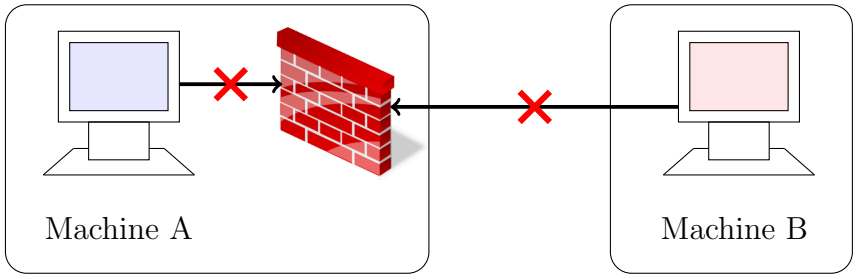


Figure 1: Firewall Setup

1. To prevent Machine A from executing `Telnet` to Machine B, it is known that Telnet uses (TCP) port 23. Furthermore from the "Remote DNS Attack Lab", source ports are randomised to prevent common ports from being hacked. Therefore, the firewall must block the destination ports instead.

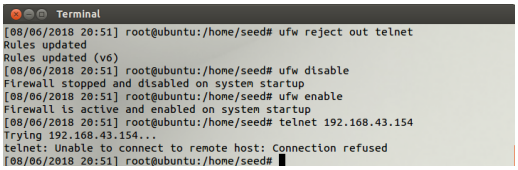
   Using `ufw`, there are multiple ways to block `Telnet` from being used from Machine A to Machine B. Below are two different methods that can be used to block outgoing `Telnet` packets.

   ```
   $ ufw deny out to any port 23
   $ ufw reject out telnet
   ```

   The commands `ufw disable` and `ufw enable` have to be used to force the firewall to enforce the new firewall rules. (*Note: because Telnet is a well known program that uses TCP port 23, the second line of code can be used. Otherwise the first line is more straightforward and direct.)

   Blocking `Telnet` on the firewall level will prevent the packets from reaching the intended destination and eventually force `Telnet` to timeout, as shown in the figure below.

(a) Before Firewall Rule



(b) After Firewall Rule

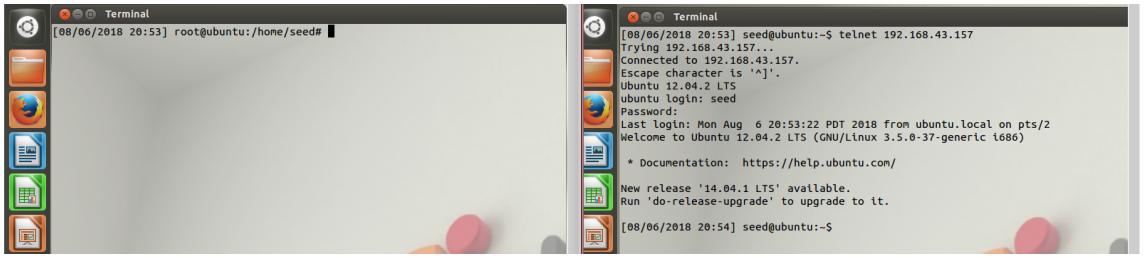
Figure 2: Prevent `Telnet` to External Host

2. To prevent Machine B from executing `Telnet` to Machine A, it is the same as the previous. However, the direction of the packet flow has been swapped. Due to this, the direction when writing the commands is of importance in determining whether the blocking is successful.

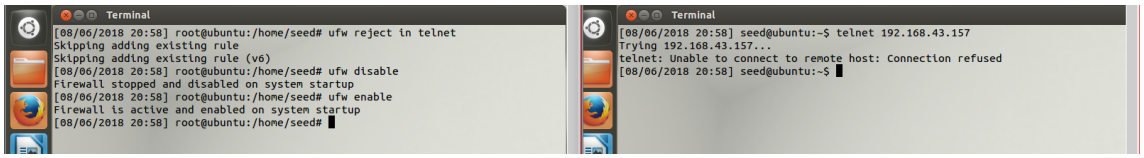
   Again, there are two ways of blocking which is provided below.

   ```
   $ sudo ufw deny in to any port 23
   $ sudo ufw reject in telnet
   ```

   Again, the commands `ufw disable` and `ufw enable` have to be used to force the firewall to enforce the new firewall rules. Executing `Telnet` from machine B will yield the same timeout message.



(a) Before Firewall Rule



(b) After Firewall Rule

Figure 3: Prevent `Telnet` to Current Host

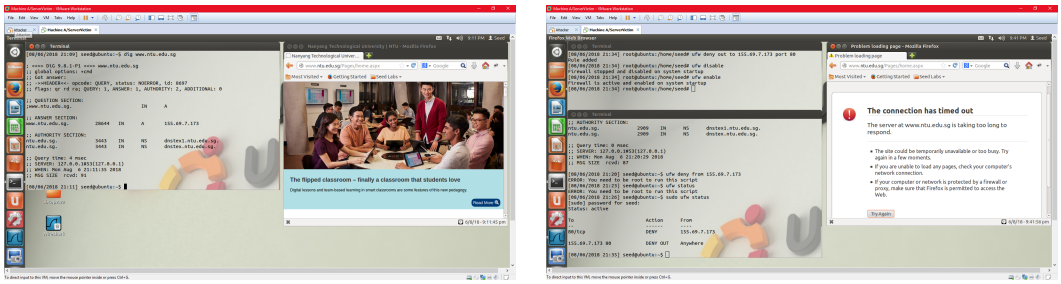
3. To prevent machine A from accessing a website, we can use `ufw` to perform the filtering. However, `ufw` is not able to block hostnames and as such the IP address must be known. In this task, the site `www.ntu.edu.sg` will be blocked.

To find out the IP address of the site, the command `dig` is sufficient to reveal the A records of the hostname.

To block (non-secured) websites, TCP port 80 must be blocked (443 for secured websites). Since the port numbers are directed to the server-side, outgoing packets are filtered. To do so, the following line can be used to block the site.

```
$ sudo ufw deny out to 155.69.7.173 port 80
```

The firewall is restarted to effect the changes and refreshing the site will now show that the connection to the website has timed out and cannot be restored.



(a) Before Firewall Rule (b) After Firewall Rule

Figure 4: Block Website Access

The `ufw` firewall uses packet filtering to inspect incoming and outgoing packets and to enforce the various policies configured by the administrator. As packet processing is performed in the kernel, as can be demonstrated in the "Packet Sniffing and Spoofing Lab", the filtering must also be performed within the kernel. However, using the *Loadable Kernel Module* (`LKM`) and `Netfilter`, the kernel need not be rebuilt to manipulate packets.

`LKM` extends the functionalities of the kernel by allowing administrators to add modules without rebuilding the kernel or rebooting the computer. To process the packets and to block it, the module must be integrated into the packet processing path, which was not easily implemented before `Netfilter`.

Netfilter allows the manipulation of packets by allowing administrators to implement hooks into the Linux kernel. These hooks exists in various places and programs (within LKM) and gets invoked when packets pass through.

Question 1:
What types of hooks does `Netfilter` support and what can be done using these hooks? Draw also a diagram to show how packets flow through these hooks.

Answer 1:
There are a total of `5` hooks that can be used with `Netfilter` and these hooks

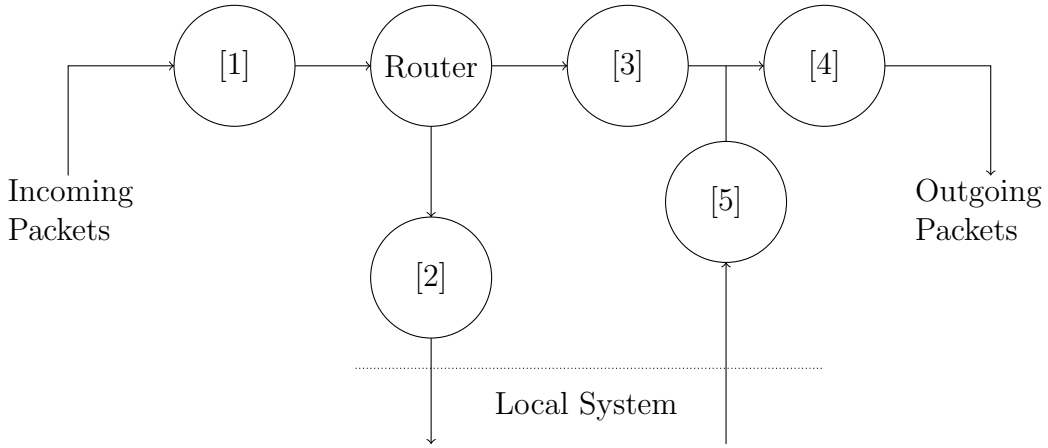
can be found in the header definition of `netfilter_ipv4.h` and `netfilter_ipv6.h`, both located in the folder `/usr/include/linux` (If `gcc` is installed). The various hooks, together with its description are listed in the table below.

IPv4 Hooks	IPv6 Hooks	Description
NF_IP_PRE_ROUTING	NF_IP6_PRE_ROUTING	This hook is triggered when packets enter into the network stack. After that, the packet is routed or dropped depending on the destination of the packet.
NF_IP_LOCAL_IN	NF_IP6_LOCAL_IN	This hook is triggered when the packet is designated for the local system.
NF_IP_FORWARD	NF_IP6_FORWARD	This hook is triggered when the packet is designed to be routed out to another system.
NF_IP_LOCAL_OUT	NF_IP6_LOCAL_OUT	This hook is triggered when packets are created locally and are outgoing to the network.
NF_IP_POST_ROUTING	NF_IP6_POST_ROUTING	This hook is triggered just before packets are to be sent out the physical wire.

Table 1: Table of `Netfilter` Hooks

*For newer kernel versions, the prefix `NF_IP` has been changed to `NF_INET`.

The figure below depicts the different locations where the programs can hook onto the networking stack to manipulate the respective packets.



[1]: NF_IP_PRE_ROUTING
[2]: NF_IP_LOCAL_IN
[3]: NF_IP_FORWARD
[4]: NF_IP_POST_ROUTING
[5]: NF_IP_LOCAL_OUT

Figure 5: `Netfilter` System

Question 2:
Where should a hook be placed for ingress and egress filtering?

Answer 2:
Ingress filtering is a technique to prevent suspicious traffic from entering the internal network. Ingress filtering checks for the validity of packets by analysing its source addresses to ensure that the IP address used does not originate within the network or any private addresses such as multicast. Therefore, an ingress filtering hook should be placed at NF_IP_PRE_ROUTING, before it proceeds to any part of the network.

For egress filtering, it is the opposite of ingress filtering. It checks for outgoing packets and prevents data from flowing out, such as malicious packets containing spyware. In this case, egress filtering must be hooked to NF_IP_POST_ROUTING.

Question 3:
Can packets be modified using `Netfilter`?

Answer 3:
While `Netfilter` is primarily used for analysing incoming, outgoing packets and to block where required based on the firewall policy, the user can write programs (within LKM) that hook onto `Netfilter` and use it to modify packets instead.

3.3 Optional: Implementing Packet Filtering Module

A firewall should support dynamic configurations and be able to dynamically change firewall policies. The configuration tool runs in the user space, but the data has to be sent to the kernel space where the packet filtering module (LKM) can obtain the data. These data must reside in the kernel memory instead of a file due to performance issues. This involves interactions between a user-level program and the kernel module.

Using sample code provided by Paul Kiddie[1], the code is extended to drop IPv4 ICMP (ping) packets instead of all packets. Before proceeding, there are multiple ways to process the packets using our module. These definitions are located inside `netfilter.h`.

1. NF_DROP: Drop packet.
2. NF_ACCEPT: Accept packet and process through network chain.
3. NF_STOLEN: Ownership of packet transferred to hook and `Netfilter` will not process the packet further.
4. NF_QUEUE: Get `Netfilter` to queue the packet for userspace.
5. NF_REPEAT: Call the hook again.
6. NF_STOP: Accept packet and stop further processing along network chain.

[1] `https://www.paulkiddie.com/2009/10/creating-a-simple-hello-world-netfilter-module/`

The kernel module needs to be compiled via a `Makefile`. Both the packet filtering module code and the `makefile` code have been attached to Appendix A. It also interesting to note that when compiling the file via `Makefile`, the module libraries are located in `/lib/modules/$(shell uname -r)/build/include`. This is in contrast to normal compilation using `gcc`, where the libraries are located in the folder `/usr/include`.

When compiling code, it is best to ensure that there are no warnings that appear as these errors may crash the system when the module is loaded into the kernel.

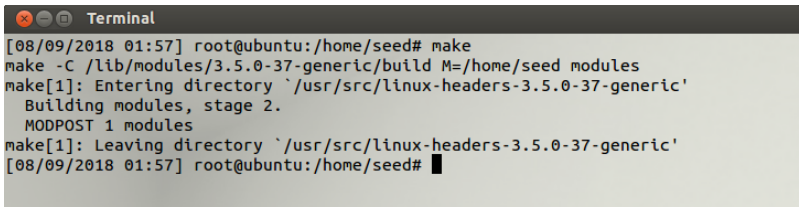


Figure 6: No errors during `make`

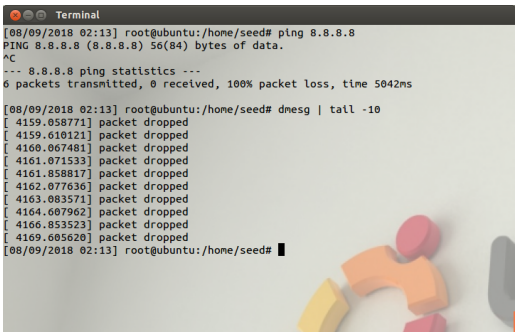
To add the compiled program into the kernel, the following command is used.

```
$ sudo insmod pfm.ko
```

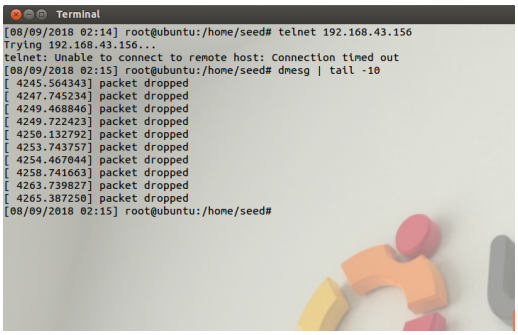
The code provided is compiled and further extended to only drop IPv4 ICMP packets. These programs are loaded into the kernel with names `pfm.ko` and `pfm2.ko` respectively. To remove the kernel modules and to display the kernel log, the following two lines can be used.

```
$ sudo rmmod pfm.ko
$ dmesg | tail -10 //Where 10 stands for the last x lines displayed
```

Looking at the figures below, both `ping` and `Telnet` are used to show the different types of packets that are accepted or dropped.

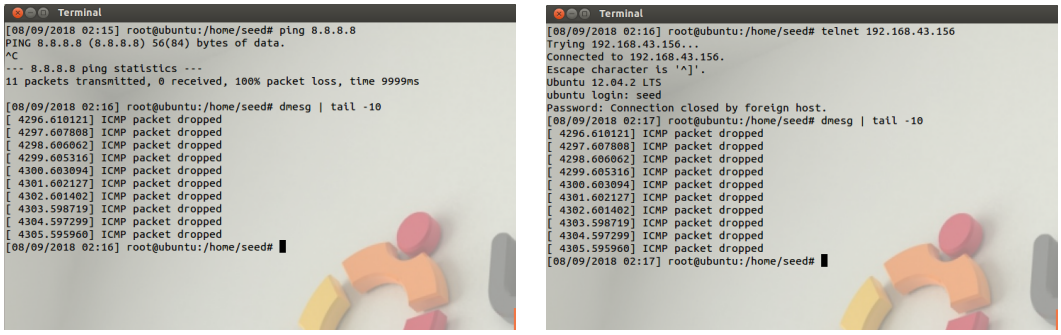


(a) Dropped `ping` Packets



(b) Dropped `Telnet` Packets

Figure 7: All Packets Dropped



(a) Dropped `ping` Packets (b) `Telnet` Packets Allowed

Figure 8: Only ICMP Packets Dropped

From the two figures above, all packets were dropped when the module was attached to the kernel, except for figure 8b. In can be seen that `Telnet` can be executed and the timestamp for the logs are for the dropped `ping` packets in figure 8a.

3.4 Evading Egress Filtering

Egress filtering is commonly used in companies and schools to restrict the use of certain services and applications. These firewalls inspect the destination IP addresses and port numbers of outgoing packets and dropped if any matches the firewall rules. Deep packet inspections are not performed on the packets due to performance reasons and therefore using the tunnelling mechanism, egress filtering can be bypassed.

Three VMs are setup to demonstrate this purpose. Machine A is behind a firewall while Machines B and C are outside of the firewall. The following firewall rules are also setup using `ufw` on Machine A as it is more convenient than re-writing the hook.

1. All outgoing traffic to external telnet servers are blocked. The `Telnet` server is run on Machine C and can be started using the command `sudo service openbsd-inetd start`.
2. All outgoing traffic to `www.facebook.com` are blocked to emulate the restriction in companies/schools to prevent them from being distracted. As Facebook runs on 2 IP addresses, both are blocked (including its https variants). The following lines of code are run to ensure all IP addresses of Facebook are blocked.

   ```
   $ sudo ufw deny out to 157.240.7.38 port 80
   $ sudo ufw deny out to 157.240.7.38 port 443
   $ sudo ufw deny out to 157.240.7.35 port 80
   $ sudo ufw deny out to 157.240.7.35 port 443
   $ sudo ufw disable
   $ sudo ufw enable
   ```

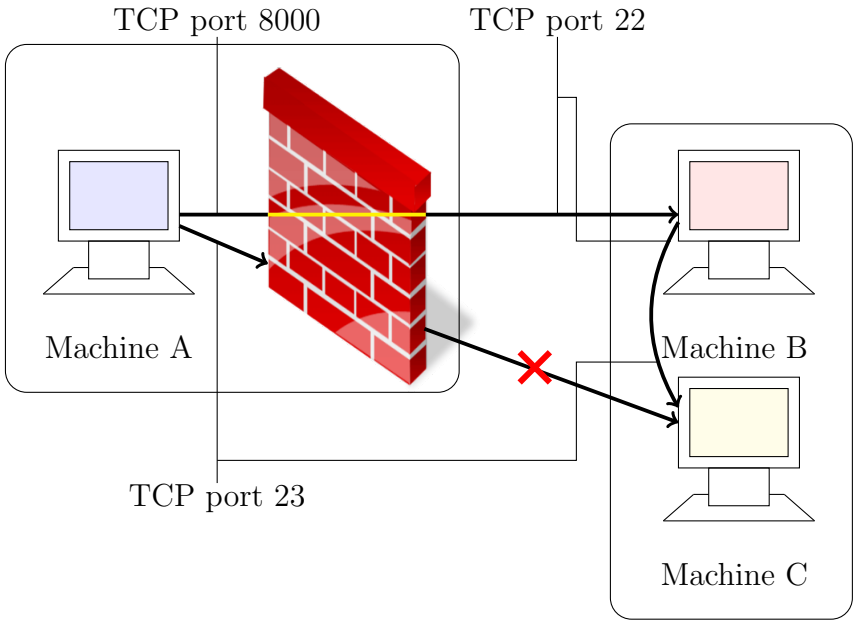
   To check whether the implementation is successful, the browser can either have its cache cleared or perform a force refresh (`Ctrl+R`).

3.4.1 `Telnet` Bypass

To bypass the firewall, a SSH tunnel between Machine A and B can be established. This will allow all telnet traffic to go through encrypted, evading any inspection. The following line establishes a SSH tunnel from the localhost's port 8000 Machine B and packets that exit Machine B's SSH port will be forwarded to Machine C.

```
$ ssh -L 8000:<Machine C IP>:23 seed@<Machine B IP>
```

More details on the local and remote forwarding switches can be found on the Unix Stack Exchange.



Machine A has IP address 192.168.43.157
Machine B has IP address 192.168.43.154
Machine C has IP address 192.168.43.156

Figure 9: SSH Tunneling

Once the SSH tunnel has been established, another Terminal window is opened. To connect to Machine C, we will force `Telnet` to use port 8000 since the packet will be forwarded to Machine C via the SSH tunnel.

```
$ telnet localhost 8000
```

Using this method, Machine A can successfully be connected to Machine C via `Telnet`. The `ifconfig` is run to check whether the IP address of the connected system is correct and this is shown in the figure below.

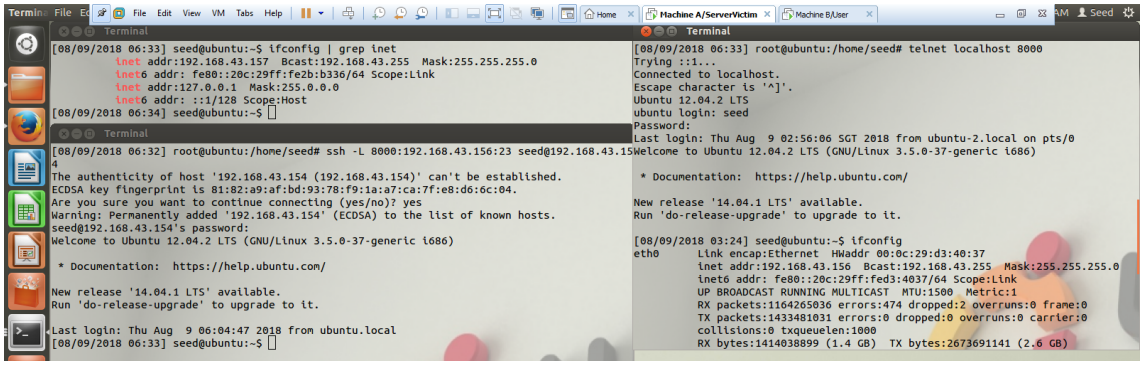


Figure 10: Successfully Connected to Machine C

Wireshark is looked at and it can be seen that one cycle out from and back to Machine A includes

1. Sending an encrypted packet from Machine A to Machine B via SSH
2. Machine B sending `Telnet` data to Machine C
3. Machine C returns back `Telnet` acknowledgement data and packet
4. Machine B sends the data back to Machine A in an encrypted response packet via SSH

Packet numbers 18 – 22 in the Wireshark screen capture below show the corresponding interactions between all three machines to transfer data.

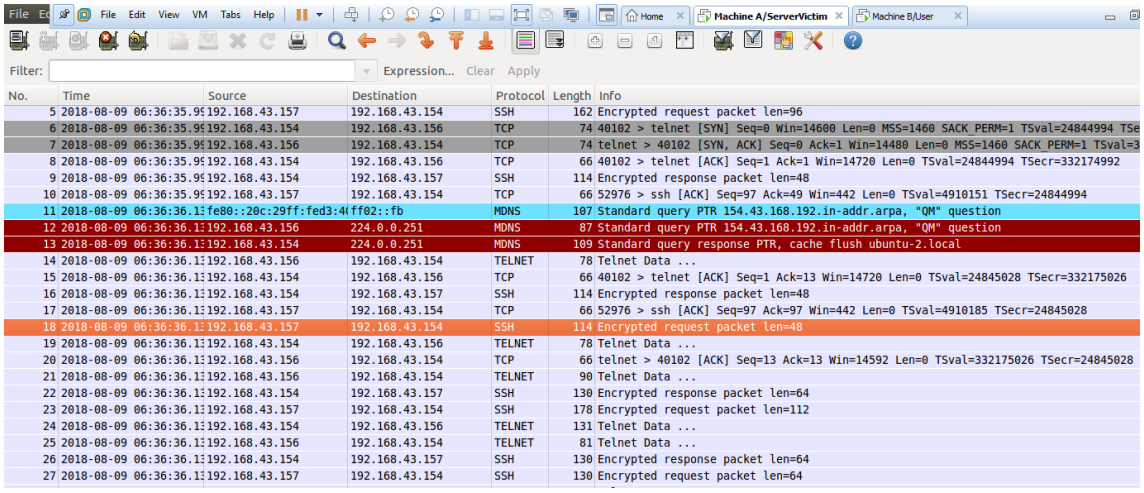


Figure 11: Wireshark Packet Flow

3.4.2 Facebook Bypass

In this task, the same approach using static port forwarding can be used to bypass the firewall. However, dynamic port forwarding can also be used. Instead of requiring that the destination IP address and port number be specified, these can now be omitted. Using dynamic port forwarding only requires the source port and the intermediary IP address be specified, as the intermediary system will determine

where it will forward based on the destination IP address encoded in the packet.

The following command allows us to perform dynamic port forwarding using SSH.

```
$ ssh -D 9000 -C seed@<Machine B IP>
```

The `-D` flag is used to specify that it is a dynamic port forwarding tunnel and the `-C` flag is used to force data compression.

To make use of port 9000 SSH tunnel, the browser must also be configured to make use of it. To configure Firefox, navigate to `Edit > Preferences > Advanced > Network > Settings`. Under "Manual proxy configuration", SOCKS Host should be set to either `localhost` or 127.0.0.1, port set to 9000 and "SOCKS v5" selected. Under "No Proxy for:", the entries should be `localhost, 127.0.0.1`.

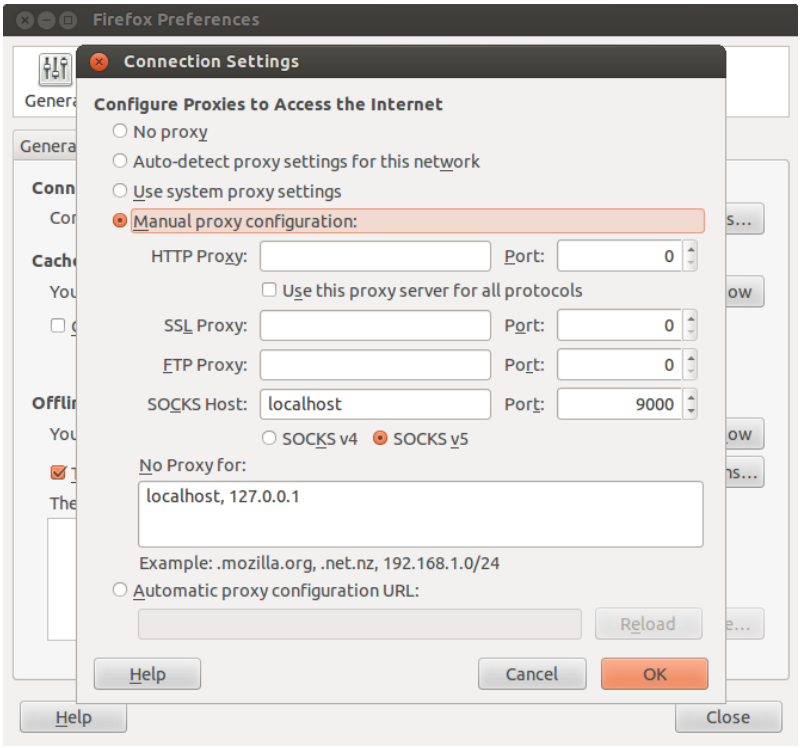
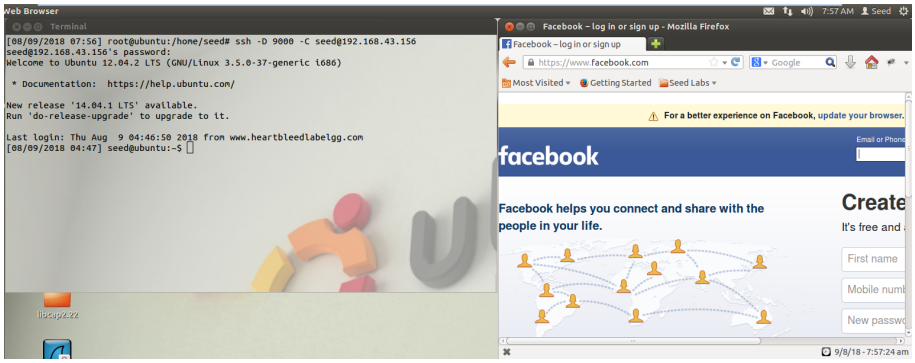
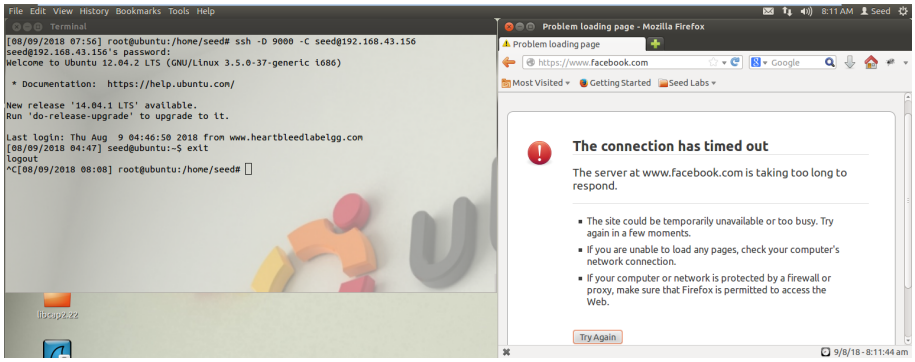


Figure 12: Firefox Configuration

When the SSH tunnel is running in the background on a `Terminal` window, Facebook can be accessed normally via the browser. If the SSH tunnel is broken, Facebook will again be blocked due to the firewall (after removing the proxy settings).



(a) Accessing Facebook via SSH



(b) Blocked Without SSH

Figure 13: Differences with SSH Tunnel

Using Wireshark to analyse the transmitted packets, the same pattern is reflected as the previous task on `Telnet`. The data is transmitted to Machine C via SSH and it is then forwarded over to Facebook's servers (Client Hello). The server returns back the response to Machine C and it is forwarded back to Machine A via SSH.

No.	Time	Source	Destination	Protocol	Length	Info
15	2018-08-09 08:20:56.94	192.168.43.156	192.168.43.157	SSHv2	378	Server: New Keys
16	2018-08-09 08:20:56.95	192.168.43.157	192.168.43.156	SSHv2	82	Client: New Keys
17	2018-08-09 08:20:56.98	192.168.43.156	192.168.43.157	TCP	66	ssh > 52601 [ACK] Seq=1338 Ack=1410 Win=17408 Len=0 TSval=333740166 TSecr=6475389
18	2018-08-09 08:20:56.98	192.168.43.157	192.168.43.156	TCP	114	[TCP segment of a reassembled PDU]
19	2018-08-09 08:20:56.98	192.168.43.156	192.168.43.157	TCP	66	ssh > 52601 [ACK] Seq=1338 Ack=1458 Win=17408 Len=0 TSval=333740166 TSecr=6475398
20	2018-08-09 08:20:56.95	192.168.43.156	192.168.43.157	TCP	114	[TCP segment of a reassembled PDU]
21	2018-08-09 08:20:57.01	192.168.43.157	192.168.43.156	TCP	66	52601 > ssh [ACK] Seq=1458 Ack=1386 Win=18560 Len=0 TSval=6475408 TSecr=333740166
22	2018-08-09 08:20:57.11	192.168.43.157	192.168.43.156	TCP	130	[TCP segment of a reassembled PDU]
23	2018-08-09 08:20:57.12	192.168.43.156	192.168.43.157	TCP	130	[TCP segment of a reassembled PDU]
24	2018-08-09 08:20:57.12	192.168.43.157	192.168.43.156	TCP	66	52601 > ssh [ACK] Seq=1522 Ack=1450 Win=18560 Len=0 TSval=6475431 TSecr=333740199
25	2018-08-09 08:21:02.92	192.168.43.157	192.168.43.156	TCP	210	[TCP segment of a reassembled PDU]
26	2018-08-09 08:21:02.94	192.168.43.156	192.168.43.157	TCP	98	[TCP segment of a reassembled PDU]
27	2018-08-09 08:21:02.94	192.168.43.157	192.168.43.156	TCP	66	52601 > ssh [ACK] Seq=1666 Ack=1482 Win=18560 Len=0 TSval=6476887 TSecr=333741654
28	2018-08-09 08:21:02.94	192.168.43.157	192.168.43.156	TCP	194	[TCP segment of a reassembled PDU]
29	2018-08-09 08:21:02.96	192.168.43.156	192.168.43.157	TCP	66	ssh > 52601 [ACK] Seq=1482 Ack=1794 Win=22528 Len=0 TSval=333741664 TSecr=6476887
30	2018-08-09 08:21:03.01	192.168.43.156	192.168.43.157	TCP	114	[TCP segment of a reassembled PDU]
31	2018-08-09 08:21:03.01	192.168.43.157	192.168.43.156	TCP	386	[TCP segment of a reassembled PDU]
32	2018-08-09 08:21:03.01	192.168.43.156	192.168.43.157	TCP	66	ssh > 52601 [ACK] Seq=1530 Ack=2114 Win=25088 Len=0 TSval=333741678 TSecr=6476911
33	2018-08-09 08:21:03.03	192.168.43.156	192.168.43.157	TCP	178	[TCP segment of a reassembled PDU]
34	2018-08-09 08:21:03.04	192.168.43.156	192.168.43.157	TCP	322	[TCP segment of a reassembled PDU]
35	2018-08-09 08:21:03.04	192.168.43.157	192.168.43.156	TCP	66	52601 > ssh [ACK] Seq=2114 Ack=1898 Win=20544 Len=0 TSval=6476912 TSecr=333741678
36	2018-08-09 08:21:03.31	192.168.43.156	192.168.43.157	TCP	130	[TCP segment of a reassembled PDU]
37	2018-08-09 08:21:03.35	192.168.43.157	192.168.43.156	TCP	66	52601 > ssh [ACK] Seq=2114 Ack=1962 Win=20544 Len=0 TSval=6476990 TSecr=333741747
44	2018-08-09 08:21:17.85	192.168.43.157	192.168.43.156	TCP	146	[TCP segment of a reassembled PDU]
47	2018-08-09 08:21:17.85	192.168.43.156	157.240.7.38	TCP	74	35114 > https [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=333745392 TSecr=0 WS=128
48	2018-08-09 08:21:17.96	157.240.7.38	192.168.43.156	TCP	60	https > 35114 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
49	2018-08-09 08:21:17.96	192.168.43.156	157.240.7.38	TCP	60	35114 > https [ACK] Seq=1 Ack=1 Win=14600 Len=0
50	2018-08-09 08:21:17.96	192.168.43.156	192.168.43.157	TCP	114	[TCP segment of a reassembled PDU]
51	2018-08-09 08:21:17.96	192.168.43.157	192.168.43.156	TCP	66	52601 > ssh [ACK] Seq=2194 Ack=2010 Win=20544 Len=0 TSval=6480627 TSecr=333745394
52	2018-08-09 08:21:17.96	192.168.43.157	192.168.43.156	TCP	514	[TCP segment of a reassembled PDU]
53	2018-08-09 08:21:17.96	192.168.43.156	157.240.7.38	TLSv1	451	Client Hello
54	2018-08-09 08:21:17.96	157.240.7.38	192.168.43.156	TCP	60	https > 35114 [ACK] Seq=1 Ack=398 Win=64240 Len=0
55	2018-08-09 08:21:17.91	157.240.7.38	192.168.43.156	TLSv1	199	Server Hello, Change Cipher Spec, Encrypted Handshake Message
56	2018-08-09 08:21:17.91	192.168.43.156	157.240.7.38	TCP	60	35114 > https [ACK] Seq=398 Ack=146 Win=15544 Len=0
57	2018-08-09 08:21:17.91	192.168.43.156	192.168.43.157	TCP	258	[TCP segment of a reassembled PDU]
58	2018-08-09 08:21:17.91	192.168.43.157	192.168.43.156	TCP	162	[TCP segment of a reassembled PDU]

Figure 14: Proxy Packet Transfer

Question 4:

If `ufw` blocks SSH TCP port 22, can a SSH tunnel still be set up to evade egress filtering?

Answer 4:
Yes. Both on the server and client side, the port number for SSH can be changed by modifying the file `/etc/ssh/sshd_config`. For this example, the port number is changed from 22 to 26. The SSH service is then restarted and tested. For convenience, the `-p` switch is sufficient to show that a SSH tunnel can be established.

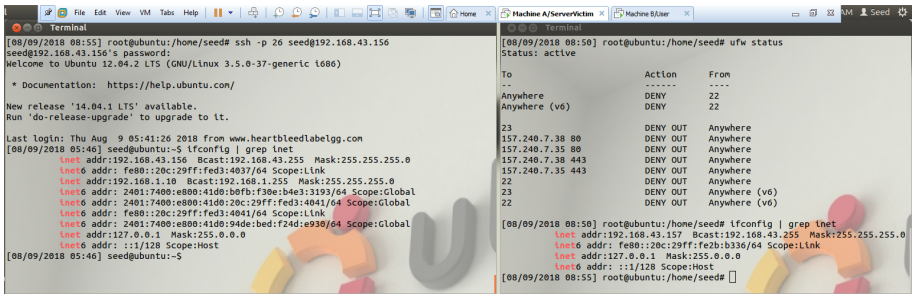


Figure 15: SSH Egress Filtering Bypassed

Again, we see from the figure above that we can obtain a connection to a remote server and obtain the server's IP address, bypassing the `ufw` firewall.

3.5 Web Proxy – Application Firewall

Application firewalls are another type of firewall that can be used to filter packets. Instead of inspecting data at the layer data (looking at the header fields of the packets), these firewalls look at application-layer data. These firewalls control access from/to applications or services.

3.5.1 `squid` Server Setup

A common implementation of this firewall are web proxies. For this lab, the web proxy that will be used is `squid` and can be installed by executing `sudo apt-get squid`.

After installation, the firewall policies can be set-up by modifying the file `squid.conf` in the folder `/etc/squid3`. The `squid` server needs to be restarted every time a modification has been done to the file, otherwise the new rules will not effect.

Two VMs are set-up for this task, Machine A and B. Machine A will emulate the system that requires its browsing restricted and Machine B will run the web proxy. To configure the browser for Machine A, the "HTTP Proxy" option is set to Machine B's IP address and port 3128 (default port for `squid` service).

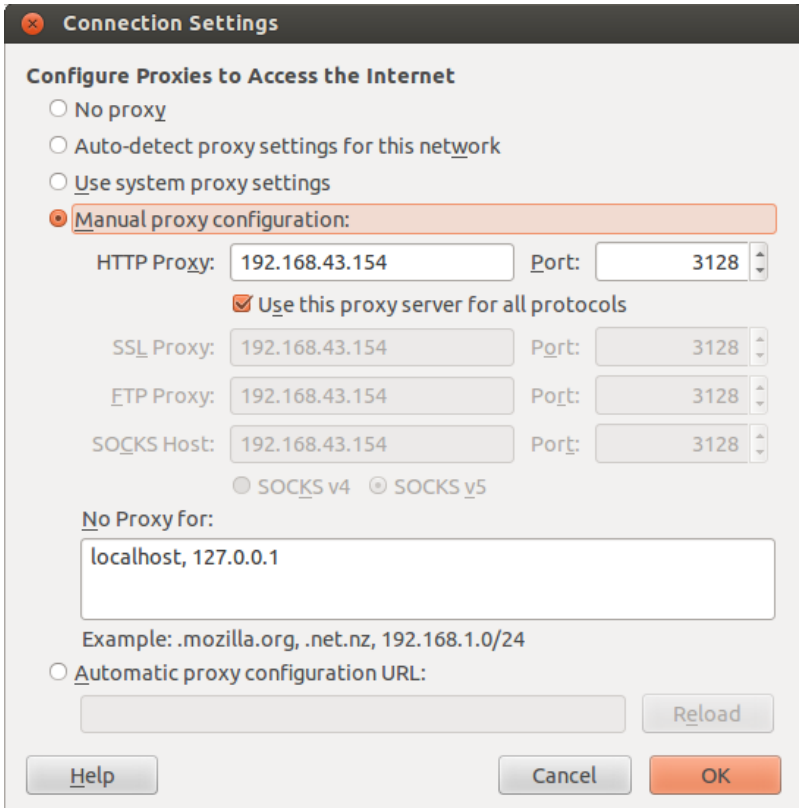


Figure 16: Proxy Configuration Settings

After configuration, any website that is visited will show an access denied error. Also, we notice that at the footer that `squid` and the version of the software is mentioned when the site is blocked. This shows that the configuration of the software is correct and that all sites are blocked by default after installing `squid`.

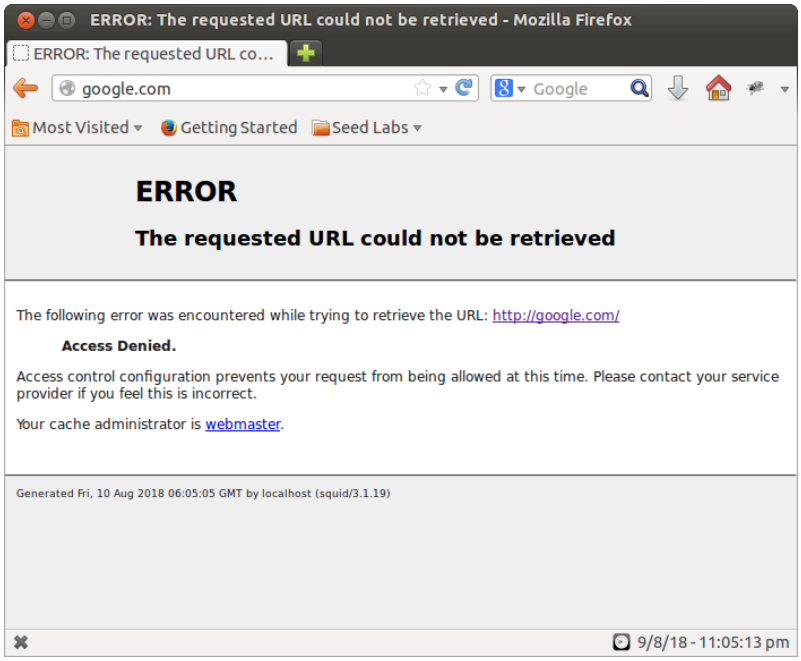


Figure 17: Blocked By Proxy

To check why all sites are blocked, the configuration file needs to be accessed. To find the line causing this quickly, this line is used.

```
$ cat /etc/squid3/squid.conf | grep deny
```

The lines that stands out from the printed result are in the firewall access block of code. In this block, every site is denied with the exception of localhost sites, which are not affected.

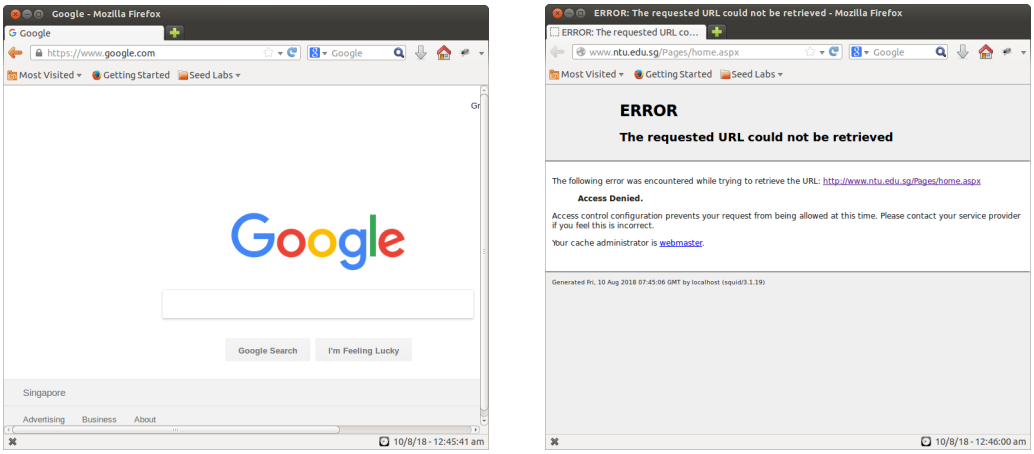
```
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
#http_access deny to_localhost
# And finally deny all other access to this proxy
http_access deny all
```

To allow all websites through the web proxy, an Access Control List (ACL) needs to be created. By doing so, we can allow this ACL to access any website. The following lines must be added **before** `http_access deny all`.

```
acl all src 0.0.0.0/0   #acl <acl name> <type> <data>
http_access allow all   #http_access <allow/deny> <acl name>
```

To allow specific sites like `google.com` to be accessed and other sites blocked, the ACL needs to be used. The previous entries need to be commented out or it will conflict with the current ACL.

```
#acl all src 0.0.0.0/0
#http_access allow all
acl google dstdomain .google.com.
http_access allow google
```



(a) Access to Google (b) Blocked to Anywhere Else

Figure 18: Selective Access via ACL

Analysing the captured packets via Wireshark, we noticed that for Google the packet flow was the same as when SSH was used. The data is passed from Machine A to the web proxy and that data was forwarded to the respective destination IP address and port. Similarly, return of data from the destination goes through the web proxy and back to Machine A.

However, for the site that was denied, the data goes through from Machine A to the web proxy but is now returned an access denied page. This is in contrast to the previous firewall where the packets are dropped.

3.5.2 Firewall Evasion

Question 5:
If `ufw` blocks TCP port 3128, can the web proxy still be used to evade the firewall?

Answer 5:
Yes. Similar to the SSH tunnel, the listening port for the `squid` server can still be changed to some arbitrary port number. To show that this is possible, an entry in the `ufw` firewall is added to block any packets to destination port 3128.

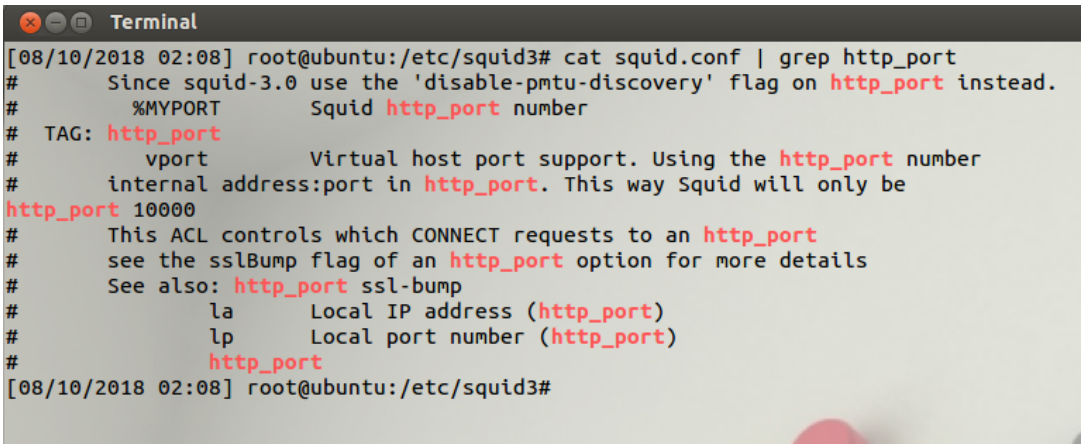
```
$ sudo ufw deny out 3128/tcp
$ sudo ufw disable
$ sudo ufw enable
```

A random port number is selected, for this scenario the port number is amended to 10000. Port numbers below 1024 are usually reserved for specific applications and it is highly advisable to not use that range.

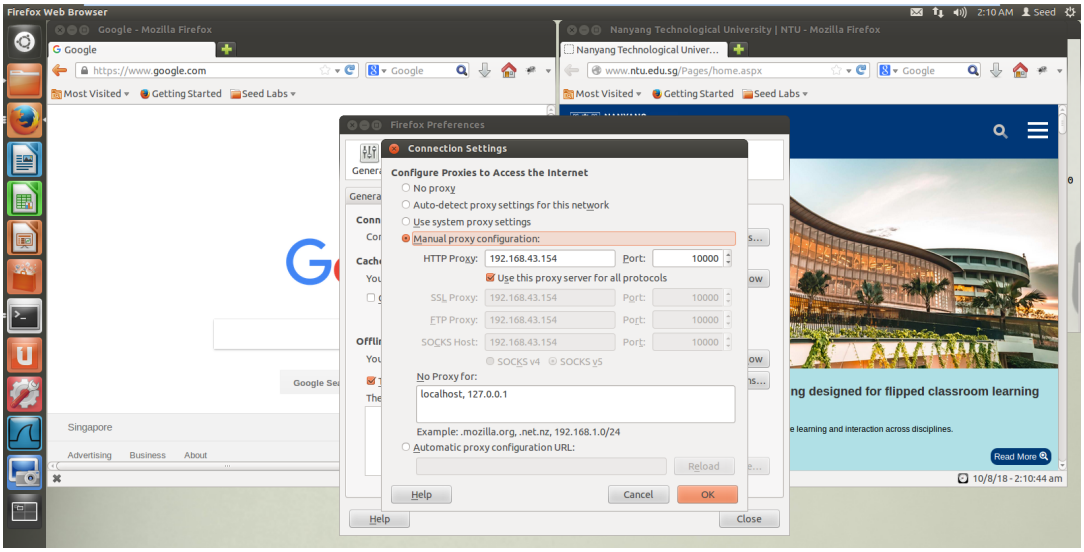
In the `squid.conf` file, the line `http_port` is searched for (`Ctrl + W` in nano) and modified.

`http_port 3128` → `http_port 10000`

The `squid` service is restarted and the browser is configured to use port 10000 also.



(a) `squid` Port Configuration



(b) Access Re-established

Figure 19: Firewall Bypassed

3.6 URL Rewriting/Redirection

3.6.1 `myprog.pl`

Apart from acting as a firewall, `squid` can also be used to rewrite URLs to redirect users to another web site. This can be done using any type of programming language. A perl program `myprog.pl` has been provided to demonstrate URL rewriting capabilities. It has been attached to Appendix B.

In short, the code checks whether the URL entered is identical to `www.ntu.edu.sg`. If it is, then the user is redirected to `www.google.com.sg`. The operator `=~` is used to check if the string is identical and not exact/equal (`==`).

To attach the the program to `squid`, the following lines are added to the `squid.conf` file.

```
url_rewrite_program /home/seed/myprog.pl #Location of file
url_rewrite_children 5
```

The perl file must be marked as executable (using `chmod +x`) or the firewall will not work. In the event the file cannot be found by `squid`, the URL rewrite program can be placed in `/etc/squid3` instead.

As previously mentioned, any attempt to view the site `www.ntu.edu.sg` will redirect the user to `www.google.com.sg`. The packets from this action are captured in Wireshark and analysed in detail. In particular, we look into the initial packet that was sent over the wire to the server and follow the packet stream. This action can be done by right-clicking the corresponding packet and selecting the option "Follow TCP Stream". The data printed shows that the web proxy replies with a "HTTP 302 Moved Temporarily" and the rewritten URL response instead of forwarding the response to our intended destination.

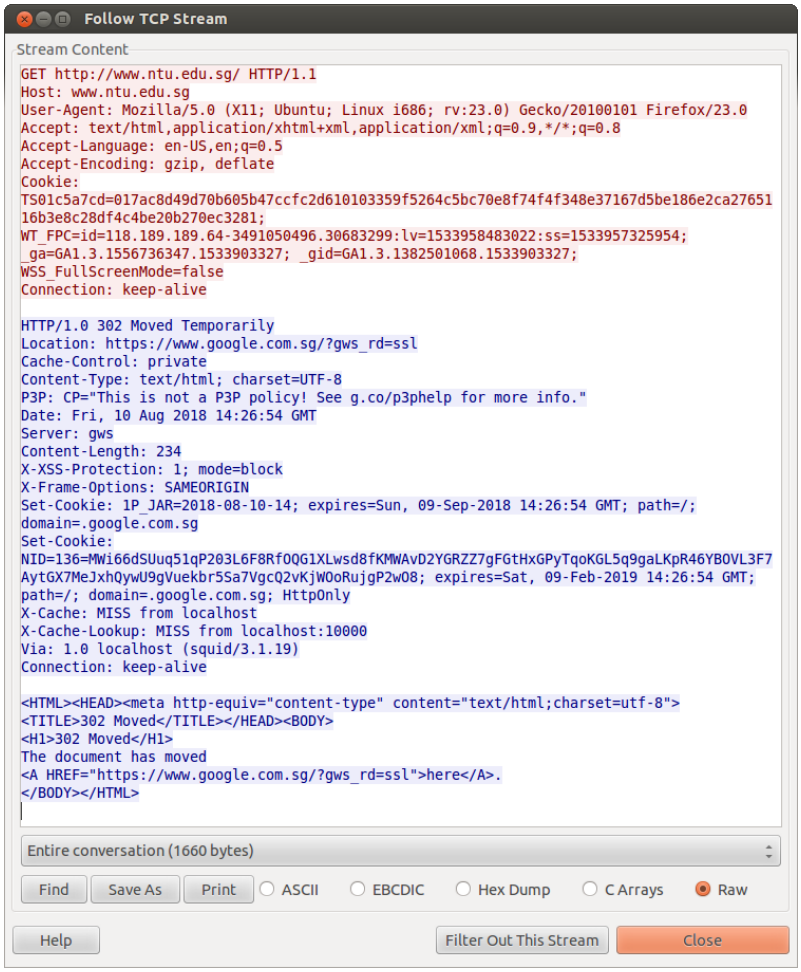


Figure 20: TCP Stream Data

3.6.2 Replace Facebook Pages With Stop Sign

In this lab instead of redirecting users from a legitimate site to another, we redirect users from Facebook to a big red stop sign to act as a deterrent to prevent users from being distracted. The same program is used with some modifications and it has been reflected in Appendix B.

Again, typing `facebook.com` into the browser will force a redirection to a stop sign. This is evident if the TCP stream was followed in Wireshark.

(a) Facebook Redirect (b) Stream Capture

Figure 21: Image Redirect

3.6.3 Replacing All Images In A Page

This part will look at replacing all images on a web page with an image of our choice. When an image is loaded on the page, it will immediately send a URL request for each image. These URLs will be identified and replaced accordingly by the web proxy. The provided code again has been attached to Appendix B.

A site that has multiple images is accessed, such as the NTU homepage. All images that end with any mentioned image extension (in the code) will have it replaced by a stop sign. This is evident as all the images have been replaced, such as shown in the figure below.

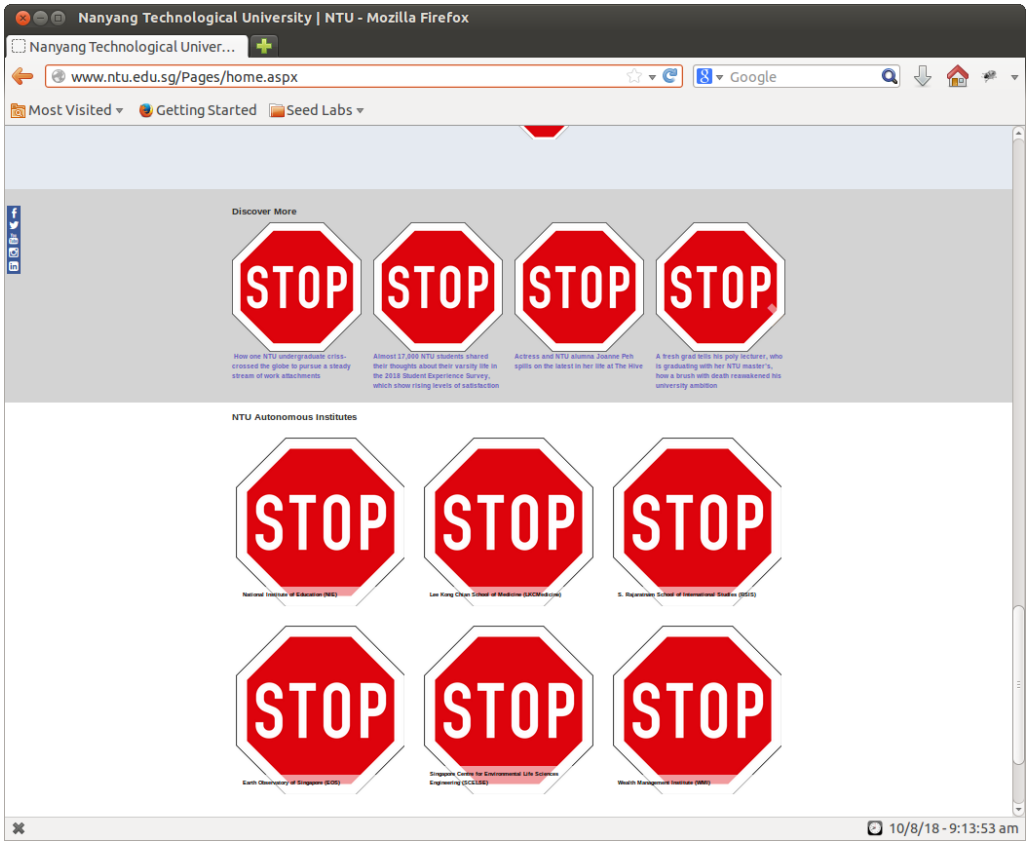


Figure 22: NTU Defaced via Web Proxy URL Rewriting (Zoomed out to 30%)

The figure shown above shows how powerful web proxies can be used to deface websites and redirect as well as to control access to websites, shown in the previous sections. Programs and proxies can also be adapted to evade filtering on any level or to inject malicious code into websites where the user is not aware of.

4 Appendix A

4.1 Packet Filtering Module – pfm.c

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

static struct nf_hook_ops nfho;          //struct holding set of hook function options
struct iphdr *iphd;

//function to be called by hook
unsigned int hook_func(unsigned int hooknum, struct sk_buff *skb, const struct net_device *in, const struct net_device *out, int (*okfn)(struct sk_buff *))
{
  iphd = ip_hdr(skb);
  if(iphd->protocol ==IPPROTO_ICMP){                //Checks for ICMP packet (Added from original)
          printk(KERN_INFO "packet dropped\n");    //log to var/log/messages
          return NF_DROP;                          //drops the packet
        }
  else        //Accept if not ICMP (Added from original)
        return NF_ACCEPT;
}

//Called when module loaded using 'insmod'
int init_module()
{
  nfho.hook = &hook_func;                //function to call when conditions below met
  nfho.hooknum = NF_INET_PRE_ROUTING;  //called right after packet recieved, first hook in Netfilter
  nfho.pf = PF_INET;                     //IPV4 packets
  nfho.priority = NF_IP_PRI_FIRST;      //set to highest priority over all other hook functions
  nf_register_hook(&nfho);               //register hook

  return 0;                              //return 0 for success
}

```

```
35 //Called when module unloaded using 'rmmod'
36 void cleanup_module()
37 {
38   nf_unregister_hook(&nfho);                     //cleanup {
     ↪  unregister hook
39 }
```

*Notes for the code above:

1. The `&` symbol is missing from the original documentation and will cause the system to crash when the module is added to the kernel. The function definition found in `netfilter.h` explictly provides the structure for `nf_hook_ops` has been provided below.

```
struct nf_hook_ops {
        struct list_head list;

        /* User fills in from here down. */
        nf_hookfn *hook;
        struct module *owner;
        u_int8_t pf;
        unsigned int hooknum;
        /* Hooks are ordered in ascending priority. */
        int priority;
};
```

2. The function definition for the hook function has an added `*` for `*skb` when compared to the `netfilter.h` file and has been removed (although it can be simply corrected by using `ip_hdr(*skb)` on line 14).

```
typedef unsigned int nf_hookfn(unsigned int hooknum,
                               struct sk_buff *skb,
                               const struct net_device *in,
                               const struct net_device *out,
                               int (*okfn)(struct sk_buff *));
```

4.2 Makefile

```
1 obj-m += pfm2.o
2
3 all:
4         make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6 clean:
7         make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

5 Appendix B

5.1 myprog.pl

```
#!/usr/bin/perl -w
use strict;
use warnings;

# Forces a flush after every write or print on the STDOUT
select STDOUT; $| =1;

# Get the input line by line from the standard input.
# Each line contains a URL and some other information.
while (<>)
{
        my @parts = split;
        my $url = $parts[0];
        # If you copy and paste this code from this PDF file,
        # the ~ (tilde) character may not be copied correctly.
        # Remove it, and then type the character manually.
        if ($url =~ /www\.ntu\.edu\.sg/) {
                #URL Rewriting
                print "http://www.google.com.sg\n";
        }
        else {
                # No Rewriting
                print "\n";
        }
}
```

5.2 myprog.pl – Stop Sign

```
#!/usr/bin/perl -w
use strict;
use warnings;

# Forces a flush after every write or print on the STDOUT
select STDOUT; $| = 1;

# Get the input line by line from the standard input.
# Each line contains a URL and some other information.
while (<>)
{
        my @parts = split;
        my $url = $parts[0];
        #If you copy and paste this code from this PDF file,
```

```
        # the ~ (tilde) character may not be copied correctly.
        # Remove it, and then type the character manually.
        if ($url =~ /facebook\.com/){
                #URL Rewriting
                print "http://upload.wikimedia.org/wikipedia/commons/
                 ↪  thumb/1/1e/Vienna_Convention_road_sign_B2a.svg/
                 ↪  500px-Vienna_Convention_road_sign_B2a.svg.png\n";
        }
        else {
                #No Rewriting
                print "\n";
        }
}
```

*Note: Using `https` for the URL redirect will cause `squid` to identify it as an invalid protocol and will stop the redirection with an error message.

5.3 `myprog.pl` – Image Manipulation

```
#!/usr/bin/perl -w
use strict;
use warnings;

# Forces a flush after every write or print on the STDOUT
select STDOUT; $| = 1;

# Get the input line by line from the standard input.
# Each line contains a URL and some other information.
while (<>)
{
    my @parts = split;
    my $url = $parts[0];
    #If you copy and paste this code from this PDF file,
    # the ~ (tilde) character may not be copied correctly.
    # Remove it, and then type the character manually.
    if ($url =~ /\.(jpg|png|svg|PNG|JPG|JPEG|BMP|bmp|jpeg|SVG)/){
            #URL Rewriting
            print "http://upload.wikimedia.org/wikipedia/commons/
             ↪  thumb/1/1e/Vienna_Convention_road_sign_B2a.svg/
             ↪  500px-Vienna_Convention_road_sign_B2a.svg.png\n";
    }
    else {
            #No Rewriting
            print "\n";
    }
}
```