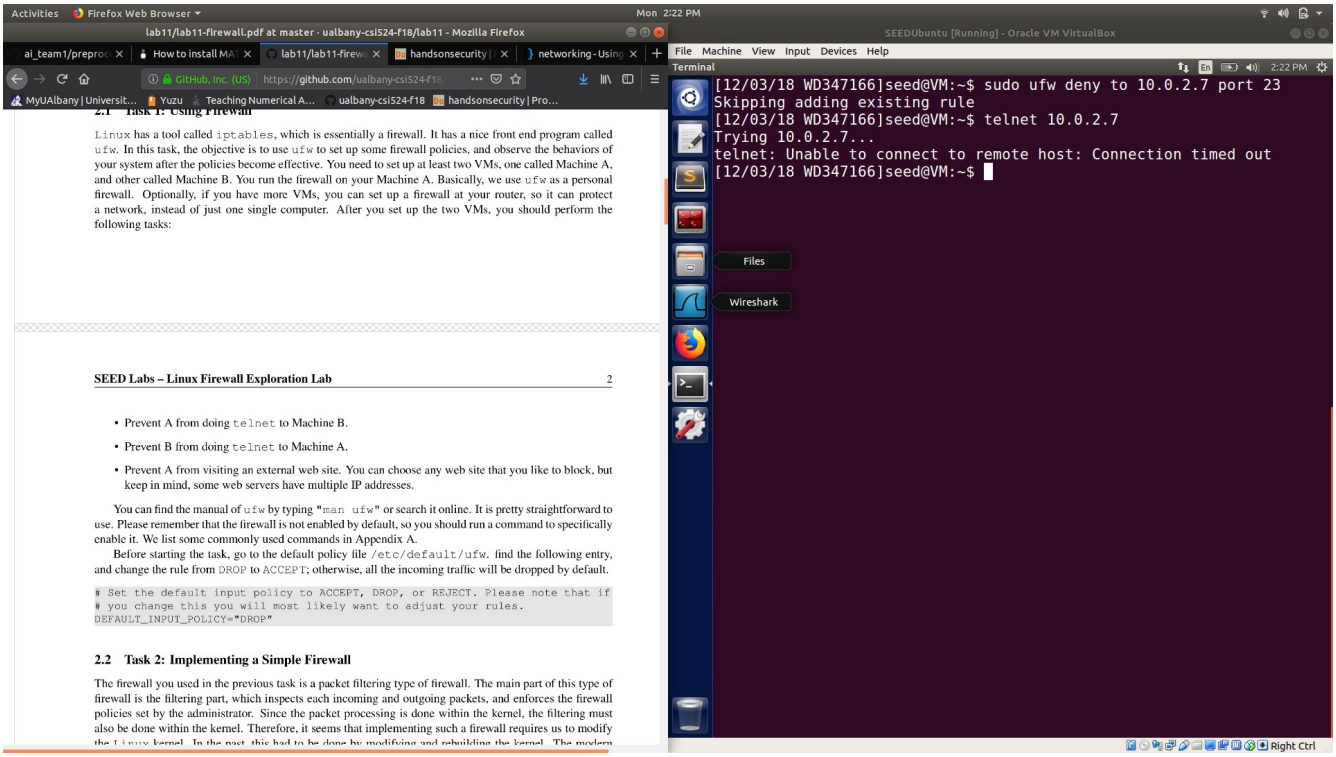
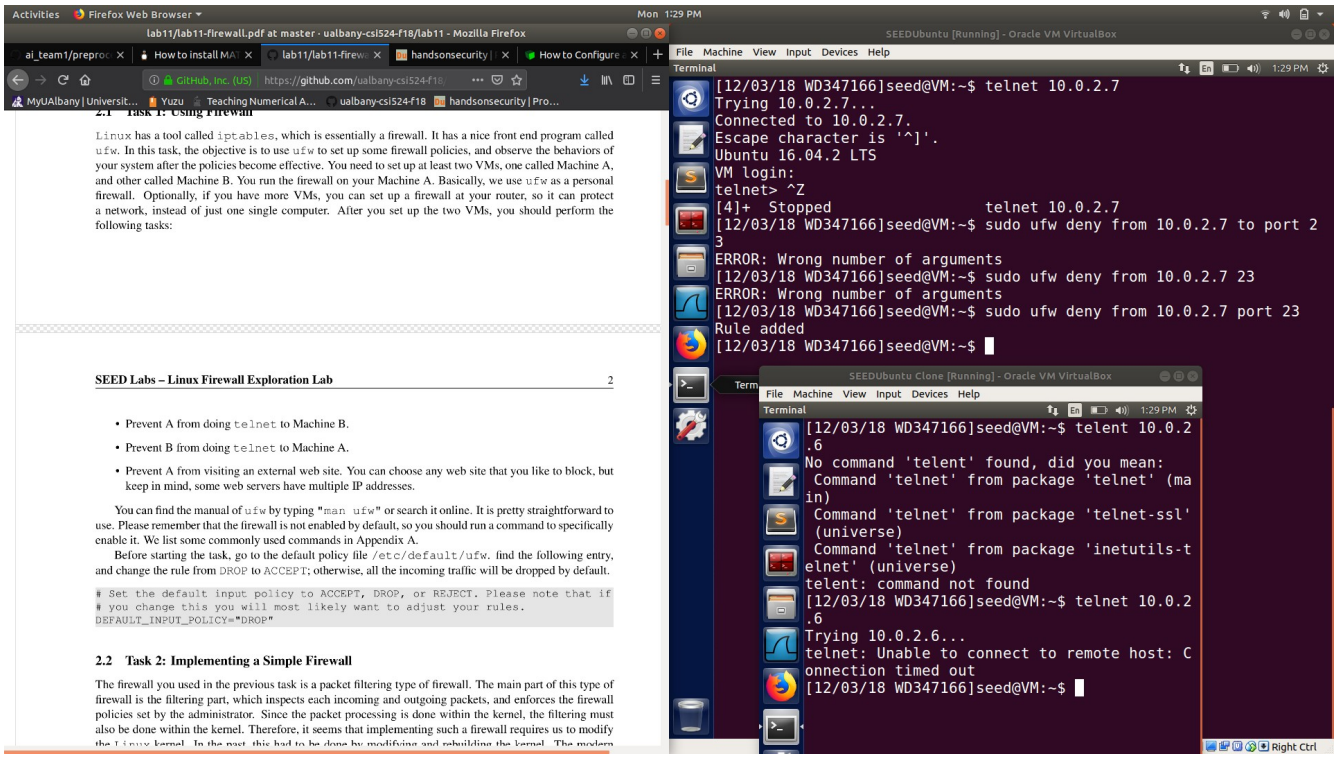
William Dahl
ICSI 424 Information Security
Lab 11

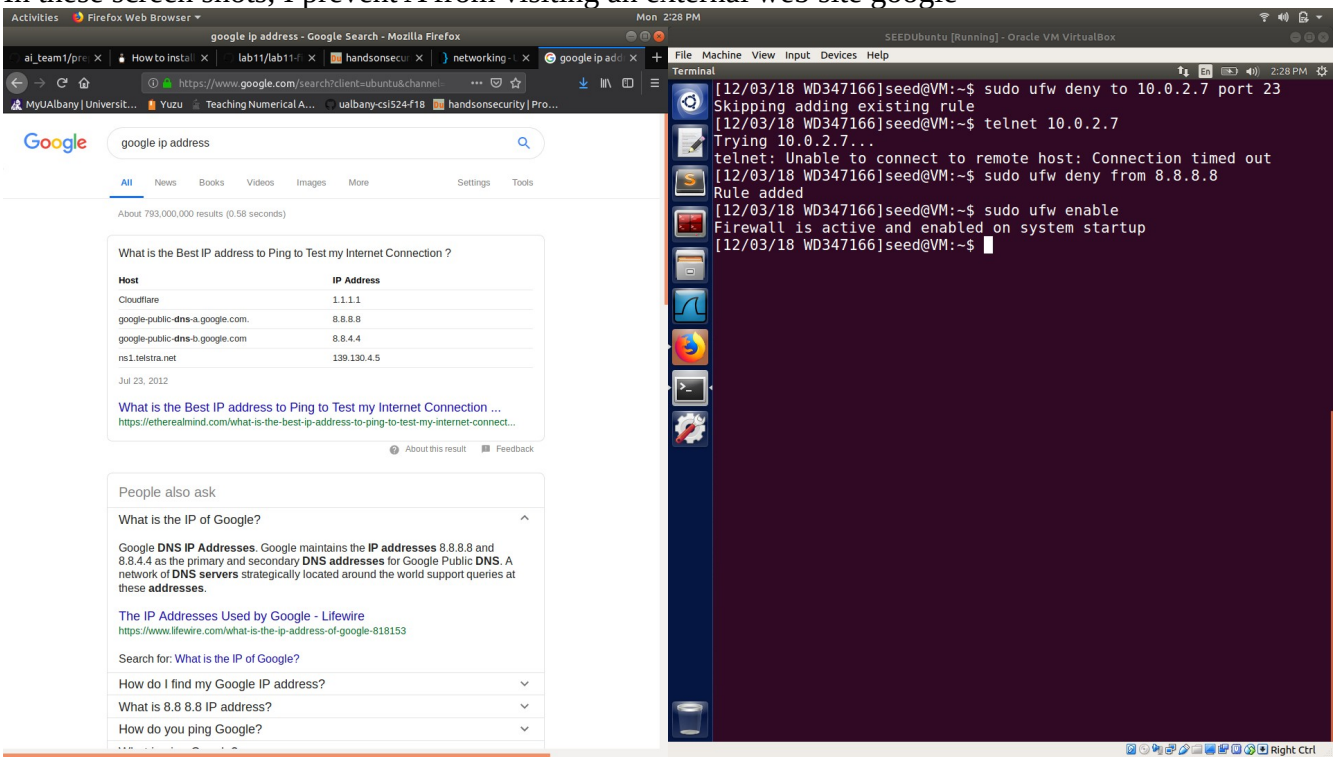
Task 1: In this screen shot I prevent A from doing telnet to machine B

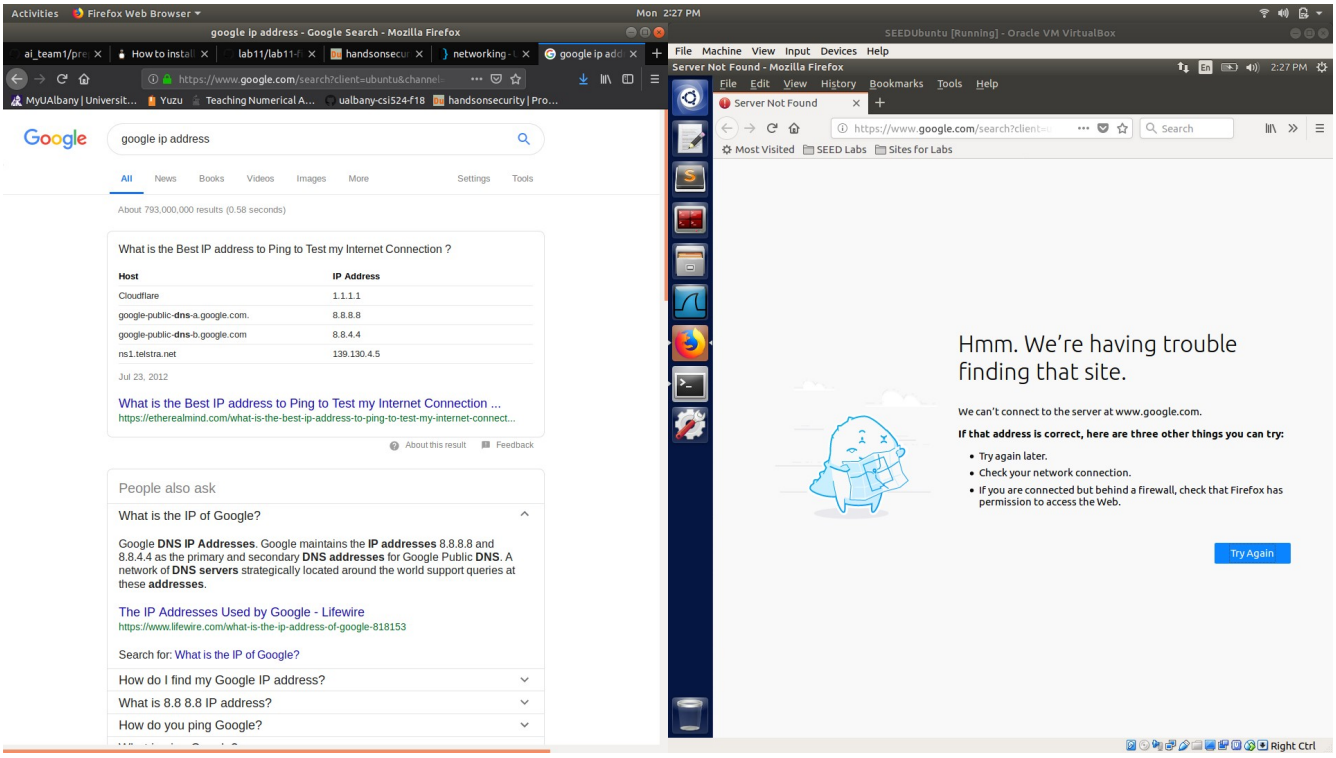


In this screen shot I prevent B from doing telnet to machine A



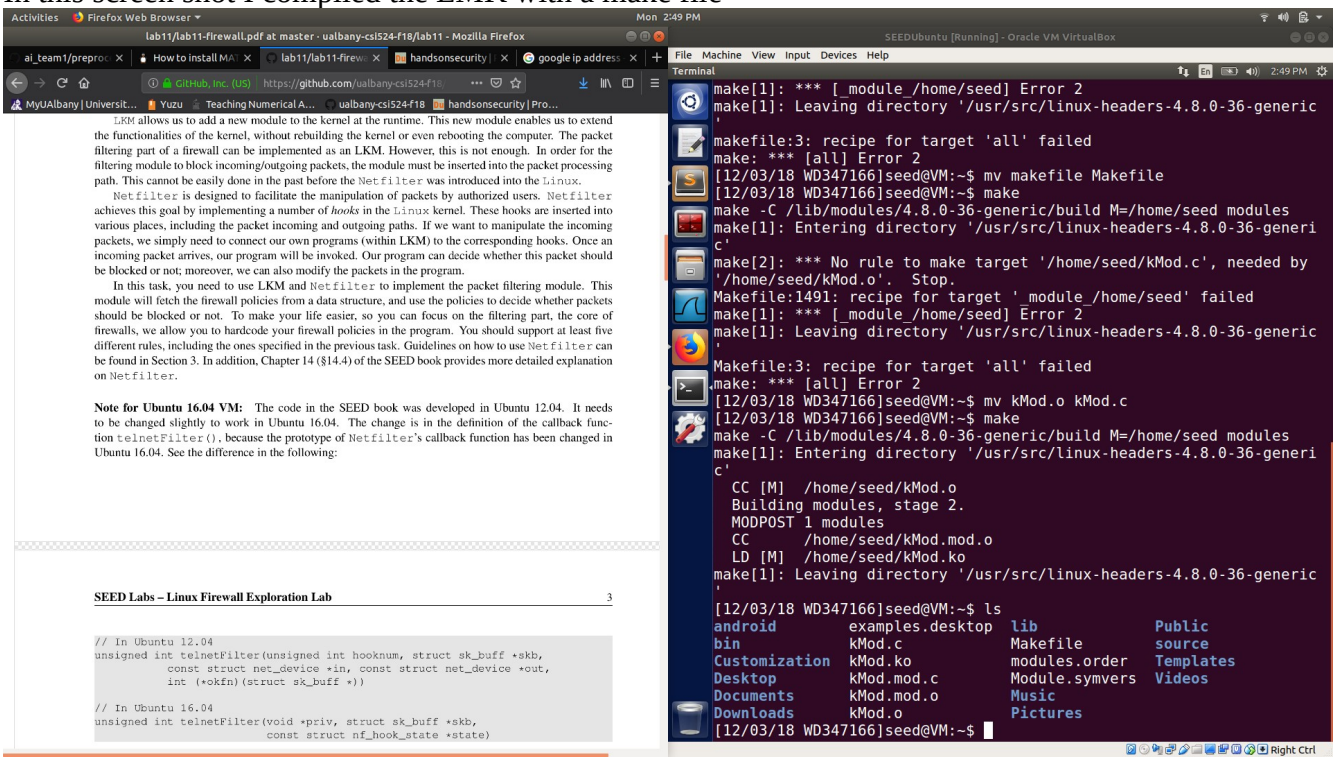
In these screen shots, I prevent A from visiting an external web site google





Task 2:

In this screen shot I complied the LMK with a make file



LKM allows us to add a new module to the kernel at the runtime. This new module enables us to extend the functionalities of the kernel, without rebuilding the kernel or even rebooting the computer. The packet filtering part of a firewall can be implemented as an LKM. However, this is not enough. In order for the filtering module to block incoming/outgoing packets, the module must be inserted into the packet processing path. This cannot be easily done in the past before the `Netfilter` was introduced into the `Linux`.

`Netfilter` is designed to facilitate the manipulation of packets by authorized users. `Netfilter` achieves this goal by implementing a number of *hooks* in the `Linux` kernel. These hooks are inserted into various places, including the packet incoming and outgoing paths. If we want to manipulate the incoming packets, we simply need to connect our own programs (within LKM) to the corresponding hooks. Once an incoming packet arrives, our program will be invoked. Our program can decide whether this packet should be blocked or not; moreover, we can also modify the packets in the program.

In this task, you need to use LKM and `Netfilter` to implement the packet filtering module. This module will fetch the firewall policies from a data structure, and use the policies to decide whether packets should be blocked or not. To make your life easier, so you can focus on the filtering part, the core of firewalls, we allow you to hardcode your firewall policies in the program. You should support at least five different rules, including the ones specified in the previous task. Guidelines on how to use `Netfilter` can be found in Section 3. In addition, Chapter 14 (§14.4) of the SEED book provides more detailed explanation on `Netfilter`.

Note for Ubuntu 16.04 VM: The code in the SEED book was developed in Ubuntu 12.04. It needs to be changed slightly to work in Ubuntu 16.04. The change is in the definition of the callback function `telnetFilter()`, because the prototype of `Netfilter`'s callback function has been changed in Ubuntu 16.04. See the difference in the following:

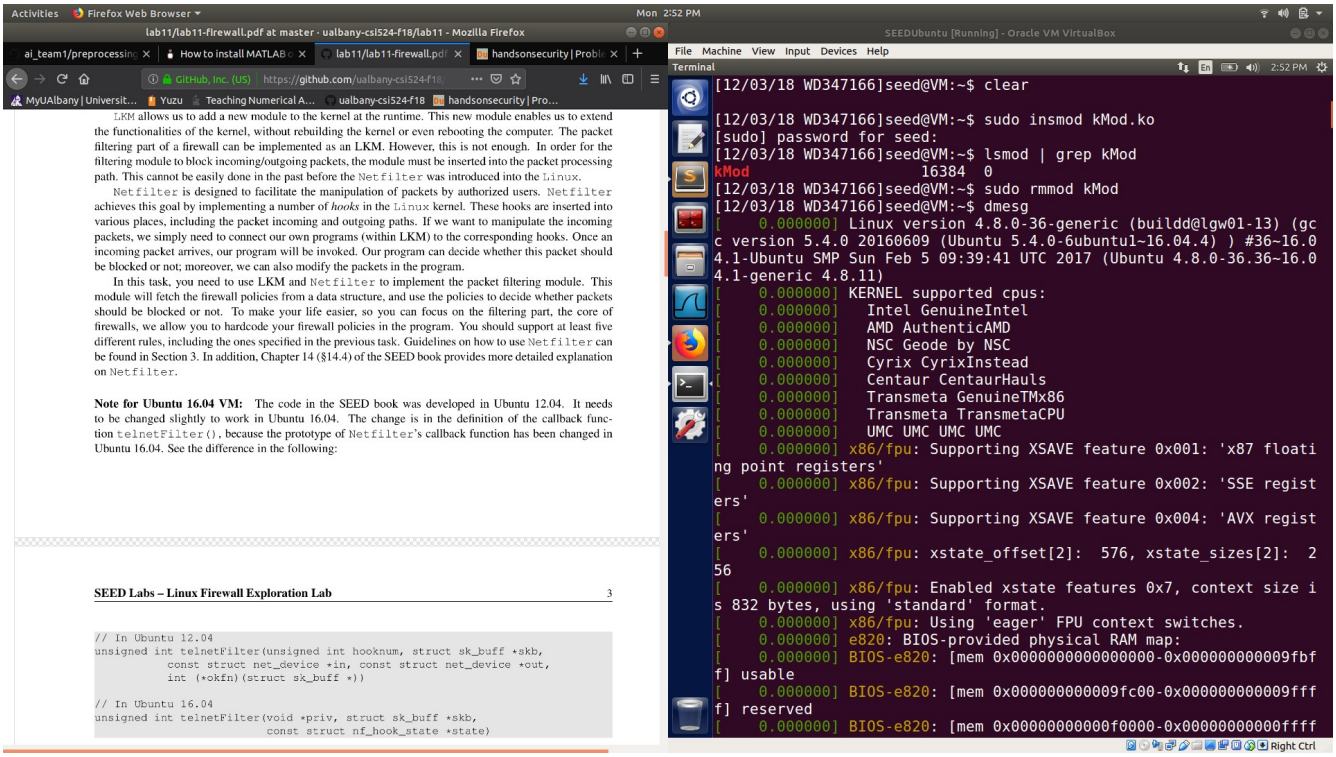
SEED Labs – Linux Firewall Exploration Lab 3

```
// In Ubuntu 12.04
unsigned int telnetFilter(unsigned int hooknum, struct sk_buff *skb,
          const struct net_device *in, const struct net_device *out,
          int (*okfn)(struct sk_buff *))

// In Ubuntu 16.04
unsigned int telnetFilter(void *priv, struct sk_buff *skb,
                          const struct nf_hook_state *state)
```

```
make[1]: *** [_module_/home/seed] Error 2
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic
'
makefile:3: recipe for target 'all' failed
make: *** [all] Error 2
[12/03/18 WD347166]seed@VM:~$ mv makefile Makefile
[12/03/18 WD347166]seed@VM:~$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generi
c'
make[2]: *** No rule to make target '/home/seed/kMod.c', needed by
'/home/seed/kMod.o'.  Stop.
Makefile:1491: recipe for target '_module_/home/seed' failed
make[1]: *** [_module_/home/seed] Error 2
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic
'
Makefile:3: recipe for target 'all' failed
make: *** [all] Error 2
[12/03/18 WD347166]seed@VM:~$ mv kMod.o kMod.c
[12/03/18 WD347166]seed@VM:~$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generi
c'
  CC [M]  /home/seed/kMod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/kMod.mod.o
  LD [M]  /home/seed/kMod.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic
'
[12/03/18 WD347166]seed@VM:~$ ls
android         examples.desktop  lib             Public
bin             kMod.c            Makefile        source
Customization   kMod.ko           modules.order   Templates
Desktop         kMod.mod.c        Module.symvers  Videos
Documents       kMod.mod.o        Music
Downloads       kMod.o            Pictures
[12/03/18 WD347166]seed@VM:~$
```

In this screen shot I install the kernel module



LKM allows us to add a new module to the kernel at the runtime. This new module enables us to extend the functionalities of the kernel, without rebuilding the kernel or even rebooting the computer. The packet filtering part of a firewall can be implemented as an LKM. However, this is not enough. In order for the filtering module to block incoming/outgoing packets, the module must be inserted into the packet processing path. This cannot be easily done in the past before the `Netfilter` was introduced into the `Linux`.

`Netfilter` is designed to facilitate the manipulation of packets by authorized users. `Netfilter` achieves this goal by implementing a number of *hooks* in the `Linux` kernel. These hooks are inserted into various places, including the packet incoming and outgoing paths. If we want to manipulate the incoming packets, we simply need to connect our own programs (within LKM) to the corresponding hooks. Once an incoming packet arrives, our program will be invoked. Our program can decide whether this packet should be blocked or not; moreover, we can also modify the packets in the program.

In this task, you need to use LKM and `Netfilter` to implement the packet filtering module. This module will fetch the firewall policies from a data structure, and use the policies to decide whether packets should be blocked or not. To make your life easier, so you can focus on the filtering part, the core of firewalls, we allow you to hardcode your firewall policies in the program. You should support at least five different rules, including the ones specified in the previous task. Guidelines on how to use `Netfilter` can be found in Section 3. In addition, Chapter 14 (§14.4) of the SEED book provides more detailed explanation on `Netfilter`.

Note for Ubuntu 16.04 VM: The code in the SEED book was developed in Ubuntu 12.04. It needs to be changed slightly to work in Ubuntu 16.04. The change is in the definition of the callback function `telnetFilter()`, because the prototype of `Netfilter`'s callback function has been changed in Ubuntu 16.04. See the difference in the following:

SEED Labs – Linux Firewall Exploration Lab 3

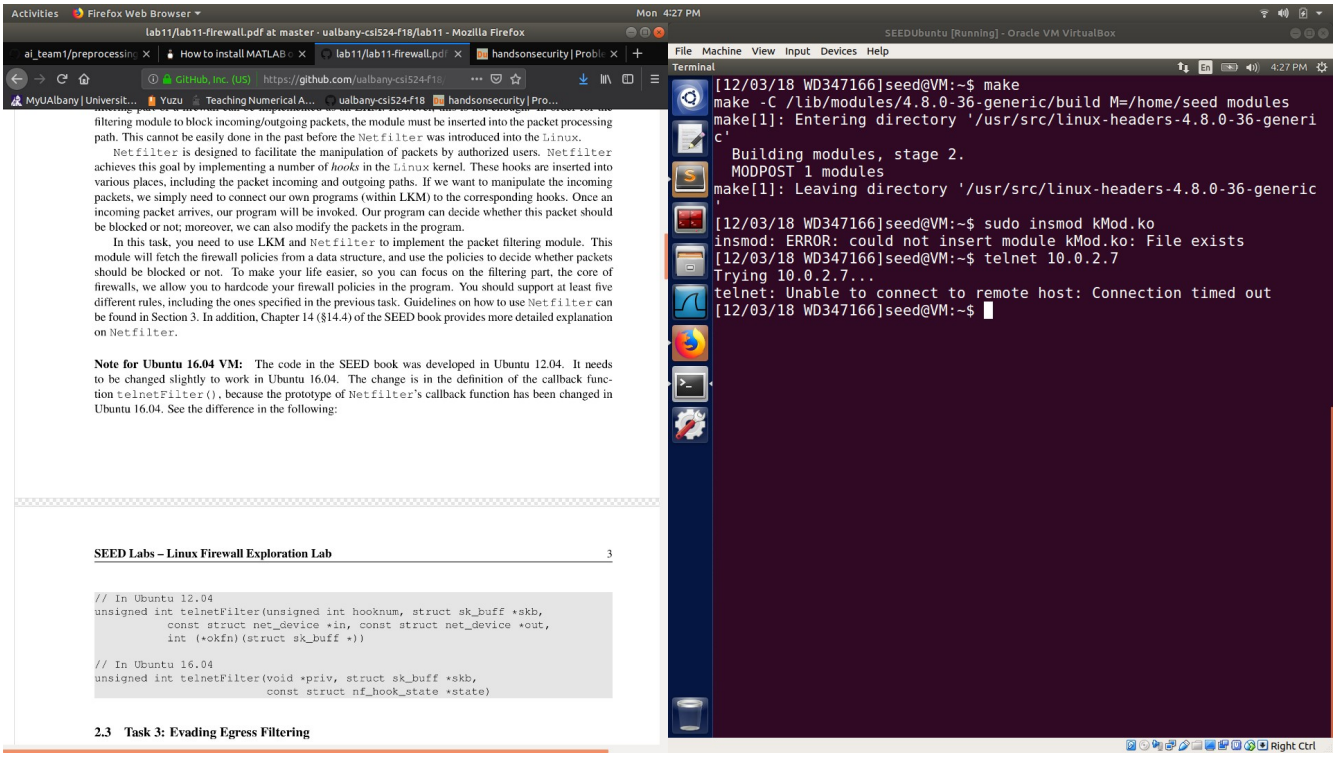
```
// In Ubuntu 12.04
unsigned int telnetFilter(unsigned int hooknum, struct sk_buff *skb,
          const struct net_device *in, const struct net_device *out,
          int (*okfn)(struct sk_buff *))

// In Ubuntu 16.04
unsigned int telnetFilter(void *priv, struct sk_buff *skb,
                          const struct nf_hook_state *state)
```

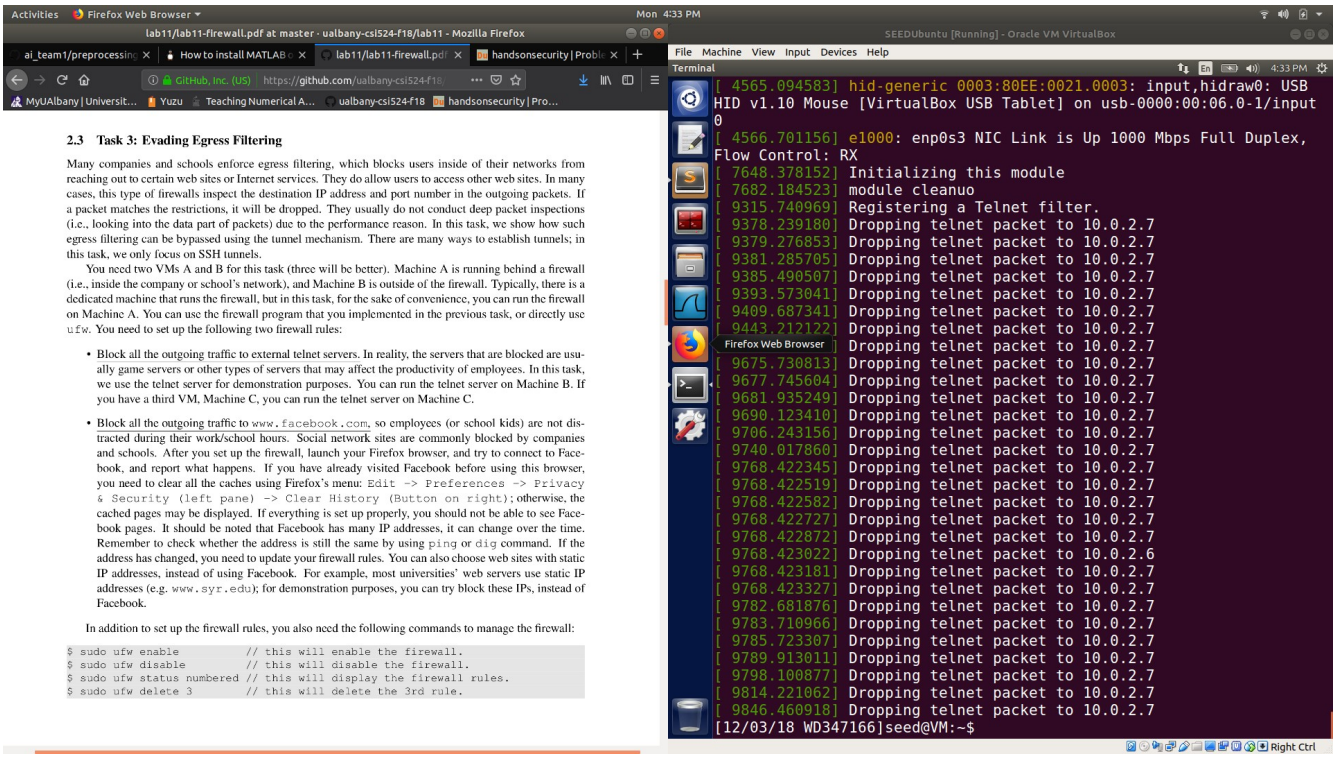
```
[12/03/18 WD347166]seed@VM:~$ clear

[12/03/18 WD347166]seed@VM:~$ sudo insmod kMod.ko
[sudo] password for seed:
[12/03/18 WD347166]seed@VM:~$ lsmod | grep kMod
kMod                   16384  0
[12/03/18 WD347166]seed@VM:~$ sudo rmmod kMod
[12/03/18 WD347166]seed@VM:~$ dmesg
[    0.000000] Linux version 4.8.0-36-generic (buildd@lgw01-13) (gc
c version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4) ) #36~16.0
4.1-Ubuntu SMP Sun Feb 5 09:39:41 UTC 2017 (Ubuntu 4.8.0-36.36~16.0
4.1-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: 'x87 floati
ng point registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE regist
ers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX regist
ers'
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  2
56
[    0.000000] x86/fpu: Enabled xstate features 0x7, context size i
s 832 bytes, using 'standard' format.
[    0.000000] x86/fpu: Using 'eager' FPU context switches.
[    0.000000] e820: BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbf
f] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009fff
f] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000ffff
```

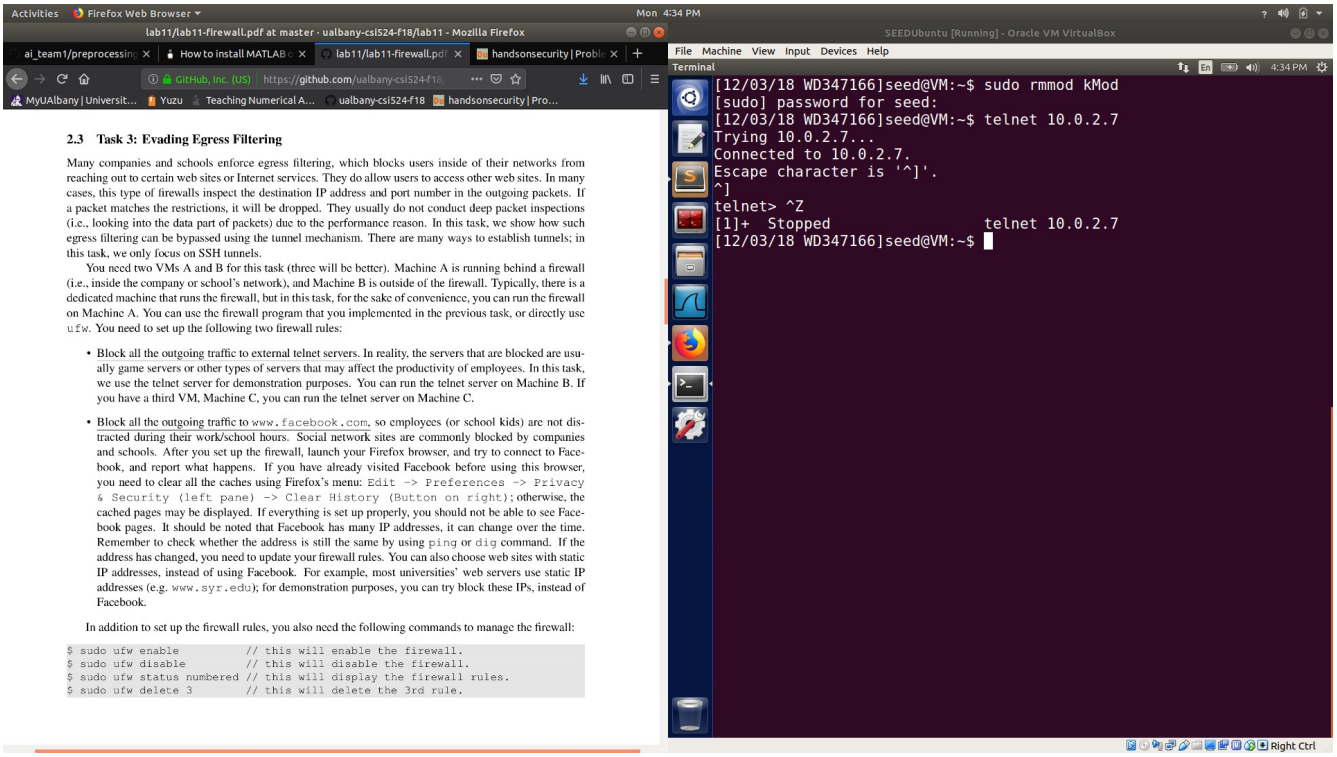
In this screen shot I show that the attempt to connect to 10.0.2.7 was not successful because of our firewall.



In This screen shot I show the messages displayed by our kernel module:

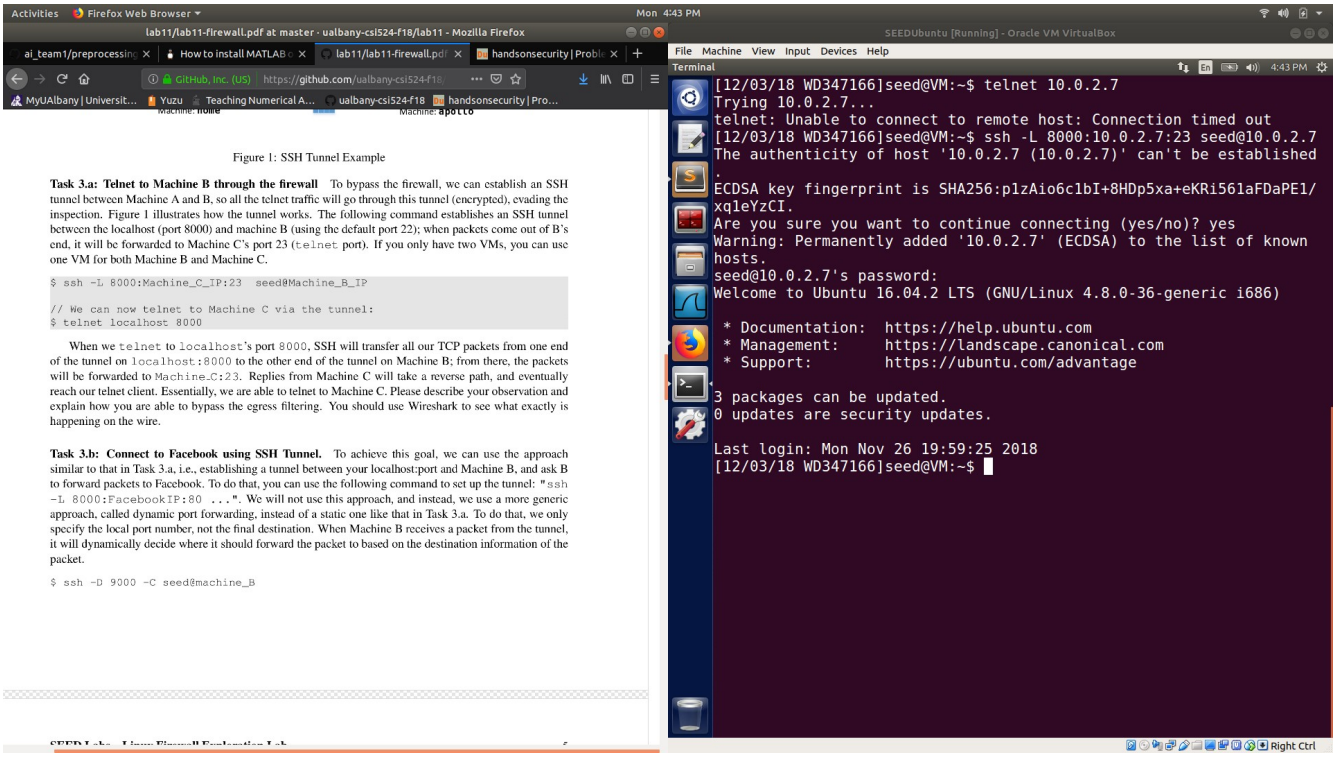


In this screen shot I show what after removing the kernel module we are able to connect to 10.0.2.7 via telnet.



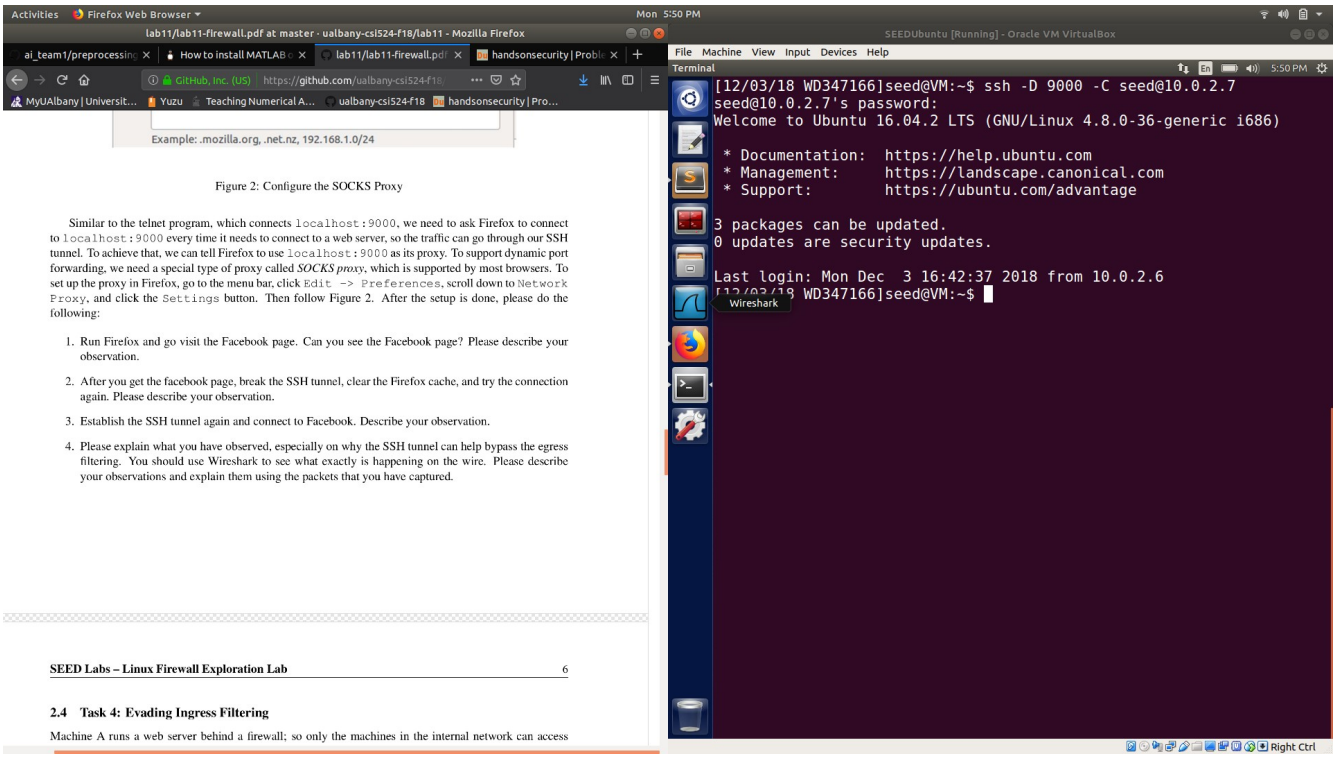
Task 3:

a.) In this screen shot I show that I was bale to connect to 10.0.2.7 using ssh but was not table to connect to it using telnet

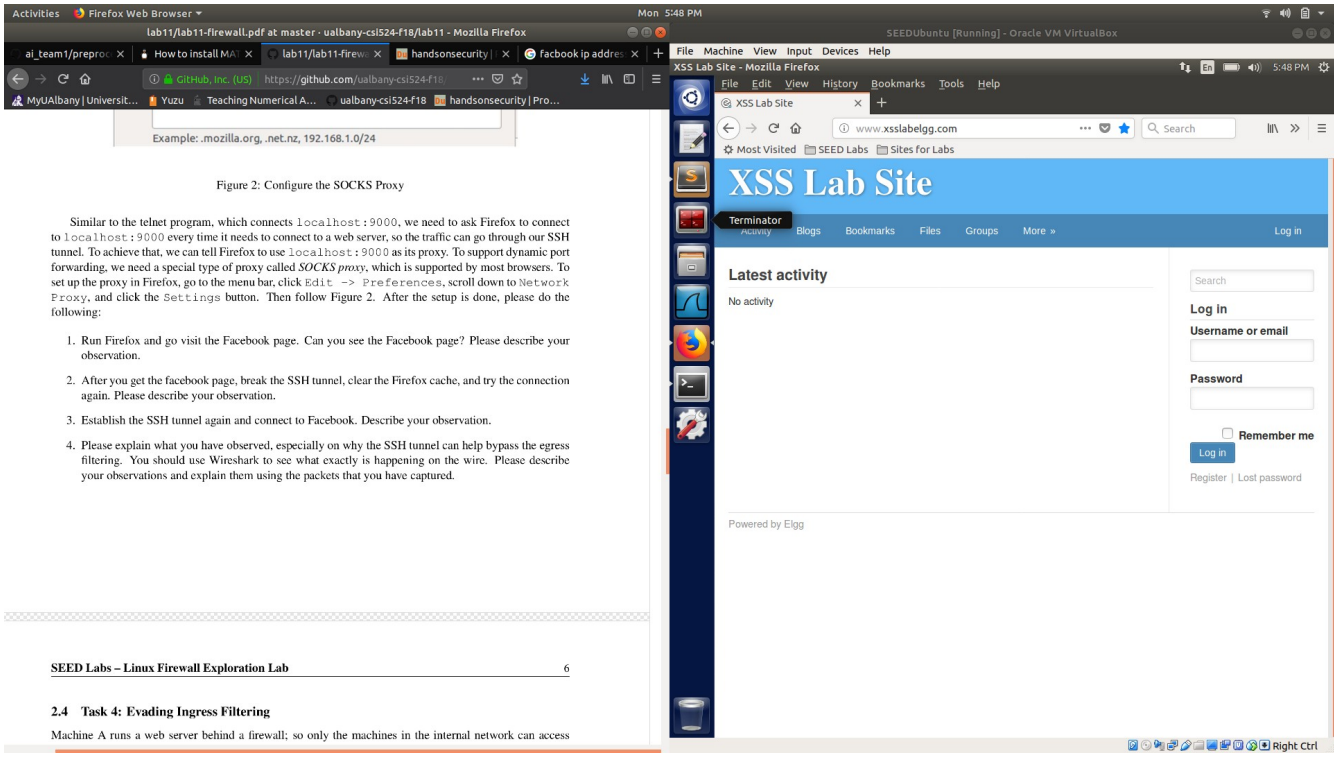


I was able to connect to 10.0.2.7 because when I established the ssh pipe between the two machines all of the tcp packets are sent through the shh pipe which is encrypted and thus hidden from the firewall because the firewall cannot tell if the packets are TCP packets or not. The tunnel receives TCP packets from the telnet client. It forwards the TCP data to the machine B end, from where the data is out in another TCP packet which is sent to machine B. The firewall can only see the traffic between A and B and not from B to A. Also ssh traffic is encrypted.

b.) In this screen shot I create my dynamic ssh tunnel

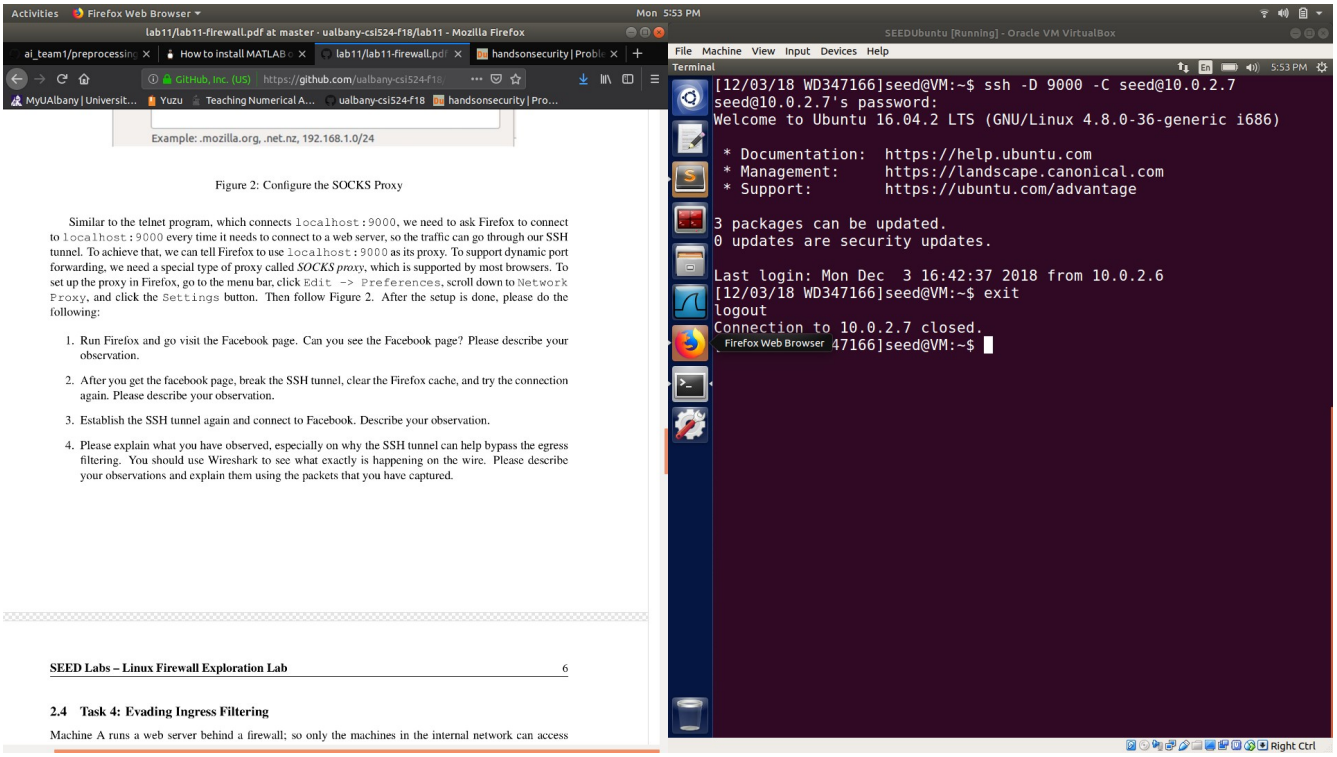


In this screen shot I show that I am able to connect to the XSS lab site through the an ssh tunnel (I used this site instead of facebook because I cannot get my vm to connect to the internet)

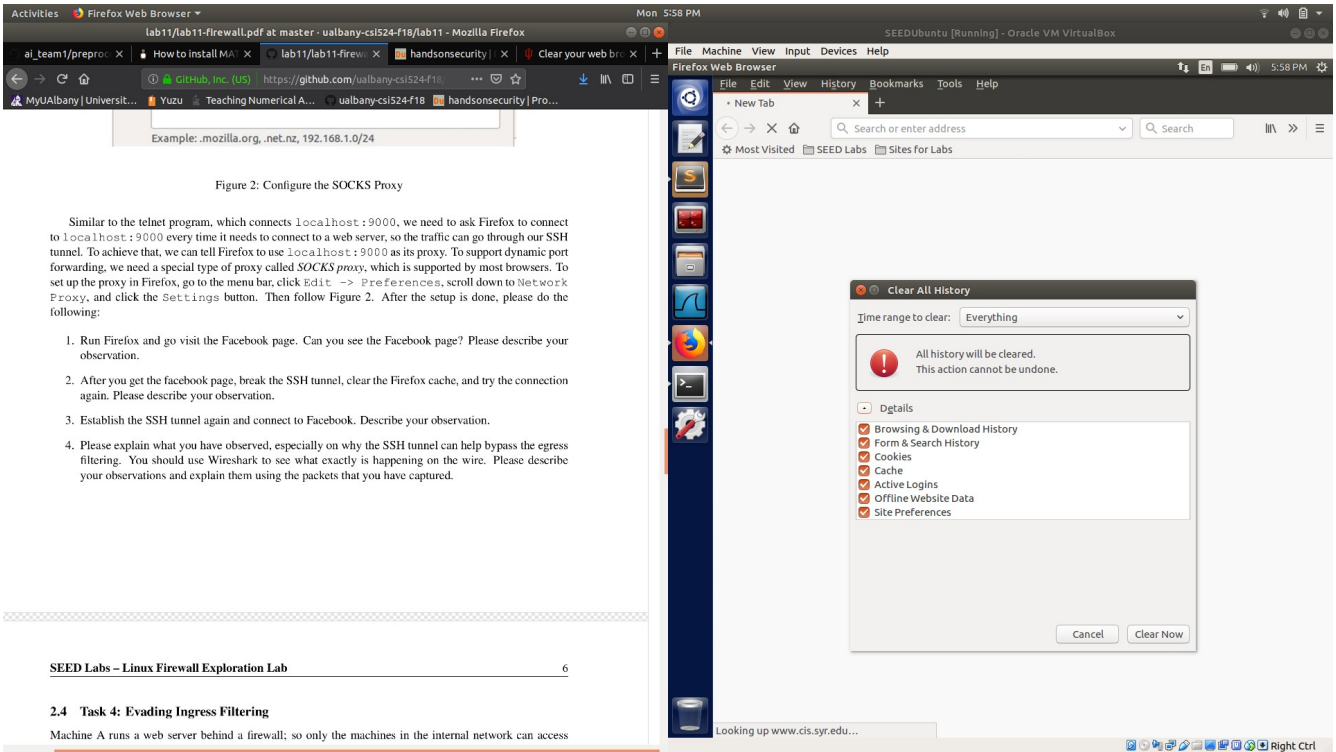


Here I am able to view the site because of the ssh tunnel that I set up. We establishes the tunnel between the localhost and machine B. We configured the browser in such a way that all the request should go through localhost:9000, treating it as a proxy. The Dynamic port forwarding that we set up using ssh is a SOCKS proxy. With the browser configured, we can type the URL of any blacked site which will connect to ssh proxy at port 9000 on the localhost. Ssh will sen the TCP data over the tunnel to the machine B which will communicate with the blocked site.

In this screen shot I break the ssh tunnel



In this screen sot I clear the Firefox cache



In this screen shot I try to connect to the site again with out the ssh tunnel

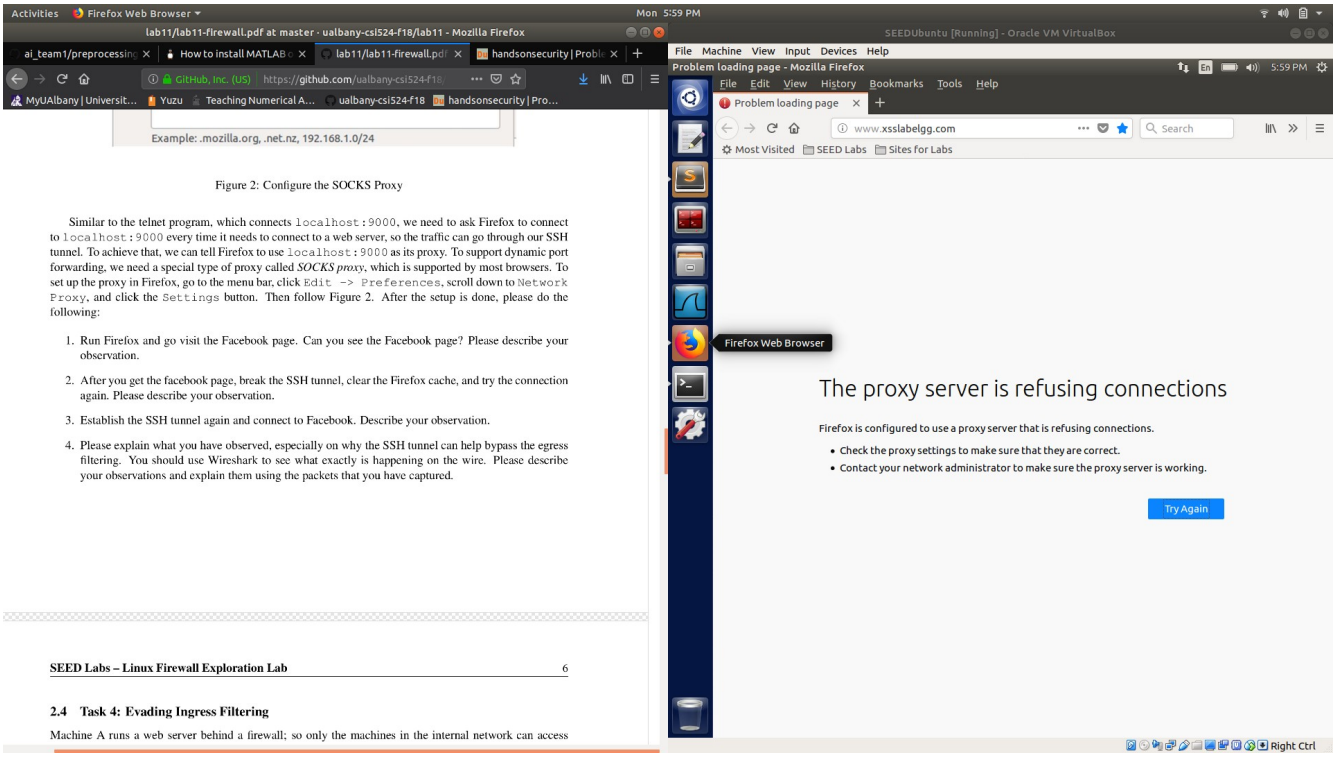


Figure 2: Configure the SOCKS Proxy

Similar to the telnet program, which connects `localhost:9000`, we need to ask Firefox to connect to `localhost:9000` every time it needs to connect to a web server, so the traffic can go through our SSH tunnel. To achieve that, we can tell Firefox to use `localhost:9000` as its proxy. To support dynamic port forwarding, we need a special type of proxy called *SOCKS proxy*, which is supported by most browsers. To set up the proxy in Firefox, go to the menu bar, click `Edit -> Preferences`, scroll down to `Network Proxy`, and click the `Settings` button. Then follow Figure 2. After the setup is done, please do the following:

1. Run Firefox and go visit the Facebook page. Can you see the Facebook page? Please describe your observation.
2. After you get the facebook page, break the SSH tunnel, clear the Firefox cache, and try the connection again. Please describe your observation.
3. Establish the SSH tunnel again and connect to Facebook. Describe your observation.
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.

2.4 Task 4: Evading Ingress Filtering

Machine A runs a web server behind a firewall; so only the machines in the internal network can access

As you can see with out the tunnel we are not able to access the site and we get the error “The Proxy is refusing connections”. This is because the firewall that we created is acting as a proxy and is dropping all of the tcp connections being sent to the machine.

In this screen shot I establish the ssh tunnel again

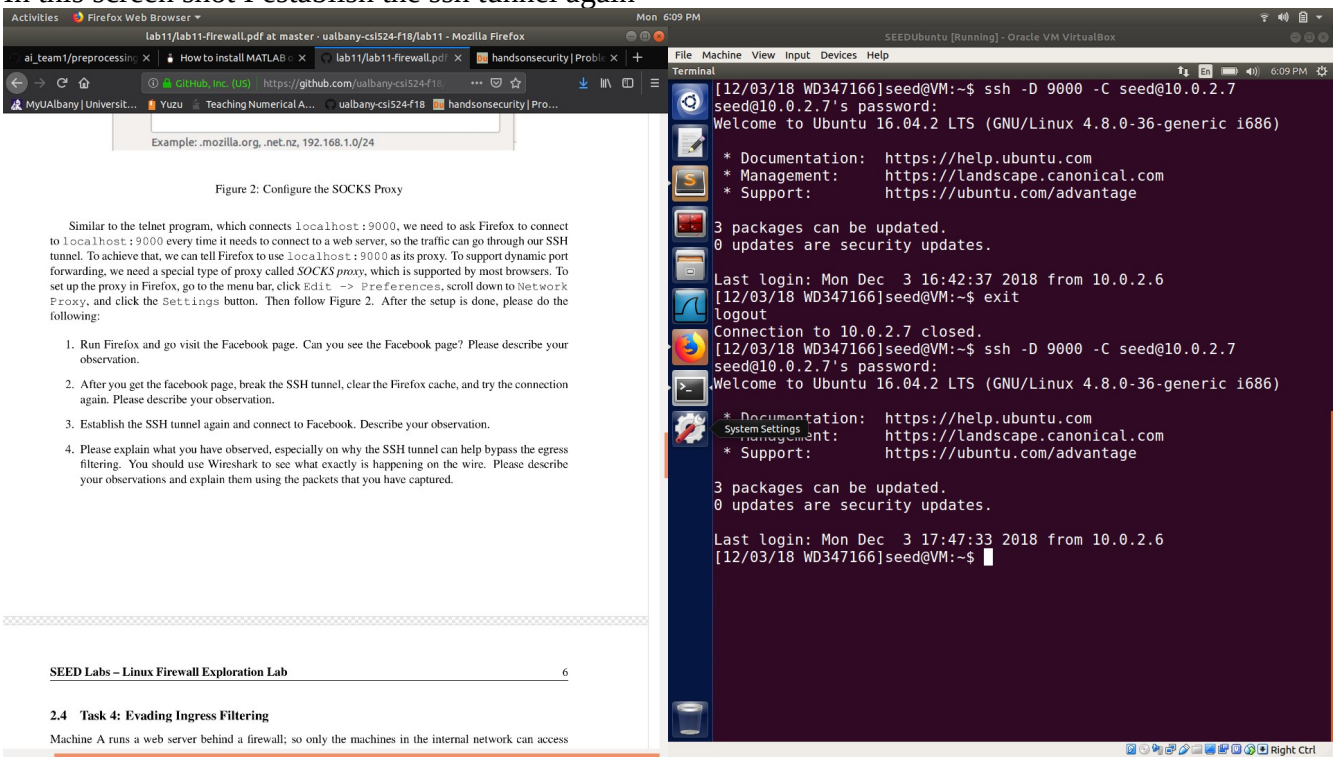


Figure 2: Configure the SOCKS Proxy

Similar to the telnet program, which connects `localhost:9000`, we need to ask Firefox to connect to `localhost:9000` every time it needs to connect to a web server, so the traffic can go through our SSH tunnel. To achieve that, we can tell Firefox to use `localhost:9000` as its proxy. To support dynamic port forwarding, we need a special type of proxy called *SOCKS proxy*, which is supported by most browsers. To set up the proxy in Firefox, go to the menu bar, click `Edit -> Preferences`, scroll down to `Network Proxy`, and click the `Settings` button. Then follow Figure 2. After the setup is done, please do the following:

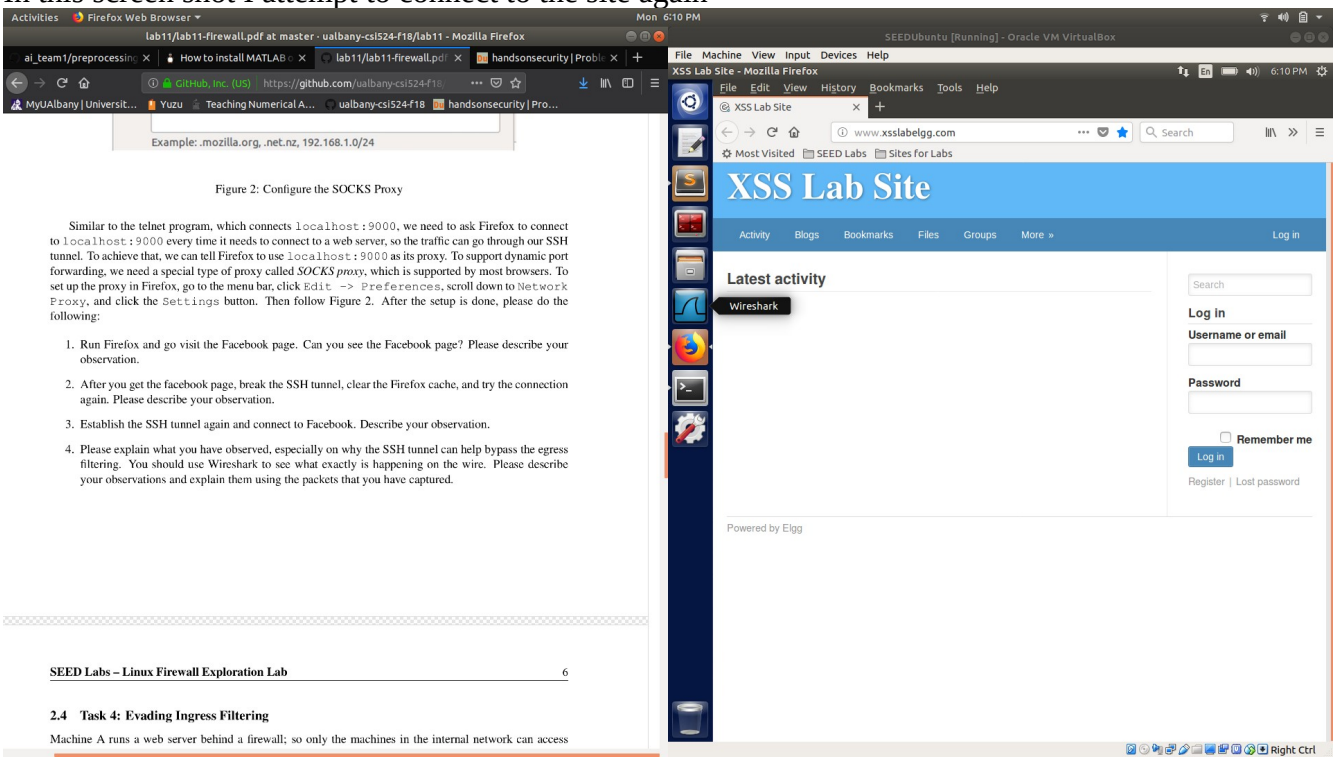
1. Run Firefox and go visit the Facebook page. Can you see the Facebook page? Please describe your observation.
2. After you get the facebook page, break the SSH tunnel, clear the Firefox cache, and try the connection again. Please describe your observation.
3. Establish the SSH tunnel again and connect to Facebook. Describe your observation.
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.

SEED Labs – Linux Firewall Exploration Lab 6

2.4 Task 4: Evading Ingress Filtering

Machine A runs a web server behind a firewall; so only the machines in the internal network can access

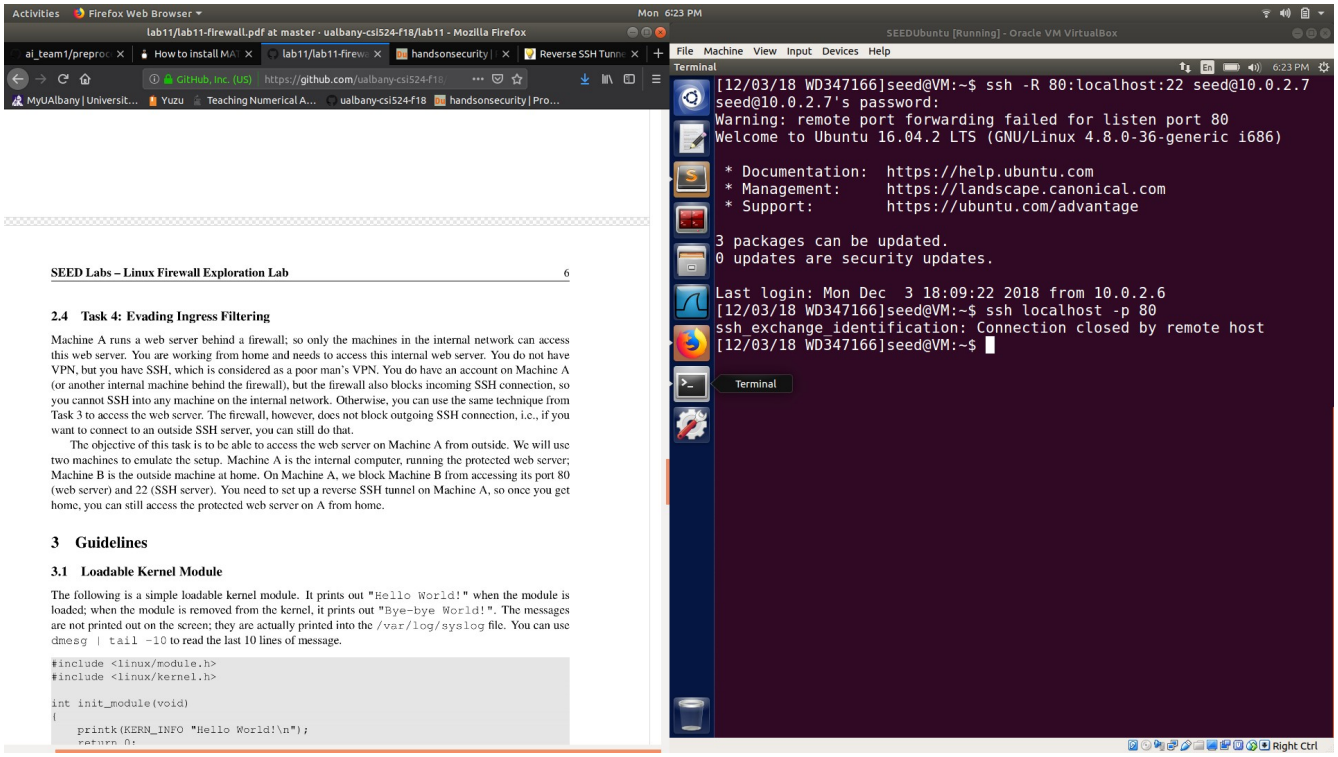
In this screen shot I attempt to connect to the site again



I was successful in connecting to the site because the ssh tunnel was re established and the tcp packets were sent to machine B.

Here I am able to view the site because of the ssh tunnel that I set up. We establishes the tunnel between the localhost and machine B. We configured the browser in such a way that all the request should go through localhost:9000, treating it as a proxy. The Dynamic port forwarding that we set up using ssh is a SOCKS proxy. With the browser configured, we can type the URL of any blacked site which will connect to ssh proxy at port 9000 on the localhost. Ssh will sen the TCP data over the tunnel to the machine B which will communicate with the blocked site. Also the ssh packets are encrypted so their information is hidden from the firewall.

Task 4: In this screen shot I show how I created my reverse ssh tunnel on machine A



SEED Labs – Linux Firewall Exploration Lab 6

2.4 Task 4: Evading Ingress Filtering

Machine A runs a web server behind a firewall; so only the machines in the internal network can access this web server. You are working from home and needs to access this internal web server. You do not have VPN, but you have SSH, which is considered as a poor man's VPN. You do have an account on Machine A (or another internal machine behind the firewall), but the firewall also blocks incoming SSH connection, so you cannot SSH into any machine on the internal network. Otherwise, you can use the same technique from Task 3 to access the web server. The firewall, however, does not block outgoing SSH connection, i.e., if you want to connect to an outside SSH server, you can still do that.

The objective of this task is to be able to access the web server on Machine A from outside. We will use two machines to emulate the setup. Machine A is the internal computer, running the protected web server; Machine B is the outside machine at home. On Machine A, we block Machine B from accessing its port 80 (web server) and 22 (SSH server). You need to set up a reverse SSH tunnel on Machine A, so once you get home, you can still access the protected web server on A from home.

3 Guidelines

3.1 Loadable Kernel Module

The following is a simple loadable kernel module. It prints out "Hello World!" when the module is loaded; when the module is removed from the kernel, it prints out "Bye-bye World!". The messages are not printed out on the screen; they are actually printed into the /var/log/syslog file. You can use dmesg | tail -10 to read the last 10 lines of message.

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

int init_module(void)
{
    printk(KERN_INFO "Hello World!\n");
    return 0;
```

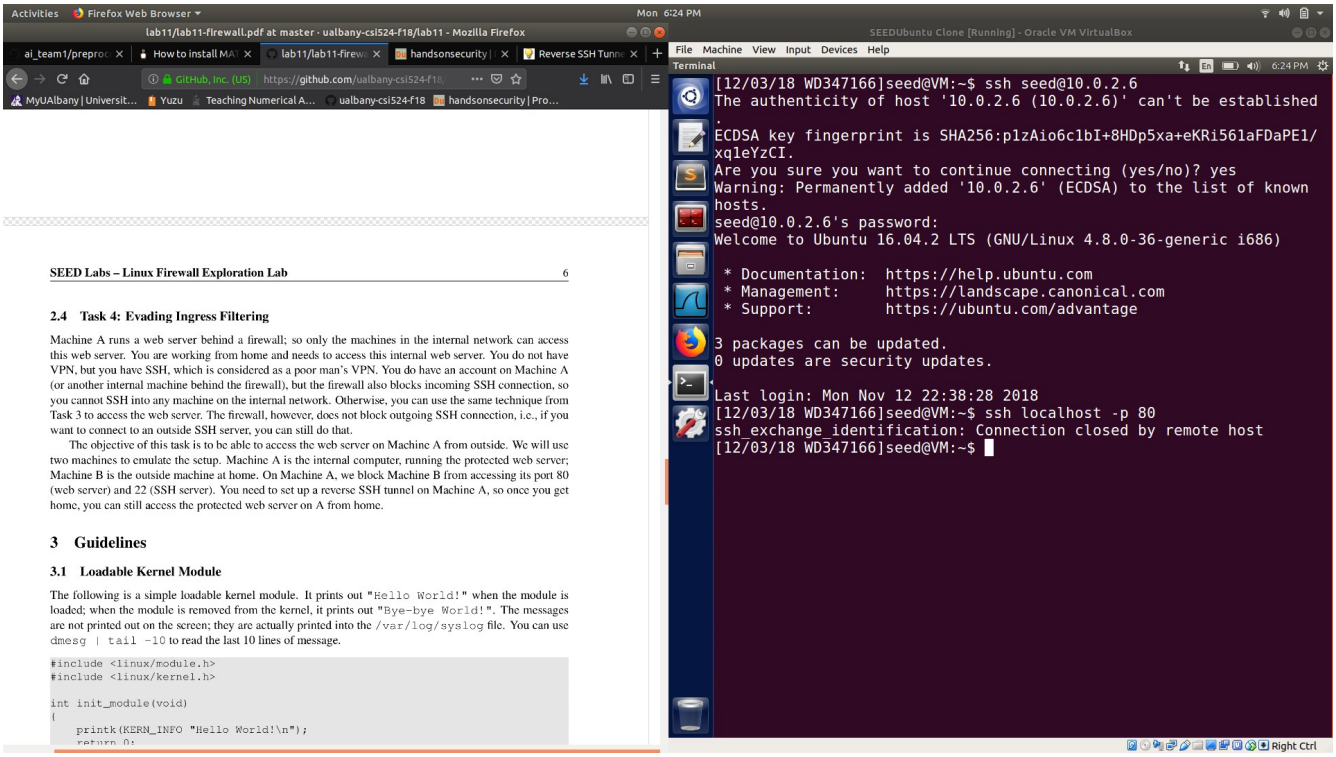
```
[12/03/18 WD347166]seed@VM:~$ ssh -R 80:localhost:22 seed@10.0.2.7
seed@10.0.2.7's password:
Warning: remote port forwarding failed for listen port 80
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

3 packages can be updated.
0 updates are security updates.

Last login: Mon Dec  3 18:09:22 2018 from 10.0.2.6
[12/03/18 WD347166]seed@VM:~$ ssh localhost -p 80
ssh_exchange_identification: Connection closed by remote host
[12/03/18 WD347166]seed@VM:~$
```

In this screen shot I show that I ma now able to access machine A from machine B



SEED Labs – Linux Firewall Exploration Lab 6

2.4 Task 4: Evading Ingress Filtering

Machine A runs a web server behind a firewall; so only the machines in the internal network can access this web server. You are working from home and needs to access this internal web server. You do not have VPN, but you have SSH, which is considered as a poor man's VPN. You do have an account on Machine A (or another internal machine behind the firewall), but the firewall also blocks incoming SSH connection, so you cannot SSH into any machine on the internal network. Otherwise, you can use the same technique from Task 3 to access the web server. The firewall, however, does not block outgoing SSH connection, i.e., if you want to connect to an outside SSH server, you can still do that.

The objective of this task is to be able to access the web server on Machine A from outside. We will use two machines to emulate the setup. Machine A is the internal computer, running the protected web server; Machine B is the outside machine at home. On Machine A, we block Machine B from accessing its port 80 (web server) and 22 (SSH server). You need to set up a reverse SSH tunnel on Machine A, so once you get home, you can still access the protected web server on A from home.

3 Guidelines

3.1 Loadable Kernel Module

The following is a simple loadable kernel module. It prints out "Hello World!" when the module is loaded; when the module is removed from the kernel, it prints out "Bye-bye World!". The messages are not printed out on the screen; they are actually printed into the /var/log/syslog file. You can use dmesg | tail -10 to read the last 10 lines of message.

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

int init_module(void)
{
    printk(KERN_INFO "Hello World!\n");
    return 0;
```

```
[12/03/18 WD347166]seed@VM:~$ ssh seed@10.0.2.6
The authenticity of host '10.0.2.6 (10.0.2.6)' 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 '10.0.2.6' (ECDSA) to the list of known
hosts.
seed@10.0.2.6'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

3 packages can be updated.
0 updates are security updates.

Last login: Mon Nov 12 22:38:28 2018
[12/03/18 WD347166]seed@VM:~$ ssh localhost -p 80
ssh_exchange_identification: Connection closed by remote host
[12/03/18 WD347166]seed@VM:~$
```