LINUX FIREWALL EXPLORATION LAB

- Arvind Ponnarassery Jayan
(aponnar1)

TASK 0: Initial Setup

For this lab 3 machines A, B and C will be used. Machine A will have an IP address of `10.0.2.7`, Machine B will have an IP address of `10.0.2.5` and Machine C will have an IP address of `10.0.2.6`. This change has been reflected in the file `/etc/hosts` for simplicity. Machine A will have the entry `10.0.2.5 Machine-B` and `10.0.2.6 Machine-C`, Machine B will have the entry `10.0.2.7 Machine-A` and `10.0.2.6 Machine-C` and Machine C will have the entry `10.0.2.7 Machine-A` and `10.0.2.5 Machine-B`in thier respective files.

TASK 1: Using Firewall

For this task we will be using the Linux tool `iptables`, which essentially is a firewall, for packet filtering.

initial setup
Initially we disable the Uncomplicated FireWall or `ufw`, which is a userfriendly firewall tool in Ubuntu, as shown in Figure 1 .

```
[11/07/19]seed@VM:~$ sudo ufw disable
[sudo] password for seed:
Firewall stopped and disabled on system startup
[11/07/19]seed@VM:~$
```

Figure 1: A - Disable ufw

Then we flush the iptables, to remove any existing firewall rule in place. This is further confirmed by displaying the iptables rules, as shown in Figure 2. We can observe in the output that by flushing the iptables, the firewall is now configured to accept or ALLOW any outbound and inbound packets from anywhere, since it all the rules are removed.

Now the firewall can be configured with the help of `iptables`. For this we refer its manual, as shown in Figure 3.

1. Prevent A from doing telnet to Machine B

For this task we include a rule in the iptables to prevent Machine A to telnet to Machine B. Initially, before setting up the firewall rule, we can see that Machine A can telnet to Machine B as shown in Figure 4.

Now we will add a firewall rule that prevents any packets that are outbound from Machine A that is a TCP protocol to reach a system with destination IP address of 10.0.2.5 and a destination port 23. This essentially prevents a telnet connection from Machine A to Machine B. This is shown in Figure 5.

```
[11/07/19]seed@VM:~$ sudo iptables -F
[11/07/19]seed@VM:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
[11/07/19]seed@VM:~$
```

Figure 2: A - Flush iptables

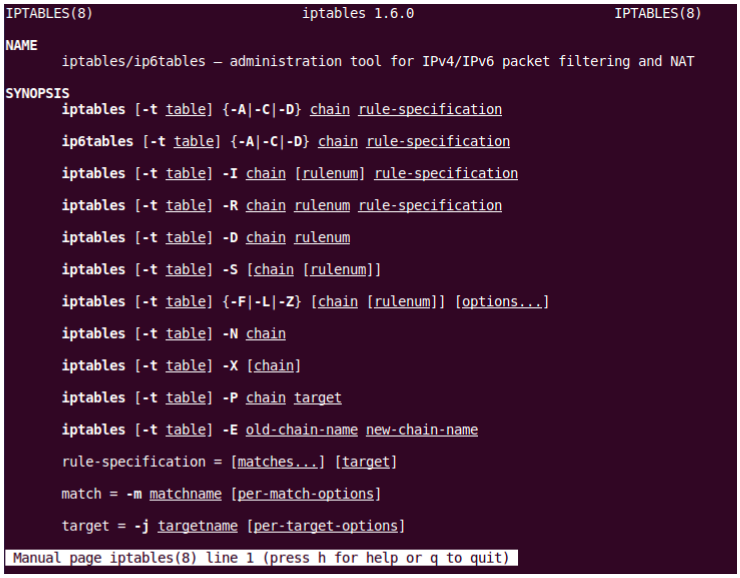


Figure 3: A - man iptables

```
[11/07/19]seed@VM:~$ telnet Machine-B
Trying 10.0.2.5...
Connected to Machine-B.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Oct 28 13:28:04 EDT 2019 from 10.0.2.6 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

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

[11/07/19]seed@VM:~$
```

Figure 4: A - telnet to B

```
[11/07/19]seed@VM:~$ sudo iptables -A OUTPUT -p tcp --dport 23 -j REJECT -d 10.0.2.5
[11/07/19]seed@VM:~$
```

Figure 5: A - iptables REJECT telnet to B command

Now if we were to try the telnet command to Machine B. We will get a `Connection Fail` response. We get this response because we have added a `REJECT` policy in the firewall rule. If we had used a `DROP` policy the firewall wouldn't respond to the system that the action is denied. This is verified as shown in Figure 6.

```
[11/07/19]seed@VM:~$ telnet Machine-B
Trying 10.0.2.5...
telnet: Unable to connect to remote host: Connection refused
[11/08/19]seed@VM:~$
```

Figure 6: A - Prevented Telnet from A to B

The command used is
`sudo iptables -A OUTPUT -p tcp --dport 23 -j REJECT -d 10.0.2.5`
The command specifies to append to the iptables firewall rule using the flag `-A`. It specifies to consider all outbound packets from Machine A using the flag `OUTPUT`. The rule specifies to focus on telnet protocol using the flags `-p tcp` and `--dport 23` (telnet port), since telnet is a TCP connection to the port 23. It also specifies the destination of the packet as Machine B with the `-d 10.0.2.5` flag, as shown in Figure 5. By appending this rule all outbound packets to create a TCP telnet connection with Machine B will be rejected. We should observe that this doesn't prevent Machine B to telnet to Machine A. Also it is understandable that Machine can still telnet to systems other than Machine B.

2. Prevent B from doing telnet to Machine A

For this task we include a rule in the iptables of Machine A to prevent Machine B to telnet to Machine A. Initially, before setting up the firewall rule, we can see that Machine A can telnet to Machine B as shown in Figure 7.

```
[11/08/19]seed@VM:~$ telnet Machine-A
Trying 10.0.2.7...
Connected to Machine-A.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Oct 28 12:37:07 EDT 2019 from 10.0.2.7 on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

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

[11/08/19]seed@VM:~$
```

Figure 7: B - Telnet to A

Now we will add a firewall rule that prevents any packets that are inbound to Machine A from Machine B, that is a TCP protocol also it should have a source IP address of 10.0.2.5 and a destination port 23. This essentially prevents a telnet connection from Machine B to Machine A. This is shown in Figure 8.

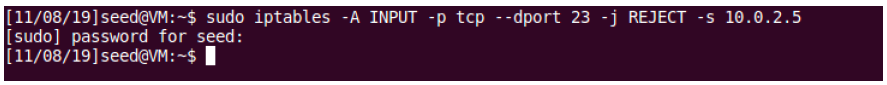


Figure 8: A - iptables REJECT telnet from B command

The command used is
`sudo iptables -A INPUT -p tcp --dport 23 -j REJECT -s 10.0.2.5`
The command specifies to append this REJECT policy rule to the iptables firewall rule using the flag `-A`. It specifies to consider inbound packets from Machine A using the flag `INPUT`. The rule specifies to focus on telnet protocol using the flags `-p tcp` and `--dport 23` (telnet port), and it also specifies the source of the packet should be Machine B with the `-s 10.0.2.5` flag, as shown in Figure 8. By appending this rule all inbound packets to create a TCP telnet connection from Machine B will be rejected. This can be confirmed with an telnet attempt from Machine B to Machine A as shown in Figure 9

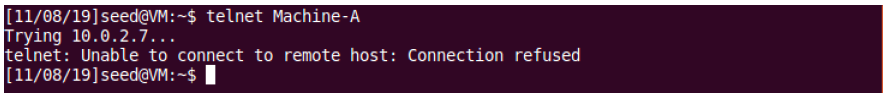


Figure 9: B - Prevented telnet to Machine A

The telnet connection is denied and a response is received by Machine B. We get this response because we have added a `REJECT` policy in the firewall rule. If we had used a `DROP` policy the firewall wouldn't respond to the system that the action is denied. Also observe that this firewall rule alone will only prevent Machine B to connect to A not the other way around.

3. Prevent A from visiting an external web site

For this task we need to block Machine A from accessing an external website. The external website that I have considered is "www.google.com". Initially, before adding the firewall rule, it is seen that Machine A can access the `https://www.google.com` as shown in Figure 10.

Now we will add a firewall rule that prevents all the outbound packets from Machine A to go to the website "www.google.com". The rule should prevent both HTTP and HTTPS requests and also consider all the IP addresses for the website. This is shown in Figure 11.

The command used is
`sudo iptables -A OUTPUT -p tcp -m multiport --dport 80,443 -j REJECT -m string --algo bm --string "www.google.com"`

The command specifies to append the REJECT rule to the iptables firewall rule using the flag `-A`. It specifies to consider outbound packets from Machine A using the flag `INPUT`. The rule specifies to focus on HTTP and HTTPS protocol using the flags `-p tcp` and `-m multiport --dport 80,443` (HTTP and HTTPS ports respectively), and it also specifies if the request contains the string "www.google.com" using `-m string --algo bm --string "www.google.com"` flag, as shown in Figure 11. By appending this rule, all outbound packets that contains a TCP requests through HTTP or HTTPS port with the string "www.google.com" in the request will be rejected. I have used the `--string` flag to filter www.google.com instead of using multiple IP addresses of the website.This can be

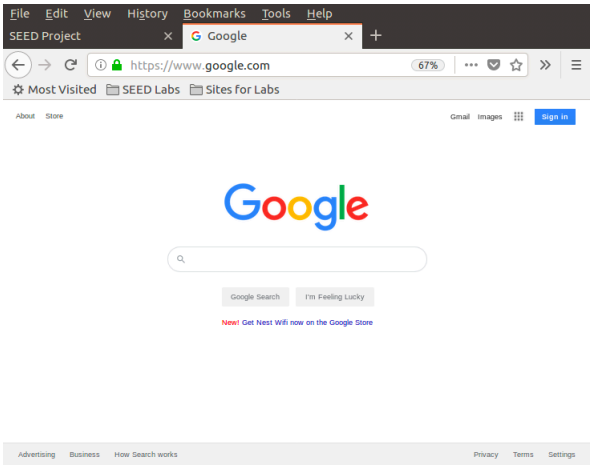


Figure 10: A - Accessing external webserver "www.google.com"

```
[11/08/19]seed@VM:~$ sudo iptables -A OUTPUT -p tcp -m multiport --dport 80,443 -j REJECT
-m string --algo bm --string "www.google.com"
[11/08/19]seed@VM:~$
```

Figure 11: A - Adding rule to prevent Machine A to visit "www.google.com"

confirmed with a request to the website as shown in Figure 12.

Alternately, we could use the `dig` command to obtain the IP addresses associated to the website, and add a rule that blocks all the packets with IP address from the list that was previously obtained.

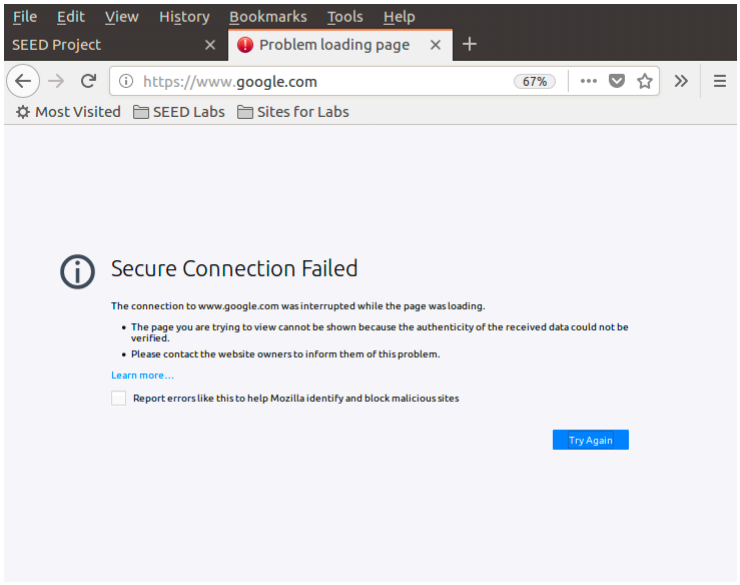


Figure 12: A - Prevented Machine A to connect to "www.google.com"

TASK 2: Implementing a Simple Firewall

Writing a Simple Loadable Kernel Modules

Initially we implement a simple Loadable Kernel Module to understand, this module will print "`Initializing the module`" when loaded and prints "`Module cleanup`" when it is removed. The program `kMod.c` required for loading as a kernel module is shown in Figure 13.

Figure 13: A - Program for simple LKM

Now we create a `Makefile`, as shown in Figure 14, which upon `make` will compile `kMod.c` into a loadable kernel module.

Figure 14: A - Makefile for simple LKM

Now we can insert, list and remove the kernel modules with the commands `sudo insmod kMod.ko`, `lsmod`, `sudo rmmod kMod.ko` respectively, as shown in Figure 15.

```
[11/10/19]seed@VM:~$ sudo insmod kMod.ko
[11/10/19]seed@VM:~$ lsmod | grep kMod
kMod                   16384  0
[11/10/19]seed@VM:~$ sudo rmmod kMod.ko
[11/10/19]seed@VM:~$
```

Figure 15: A - Inserting and removing the LKM

We can observe that in the `varlogsyslog` the entries "`Initializing the module`" and "`Module cleanup`" appear, as shown in Figure 16.

```
[11/10/19]seed@VM:~$ dmesg | tail -10
              14:53:13.982082 main     Package type: LINUX_32BITS_GENERIC
[   12.673506] 14:53:13.982827 main     6.0.10 r132072 started. Verbose level = 0
[   12.674848] 14:53:13.984160 main     vbglR3GuestCtrlDetectPeekGetCancelSupport: Support
ed (#1)
[   17.689895] ISO 9660 Extensions: Microsoft Joliet Level 3
[   17.704846] ISO 9660 Extensions: RRIP_1991A
[  378.070251] ip_tables: (C) 2000-2006 Netfilter Core Team
[ 1323.585139] Initializing this module
[ 1380.469988] Module cleanup
```

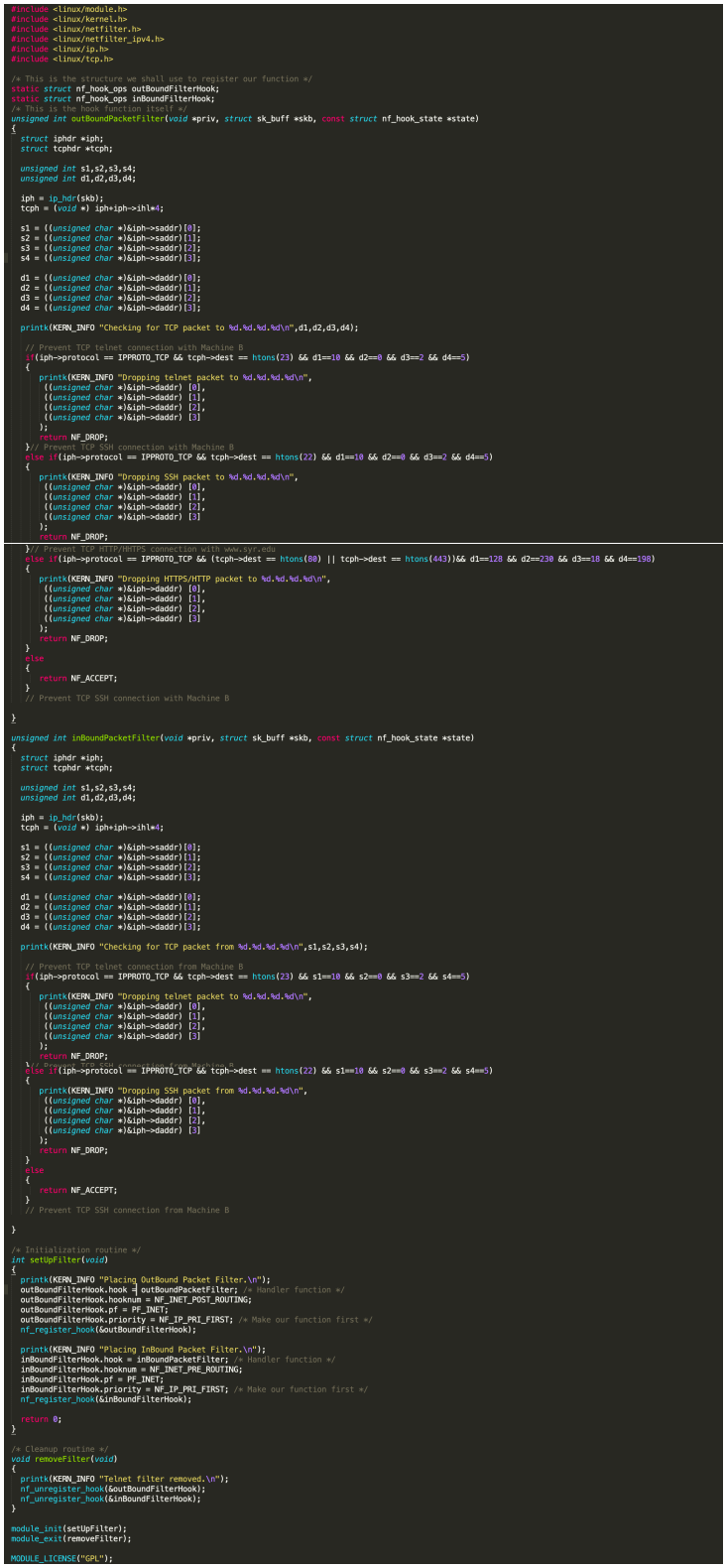
Figure 16: A - result of using the LKM

Packet filtering using Netfilter

Linux kernel offers `netfilter` for packet processing and as filtering framework. The `netfilter` provides hooks and checks if any kernel module has a callback function at these hooks. These callback function can manipulate the packet and returns a verdict whether to `ACCEPT` the packet, `DROP` the packet, `QUEUE` the packet, `STOLEN` forget the packet or `REPEAT` the function.

Netfilter Program

Figure 17 shows `netfilterFirewall.C`, it is the C program that loads 2 hooks, one hook for all out bound packets and one hook for all in bound packets. The hook that handles the inbound packets will prevent SSH and telnet connections from Machine B to Machine A. The hook that handles the outbound packets will prevent SSH and telnet connections from Machine B to Machine A, it will also prevent HTTP/HTTPS connection to an external website `www.syr.edu`.



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

/* This is the structure we shall use to register our function */
static struct nf_hook_ops outBoundFilterHook;
static struct nf_hook_ops inBoundFilterHook;
/* This is the hook function itself */
unsigned int outBoundPacketFilter(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
{
  struct iphdr *iph;
  struct tcphdr *tcph;

  unsigned int s1,s2,s3,s4;
  unsigned int d1,d2,d3,d4;

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

  s1 = ((unsigned char *)&iph->saddr)[0];
  s2 = ((unsigned char *)&iph->saddr)[1];
  s3 = ((unsigned char *)&iph->saddr)[2];
  s4 = ((unsigned char *)&iph->saddr)[3];

  d1 = ((unsigned char *)&iph->daddr)[0];
  d2 = ((unsigned char *)&iph->daddr)[1];
  d3 = ((unsigned char *)&iph->daddr)[2];
  d4 = ((unsigned char *)&iph->daddr)[3];

  printk(KERN_INFO "Checking for TCP packet to %d.%d.%d.%d\n",d1,d2,d3,d4);

   // Prevent TCP telnet connection with Machine B
   if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && d1==10 && d2==0 && d3==2 && d4==5)
   {
      printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }// Prevent TCP SSH connection with Machine B
   else if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(22) && d1==10 && d2==0 && d3==2 && d4==5)
   {
      printk(KERN_INFO "Dropping SSH packet to %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }// Prevent TCP HTTP/HTTPS connection with www.syr.edu
   else if(iph->protocol == IPPROTO_TCP && (tcph->dest == htons(80) || tcph->dest == htons(443))&& d1==128 && d2==230 && d3==10 && d4==198)
   {
      printk(KERN_INFO "Dropping HTTPS/HTTP packet to %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }
   else
   {
      return NF_ACCEPT;
   }
   // Prevent TCP SSH connection with Machine B

}

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

  unsigned int s1,s2,s3,s4;
  unsigned int d1,d2,d3,d4;

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

  s1 = ((unsigned char *)&iph->saddr)[0];
  s2 = ((unsigned char *)&iph->saddr)[1];
  s3 = ((unsigned char *)&iph->saddr)[2];
  s4 = ((unsigned char *)&iph->saddr)[3];

  d1 = ((unsigned char *)&iph->daddr)[0];
  d2 = ((unsigned char *)&iph->daddr)[1];
  d3 = ((unsigned char *)&iph->daddr)[2];
  d4 = ((unsigned char *)&iph->daddr)[3];

  printk(KERN_INFO "Checking for TCP packet from %d.%d.%d.%d\n",s1,s2,s3,s4);

   // Prevent TCP telnet connection from Machine B
   if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && s1==10 && s2==0 && s3==2 && s4==5)
   {
      printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }// Prevent TCP SSH connection from Machine B
   else if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(22) && s1==10 && s2==0 && s3==2 && s4==5)
   {
      printk(KERN_INFO "Dropping SSH packet from %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }
   else
   {
      return NF_ACCEPT;
   }
   // Prevent TCP SSH connection from Machine B

}

/* Initialization routine */
int setUpFilter(void)
{
  printk(KERN_INFO "Placing OutBound Packet Filter.\n");
  outBoundFilterHook.hook = outBoundPacketFilter; /* Handler function */
  outBoundFilterHook.hooknum = NF_INET_POST_ROUTING;
  outBoundFilterHook.pf = PF_INET;
  outBoundFilterHook.priority = NF_IP_PRI_FIRST; /* Make our function first */
  nf_register_hook(&outBoundFilterHook);

  printk(KERN_INFO "Placing InBound Packet Filter.\n");
  inBoundFilterHook.hook = inBoundPacketFilter; /* Handler function */
  inBoundFilterHook.hooknum = NF_INET_PRE_ROUTING;
  inBoundFilterHook.pf = PF_INET;
  inBoundFilterHook.priority = NF_IP_PRI_FIRST; /* Make our function first */
  nf_register_hook(&inBoundFilterHook);

  return 0;
}

/* Cleanup routine */
void removeFilter(void)
{
  printk(KERN_INFO "Telnet filter removed.\n");
  nf_unregister_hook(&outBoundFilterHook);
  nf_unregister_hook(&inBoundFilterHook);
}

module_init(setUpFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

Figure 17: A - Netfilter Program

Makefile

Figure 18 shows the `Makefile`. This file compiles the `netfilterFirewall.C` into the loadable kernel module `netfilterFirewall.ko`.

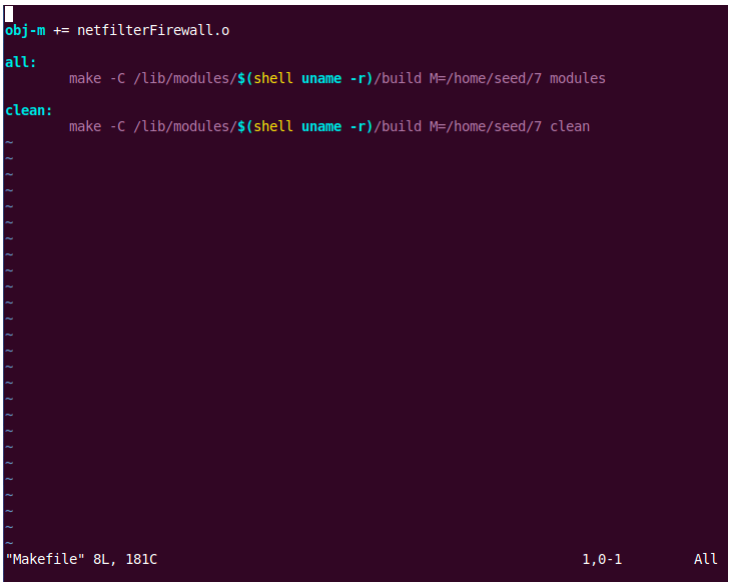


Figure 18: A - Makefile

Loading the LKM

Using command `make` the loadable kernel module is generated, as shown in Figure 19. Then using the command `sudo insmod netfilterFirewall.ko` the module is loaded, as shown in Figure 20. We can also remove the module using the command `sudo rmmod netfilterFirewall`.

```
[11/10/19]seed@VM:~/7$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/7 modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[11/10/19]seed@VM:~/7$
```

Figure 19: A - make

```
[11/10/19]seed@VM:~/7$ sudo insmod netfilterFirewall.ko
[11/10/19]seed@VM:~/7$ lsmod | grep netfilterFirewall
netfilterFirewall      16384  0
[11/10/19]seed@VM:~/7$
```

Figure 20: A - load the module

Outbound and Inbound Hooks

Figure 21 shows the setting up of the filter. The outbound hook is placed, where specifying the `hooknum`, the packet to be considered is considered. This is provided as follows `outBoundFilterHook.hooknum = NF_INET_POST_ROUTING;`, where POST_ROUTING is for outbound packets. All the packets outbound now will pass through the hook and the attached hook `outBoundPacketFilter` will do the specified packet filtering.

Similarly the inbound hook is also specified using `inBoundFilterHook.hooknum = NF_INET_PRE_ROUTING;`, where PRE_ROUTING is specified for inbound packets. This will do the specified packet filtering present in `inBoundPacketFilter` funciton. Both these hooks are registered using the function `nf_register_hook()` and unregistered with `nf_unregister_hook()`.

```
/* Initialization routine */
int setUpFilter(void)
{
  printk(KERN_INFO "Placing OutBound Packet Filter.\n");
  outBoundFilterHook.hook = outBoundPacketFilter; /* Handler function */
  outBoundFilterHook.hooknum = NF_INET_POST_ROUTING;
  outBoundFilterHook.pf = PF_INET;
  outBoundFilterHook.priority = NF_IP_PRI_FIRST; /* Make our function first */
  nf_register_hook(&outBoundFilterHook);

  printk(KERN_INFO "Placing InBound Packet Filter.\n");
  inBoundFilterHook.hook = inBoundPacketFilter; /* Handler function */
  inBoundFilterHook.hooknum = NF_INET_PRE_ROUTING;
  inBoundFilterHook.pf = PF_INET;
  inBoundFilterHook.priority = NF_IP_PRI_FIRST; /* Make our function first */
  nf_register_hook(&inBoundFilterHook);

  return 0;
}

/* Cleanup routine */
void removeFilter(void)
{
  printk(KERN_INFO "Telnet filter removed.\n");
  nf_unregister_hook(&outBoundFilterHook);
  nf_unregister_hook(&inBoundFilterHook);
}

module_init(setUpFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

Figure 21: A - Setting up the filter

1. Prevent telnet from Machine A to Machine B

For this task we will need to prevent out bound TCP packets from Machine A that is destined for Machine B's port 23.

The packets that reach the outbound hook should manipulated, we should check whether it uses TCP protocol, the destination IP address is Machine B or `10.0.2.5` and destination protocol is 23 (telnet port). The code that handles this case is shown in Figure 22.

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

  unsigned int s1,s2,s3,s4;
  unsigned int d1,d2,d3,d4;

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

  s1 = ((unsigned char *)&iph->saddr)[0];
  s2 = ((unsigned char *)&iph->saddr)[1];
  s3 = ((unsigned char *)&iph->saddr)[2];
  s4 = ((unsigned char *)&iph->saddr)[3];

  d1 = ((unsigned char *)&iph->daddr)[0];
  d2 = ((unsigned char *)&iph->daddr)[1];
  d3 = ((unsigned char *)&iph->daddr)[2];
  d4 = ((unsigned char *)&iph->daddr)[3];

  printk(KERN_INFO "Checking for TCP packet to %d.%d.%d.%d\n",d1,d2,d3,d4);

   // Prevent TCP telnet connection with Machine B
   if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && d1==10 && d2==0 && d3==2 && d4==5)
   {
      printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
        ((unsigned char *)&iph->daddr) [0],
        ((unsigned char *)&iph->daddr) [1],
        ((unsigned char *)&iph->daddr) [2],
        ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }// Prevent TCP SSH connection with Machine B
```

Figure 22: A - Outbound Telnet Filter

In the code, all out bound packets that reach the hook are filtered. The IP header of the packet is stored in the structure `iph`. We can access the TCP header using `iph` and the TCP header is stored in `tcph`. We use the IP header to verify the protocol used by the packet, `iph->protocol` must be equal to `IPPROTO_TCP` for the packet to be TCP. Also the destination address of the packet should be `10.0.2.5`, I have verified this by individually obtaining the values of the array `iph->daddr` and comparing it to 10, 0, 2 and 5 respectively and finally using `tcph->dest` is equal to `htons(23)`, I verify that the destination port of the packet is 23. If all these condition are met then the packet is DENIED else the following conditions will be verified. If the packet meets no conditions it will be accepted.

We can now verify whether the firewall rule is working. Before loading the LKM `netfilterFirewall.ko`, the telnet connection from Machine A to Machine B is possible, as shown in Figure 23.

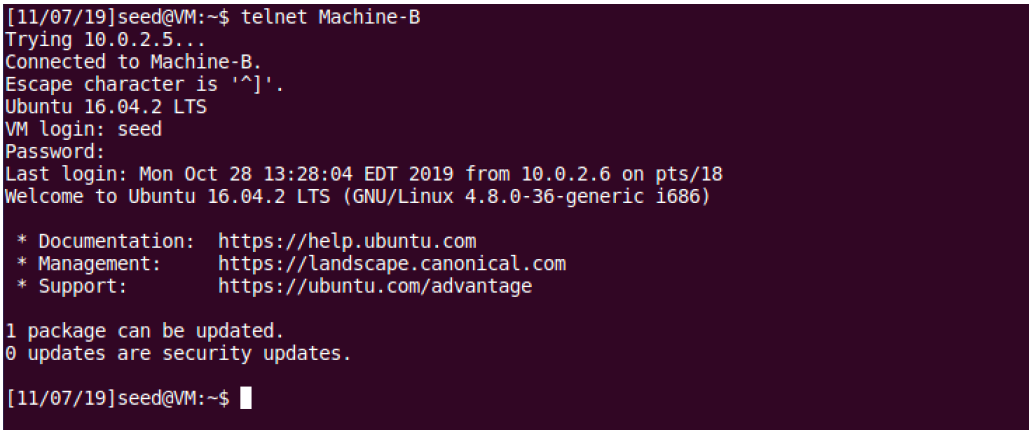


Figure 23: A - Telnet to B

Loading the module, we see that telnet connection from Machine A to Machine B is not possible, as shown in Figure 24. This can be further verified by checking `dmesg`, as shown in Figure 25.

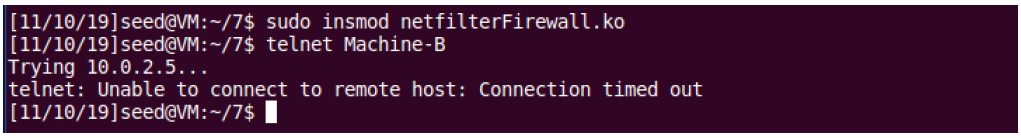


Figure 24: A - Preventing telnet to B

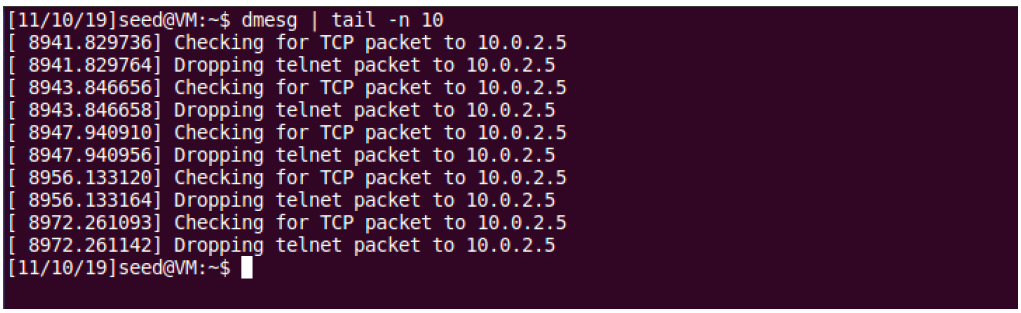


Figure 25: A - dmesg of preventing telnet to B

2. Prevent SSH from Machine A to Machine B

For this task we will need to prevent out bound TCP packets from Machine A that is destined for Machine B's port 22.

The packets that reach the outbound hook should manipulated, we should check whether it uses TCP protocol, the destination IP address is Machine B or `10.0.2.5` and destination protocol is 22 (SSH port). The code that handles this case is shown in Figure 26.

```
else if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(22) && d1==10 && d2==0 && d3==2 && d4==5)
{
   printk(KERN_INFO "Dropping SSH packet to %d.%d.%d.%d\n",
    ((unsigned char *)&iph->daddr) [0],
    ((unsigned char *)&iph->daddr) [1],
    ((unsigned char *)&iph->daddr) [2],
    ((unsigned char *)&iph->daddr) [3]
   );
   return NF_DROP;
}// Prevent TCP HTTP/HHTPS connection with www.syr.edu
```

Figure 26: A - Outbound SSH Filter

In the code, all out bound packets that reach the hook are filtered. The IP header of the packet is stored in the structure `iph`. We can access the TCP header using `iph` and the TCP header is stored in `tcph`. We use the IP header to verify the protocol used by the packet, `iph->protocol` must be equal to `IPPROTO_TCP` for the packet to be TCP. Also the destination address of the packet should be `10.0.2.5`, I have verified this by individually obtaining the values of the array `iph->daddr` and comparing it to 10, 0, 2 and 5 respectively and finally using `tcph->dest` is equal to `htons(22)`, I verify that the destination port of the packet is 22. If all these condition are met then the packet is DENIED else the following conditions will be verified. If the packet meets no conditions it will be accepted.

We can now verify whether the firewall rule is working. Before loading the LKM `netfilterFirewall.ko`, the SSH connection from Machine A to Machine B is possible, as shown in Figure 27.

```
[11/10/19]seed@VM:~$ ssh Machine-B
seed@machine-b's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

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

Last login: Sun Nov 10 19:20:00 2019 from 10.0.2.7
[11/10/19]seed@VM:~$
```

Figure 27: A - SSH to Machine B

Loading the module, we see that SSH connection from Machine A to Machine B is not possible, as shown in Figure 28.

This can be further verified by checking `dmesg`, as shown in Figure 29.

```
[11/10/19]seed@VM:~/7$ sudo insmod netfilterFirewall.ko
[11/10/19]seed@VM:~/7$ ssh Machine-B
ssh: connect to host machine-b port 22: Connection timed out
[11/10/19]seed@VM:~/7$
```

Figure 28: A - Preventing SSH to Machine B

```
[11/10/19]seed@VM:~/7$ dmesg | tail -n 10
[ 9656.836739] Checking for TCP packet to 10.0.2.5
[ 9656.836779] Dropping SSH packet to 10.0.2.5
[ 9658.852417] Checking for TCP packet to 10.0.2.5
[ 9658.852419] Dropping SSH packet to 10.0.2.5
[ 9662.948557] Checking for TCP packet to 10.0.2.5
[ 9662.948582] Dropping SSH packet to 10.0.2.5
[ 9671.168600] Checking for TCP packet to 10.0.2.5
[ 9671.168601] Dropping SSH packet to 10.0.2.5
[ 9687.268458] Checking for TCP packet to 10.0.2.5
[ 9687.268500] Dropping SSH packet to 10.0.2.5
[11/10/19]seed@VM:~/7$
```

Figure 29: A - dmesg of preventing SSH to Machine B

3. Prevent Machine A to connect to external website

For this task we will need to prevent out bound TCP packets from Machine A that is destined for an external website. I have chosen the external website as `www.syr.edu`

The packets that reach the outbound hook should manipulated, we should check whether it uses TCP protocol, the destination IP address should match the IP address of the web site and destination protocol is 80 or 443 (HTTP and HTTPS ports). The code that handles this case is shown in Figure 30.

```
}// Prevent TCP HTTP/HHTPS connection with www.syr.edu
else if(iph->protocol == IPPROTO_TCP && (tcph->dest == htons(80) || tcph->dest == htons(443))&& d1==128 && d2==230 && d3==18 && d4==198)
{
   printk(KERN_INFO "Dropping HTTPS/HTTP packet to %d.%d.%d.%d\n",
    ((unsigned char *)&iph->daddr) [0],
    ((unsigned char *)&iph->daddr) [1],
    ((unsigned char *)&iph->daddr) [2],
    ((unsigned char *)&iph->daddr) [3]
   );
   return NF_DROP;
}
else
{
   return NF_ACCEPT;
}
```

Figure 30: A - Outbound HTTP/HTTPS Filter

We initially find the IP address of the website `www.syr.edu` using the `dig` command, as shown in Figure 31. The only one IP address present in the DNS record is obtained as `128.230.18.198`.

```
[11/10/19]seed@VM:~/7$ dig www.syr.edu

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.syr.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24828
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.syr.edu.                   IN      A

;; ANSWER SECTION:
www.syr.edu.            60      IN      CNAME   syr.edu.
syr.edu.                60      IN      A       128.230.18.198

;; AUTHORITY SECTION:
syr.edu.                172695  IN      NS      ns2.syr.edu.
syr.edu.                172695  IN      NS      ns1.syr.edu.

;; ADDITIONAL SECTION:
ns1.syr.edu.            172695  IN      A       128.230.12.8
ns2.syr.edu.            172695  IN      A       128.230.12.9

;; Query time: 73 msec
;; SERVER: 10.0.2.5#53(10.0.2.5)
;; WHEN: Sun Nov 10 21:25:50 EST 2019
;; MSG SIZE  rcvd: 138
```

Figure 31: A - DNS record of www.syr.edu

In the code, all out bound packets that reach the hook are filtered. The IP header of the packet is stored in the structure `iph`. We can access the TCP header using `iph` and the TCP header is stored in `tcph`. We use the IP header to verify the protocol used by the packet, `iph->protocol` must be equal to `IPPROTO_TCP` for the packet to be TCP. Also the destination address of the packet should be `128.230.18.198`, I have verified this by individually obtaining the values of the array `iph->daddr` and comparing it to 128, 230, 18 and 198 respectively and finally using `tcph->dest` is equal to `htons(80)` or `htons(443)`, I verify that the destination port of the packet is 80 or 443. If all these condition are met then the packet is DENIED else the following conditions will be verified. If the packet meets no conditions it will be accepted.

We can now verify whether the firewall rule is working. Before loading the LKM `netfilterFirewall.ko`, we can access the website `www.syr.edu`, as shown in Figure 32.

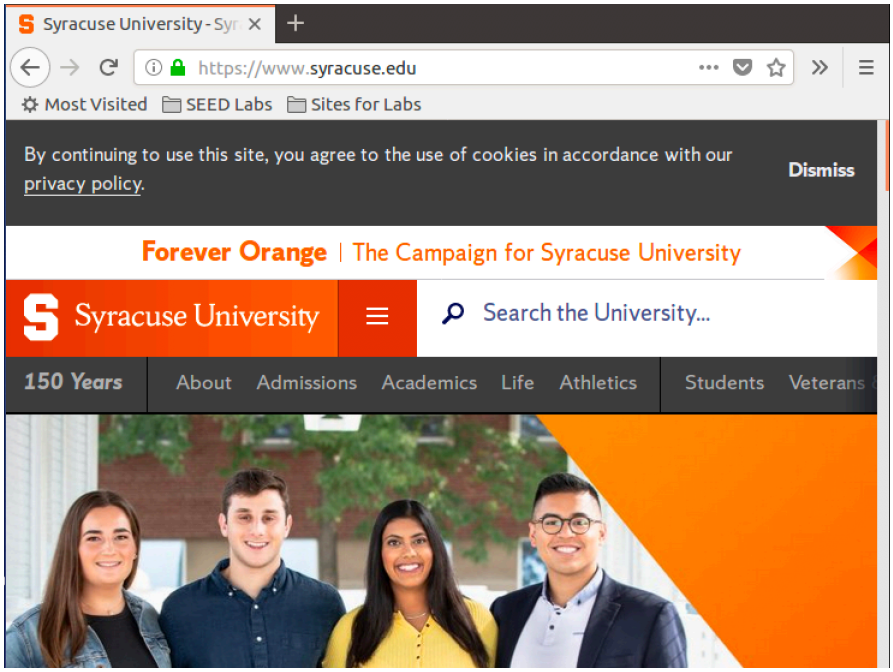


Figure 32: A - www.syr.edu

Loading the module, we observe that we cannot access the website, as shown in Figure 33. This can be further verfied by checking `dmesg`, as shown in Figure 34.

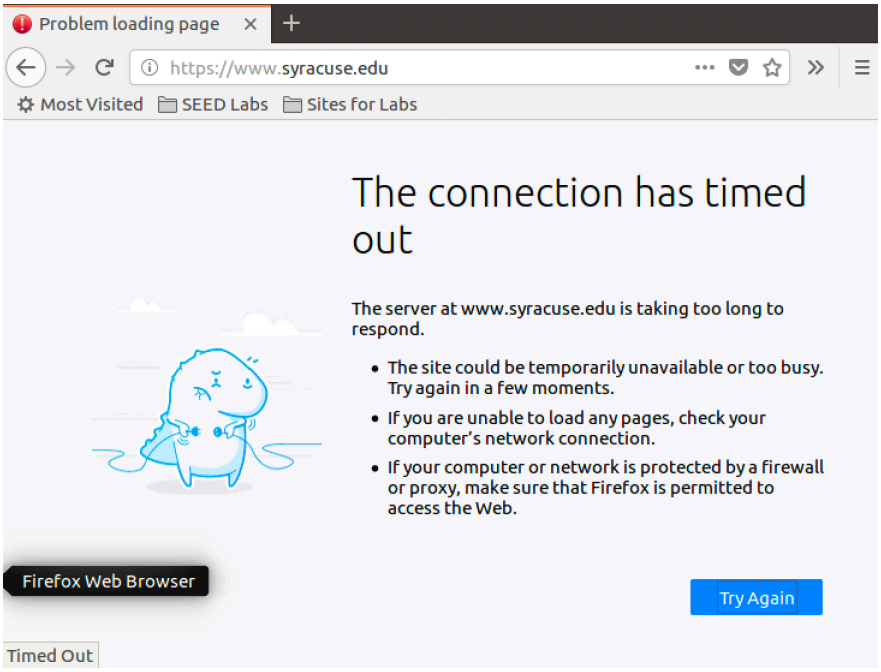


Figure 33: A - Preventing HTTP/HTTPS access to website

```
[11/10/19]seed@VM:~/7$ dmesg | tail -n 100
[ 3503.786934] Checking for TCP packet from 10.0.2.5
[ 3503.789885] Checking for TCP packet from 128.230.18.123
[ 3503.789913] Checking for TCP packet to 128.230.18.123
[ 3503.790350] Checking for TCP packet from 10.0.2.5
[ 3503.792739] Checking for TCP packet to 128.230.18.123
[ 3503.797729] Checking for TCP packet from 172.217.15.110
[ 3503.797998] Checking for TCP packet to 172.217.15.110
[ 3503.833181] Checking for TCP packet from 128.230.18.123
[ 3503.833205] Checking for TCP packet to 128.230.18.123
[ 3503.833463] Checking for TCP packet from 128.230.18.123
[ 3503.833466] Checking for TCP packet to 128.230.18.123
[ 3503.835950] Checking for TCP packet to 128.230.18.123
[ 3503.867880] Checking for TCP packet from 128.230.18.123
[ 3503.908292] Checking for TCP packet to 128.230.18.123
[ 3503.969621] Checking for TCP packet from 172.217.15.110
[ 3503.969631] Checking for TCP packet from 172.217.15.110
[ 3503.999516] Checking for TCP packet to 128.230.18.198
[ 3503.999517] Dropping HTTPS/HTTP packet to 128.230.18.198
[ 3504.772261] Checking for TCP packet to 128.230.18.198
[ 3504.772263] Dropping HTTPS/HTTP packet to 128.230.18.198
[ 3504.772265] Checking for TCP packet to 128.230.18.198
[ 3504.772266] Dropping HTTPS/HTTP packet to 128.230.18.198
[ 3505.028716] Checking for TCP packet to 128.230.18.198
[ 3505.028718] Dropping HTTPS/HTTP packet to 128.230.18.198
[ 3506.788384] Checking for TCP packet to 128.230.18.198
[ 3506.788385] Dropping HTTPS/HTTP packet to 128.230.18.198
[ 3506.788389] Checking for TCP packet to 128.230.18.198
[ 3506.788390] Dropping HTTPS/HTTP packet to 128.230.18.198
[ 3507.044279] Checking for TCP packet to 128.230.18.198
[ 3507.044280] Dropping HTTPS/HTTP packet to 128.230.18.198
[ 3509.771150] Checking for TCP packet to 128.230.18.123
[ 3509.771295] Checking for TCP packet to 128.230.18.123
[ 3509.771630] Checking for TCP packet from 128.230.18.123
[ 3509.771642] Checking for TCP packet from 128.230.18.123
```

Figure 34: A - dmesg of preventing access to website

4. Prevent telnet from Machine B to Machine A

For this task we will need to prevent in bound TCP packets to Machine A destined to port 23 from Machine B.

The packets that reach the inbound hook should manipulated, we should check whether it uses TCP protocol, the source IP address is Machine B or `10.0.2.5` and destination protocol is 23 (telnet port). The code that handles this case is shown in Figure 35.

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

  unsigned int s1,s2,s3,s4;
  unsigned int d1,d2,d3,d4;

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

  s1 = ((unsigned char *)&iph->saddr)[0];
  s2 = ((unsigned char *)&iph->saddr)[1];
  s3 = ((unsigned char *)&iph->saddr)[2];
  s4 = ((unsigned char *)&iph->saddr)[3];

  d1 = ((unsigned char *)&iph->daddr)[0];
  d2 = ((unsigned char *)&iph->daddr)[1];
  d3 = ((unsigned char *)&iph->daddr)[2];
  d4 = ((unsigned char *)&iph->daddr)[3];

  printk(KERN_INFO "Checking for TCP packet from %d.%d.%d.%d\n",s1,s2,s3,s4);

  // Prevent TCP telnet connection from Machine B
  if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && s1==10 && s2==0 && s3==2 && s4==5)
  {
     printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
      ((unsigned char *)&iph->daddr) [0],
      ((unsigned char *)&iph->daddr) [1],
      ((unsigned char *)&iph->daddr) [2],
      ((unsigned char *)&iph->daddr) [3]
     );
     return NF_DROP;
  }// Prevent TCP SSH connection from Machine B
```

Figure 35: A - Inbound Telnet Filter

In the code, all in bound packets that reach the hook are filtered. The IP header of the packet is stored in the structure `iph`. We can access the TCP header using `iph` and the TCP header is stored in `tcph`. We use the IP header to verify the protocol used by the packet, `iph->protocol` must be equal to `IPPROTO_TCP` for the packet to be TCP. Also the source address of the packet should be `10.0.2.5`, I have verified this by individually obtaining the values of the array `iph->saddr` and comparing it to 10, 0, 2 and 5 respectively and finally using `tcph->dest` is equal to `htons(23)`, I verify that the destination port of the packet is 23. If all these condition are met then the packet is DENIED else the following conditions will be verified. If the packet meets no conditions it will be accepted.

We can now verify whether the firewall rule is working. Before loading the LKM `netfilterFirewall.ko`, the telnet connection from Machine A to Machine B is possible, as shown in Figure 36.

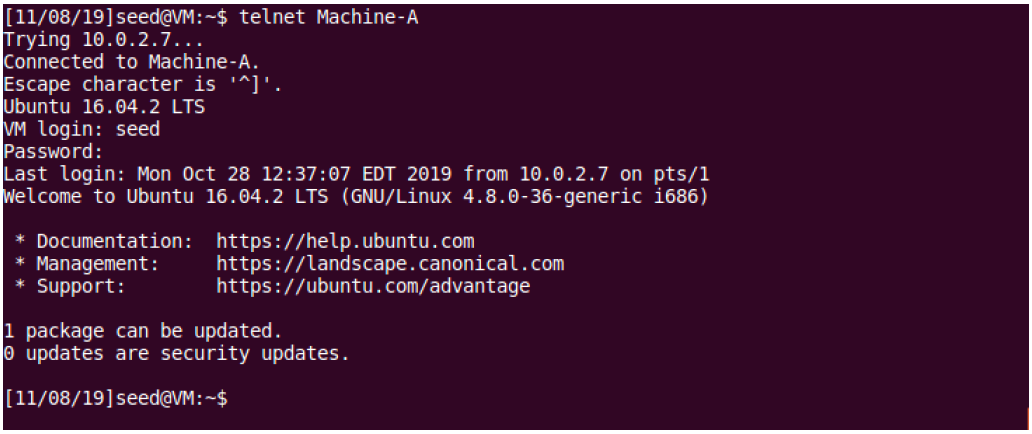


Figure 36: B - Telnet to A

Loading the module, we see that telnet connection from Machine B to Machine A is not possible, as shown in Figure 37. This can be further verified by checking `dmesg`, as shown in Figure 38.

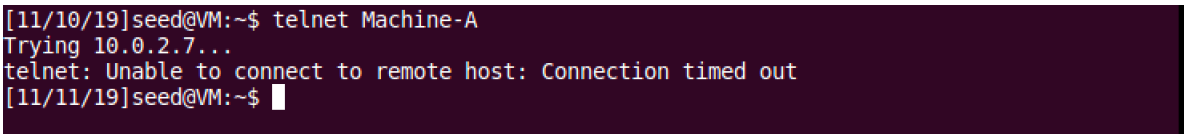


Figure 37: B - Preventing telnet to A

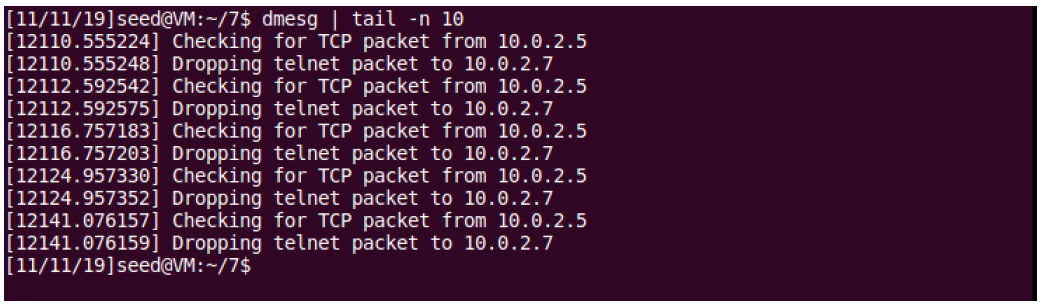


Figure 38: A - dmesg of preventing telnet from B

5. Prevent SSH from Machine B to Machine A

For this task we will need to prevent in bound TCP packets to Machine A'SSH port 22 from Machine B.

The packets that reach the inbound hook should manipulated, we should check whether it uses TCP protocol, the source IP address is Machine B or `10.0.2.5` and destination protocol is 22 (SSH port). The code that handles this case is shown in Figure 39.

```
else if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(22) && d1==10 && d2==0 && d3==2 && d4==5)
{
   printk(KERN_INFO "Dropping SSH packet to %d.%d.%d.%d\n",
    ((unsigned char *)&iph->daddr) [0],
    ((unsigned char *)&iph->daddr) [1],
    ((unsigned char *)&iph->daddr) [2],
    ((unsigned char *)&iph->daddr) [3]
   );
   return NF_DROP;
}// Prevent TCP HTTP/HHTPS connection with www.syr.edu
```

Figure 39: A - Inbound SSH Filter

In the code, all out bound packets that reach the hook are filtered. The IP header of the packet is stored in the structure `iph`. We can access the TCP header using `iph` and the TCP header is stored in `tcph`. We use the IP header to verify the protocol used by the packet, `iph->protocol` must be equal to `IPPROTO_TCP` for the packet to be TCP. Also the source address of the packet should be `10.0.2.5`, I have verified this by individually obtaining the values of the array `iph->saddr` and comparing it to 10, 0, 2 and 5 respectively and finally using `tcph->dest` is equal to `htons(22)`, I verify that the destination port of the packet is 22. If all these condition are met then the packet is DENIED else the following conditions will be verified. If the packet meets no conditions it will be accepted.

We can now verify whether the firewall rule is working. Before loading the LKM `netfilterFirewall.ko`, the SSH connection from Machine B to Machine A is possible, as shown in Figure 40.

```
[11/11/19]seed@VM:~$ ssh Machine-A
The authenticity of host 'machine-a (10.0.2.7)' 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 'machine-a,10.0.2.7' (ECDSA) to the list of known hosts.
seed@machine-a's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

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

Last login: Fri Nov  8 00:20:10 2019 from Machine-B
[11/11/19]seed@VM:~$
```

Figure 40: B - SSH to Machine A

Loading the module, we see that SSH connection from Machine B to Machine A is not possible, as shown in Figure 41.

This can be further verified by checking `dmesg`, as shown in Figure 42.

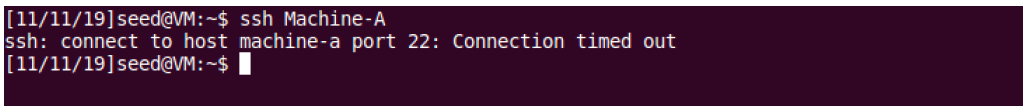


Figure 41: B - Preventing SSH to Machine A

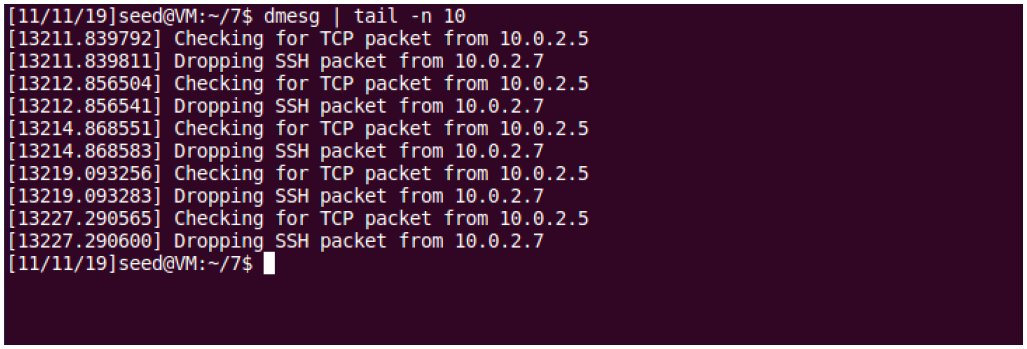


Figure 42: A - dmesg of preventing SSH from Machine B

TASK 3: Evading Egress Filtering

Initial Firewall Setup

The firewall is created using LKM and netfilter. The netfilter program is shown in Figure 43. The program essentially meets the requirements and prevents outbound telnet connections and outbound HTTP/HTTPS connections to `www.syr.edu`.

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

/* This is the structure we shall use to register our function */
static struct nf_hook_ops outBoundFilterHook;

/* This is the hook function itself */
unsigned int outBoundPacketFilter(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
{
  struct iphdr *iph;
  struct tcphdr *tcph;

  unsigned int s1,s2,s3,s4;
  unsigned int d1,d2,d3,d4;

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

  s1 = ((unsigned char *)&iph->saddr)[0];
  s2 = ((unsigned char *)&iph->saddr)[1];
  s3 = ((unsigned char *)&iph->saddr)[2];
  s4 = ((unsigned char *)&iph->saddr)[3];

  d1 = ((unsigned char *)&iph->daddr)[0];
  d2 = ((unsigned char *)&iph->daddr)[1];
  d3 = ((unsigned char *)&iph->daddr)[2];
  d4 = ((unsigned char *)&iph->daddr)[3];

  printk(KERN_INFO "Checking for TCP packet to %d.%d.%d.%d\n",d1,d2,d3,d4);

   // Prevent TCP telnet connection with Machine B
   if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(23))
   {
      printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }// Prevent TCP HTTP/HHTPS connection with www.syr.edu
   else if(iph->protocol == IPPROTO_TCP && (tcph->dest == htons(80) || tcph->dest == htons(443))&& d1==128 && d2==230 && d3==18 && d4==198)
   {
      printk(KERN_INFO "Dropping HTTPS/HTTP packet to %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }
   else
   {
      return NF_ACCEPT;
   }

}

/* Initialization routine */
int setUpFilter(void)
{
  printk(KERN_INFO "Placing OutBound Packet Filter.\n");
  outBoundFilterHook.hook = outBoundPacketFilter; /* Handler function */
  outBoundFilterHook.hooknum = NF_INET_POST_ROUTING;
  outBoundFilterHook.pf = PF_INET;
  outBoundFilterHook.priority = NF_IP_PRI_FIRST; /* Make our function first */
  nf_register_hook(&outBoundFilterHook);

  return 0;
}

/* Cleanup routine */
void removeFilter(void)
{
  printk(KERN_INFO "Telnet filter removed.\n");
  nf_unregister_hook(&outBoundFilterHook);
}

module_init(setUpFilter);
module_exit(removeFilter);
```

Figure 43: A - task3Firewall.c

The `Makefile` is updated and using the command `make` the program is compiled as a LKM and loaded into the kernel using the command `sudo insmod task3Firewall.ko`.

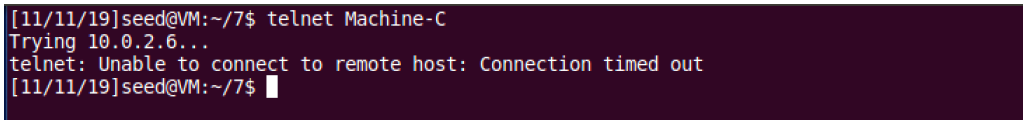
Task 3.a: Telnet to Machine B through the firewall

For this task the firewall for blocking all external telnet connection is in place. The Netfilter module, zoomed and shown in Figure 44 will filter all outbound external telnet connection.

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

  unsigned int s1,s2,s3,s4;
  unsigned int d1,d2,d3,d4;

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

  s1 = ((unsigned char *)&iph->saddr)[0];
  s2 = ((unsigned char *)&iph->saddr)[1];
  s3 = ((unsigned char *)&iph->saddr)[2];
  s4 = ((unsigned char *)&iph->saddr)[3];

  d1 = ((unsigned char *)&iph->daddr)[0];
  d2 = ((unsigned char *)&iph->daddr)[1];
  d3 = ((unsigned char *)&iph->daddr)[2];
  d4 = ((unsigned char *)&iph->daddr)[3];

  printk(KERN_INFO "Checking for TCP packet to %d.%d.%d.%d\n",d1,d2,d3,d4);

   // Prevent TCP telnet connection with Machine B
   if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(23))
   {
      printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }// Prevent TCP HTTP/HHTPS connection with www.syr.edu
```

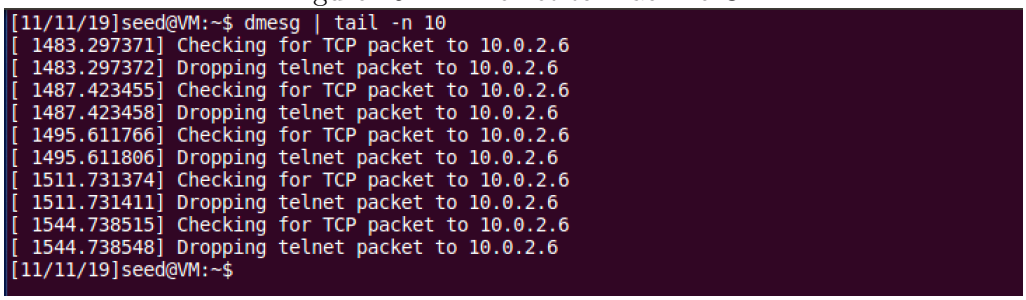
Figure 44: A - outbound telnet filter

This can be confirmed by trying to telnet to Machine C from Machine A. Figures 45 and 46 shows that the packets are dropped and the telnet connection is not possible.



```
[11/11/19]seed@VM:~/7$ telnet Machine-C
Trying 10.0.2.6...
telnet: Unable to connect to remote host: Connection timed out
[11/11/19]seed@VM:~/7$
```

Figure 45: A - Telnet to Machine C



```
[11/11/19]seed@VM:~$ dmesg | tail -n 10
[ 1483.297371] Checking for TCP packet to 10.0.2.6
[ 1483.297372] Dropping telnet packet to 10.0.2.6
[ 1487.423455] Checking for TCP packet to 10.0.2.6
[ 1487.423458] Dropping telnet packet to 10.0.2.6
[ 1495.611766] Checking for TCP packet to 10.0.2.6
[ 1495.611806] Dropping telnet packet to 10.0.2.6
[ 1511.731374] Checking for TCP packet to 10.0.2.6
[ 1511.731411] Dropping telnet packet to 10.0.2.6
[ 1544.738515] Checking for TCP packet to 10.0.2.6
[ 1544.738548] Dropping telnet packet to 10.0.2.6
[11/11/19]seed@VM:~$
```

Figure 46: A - Dropping telnet packet to Machine C

To bypass the firewall, we establish a SSH connection to Machine B, which is a system outside the firewall. We establish a SSH tunnel from port 8000 of Machine A to port 23(telnet port) of Machine C. The firewall hence will only see the SSH traffic. Since the telnet traffic is passed through the encrypted SSH traffic, the firewall will not be able to detect the telnet packets. This can be done using the `ssh` command as shown in Figure 47.

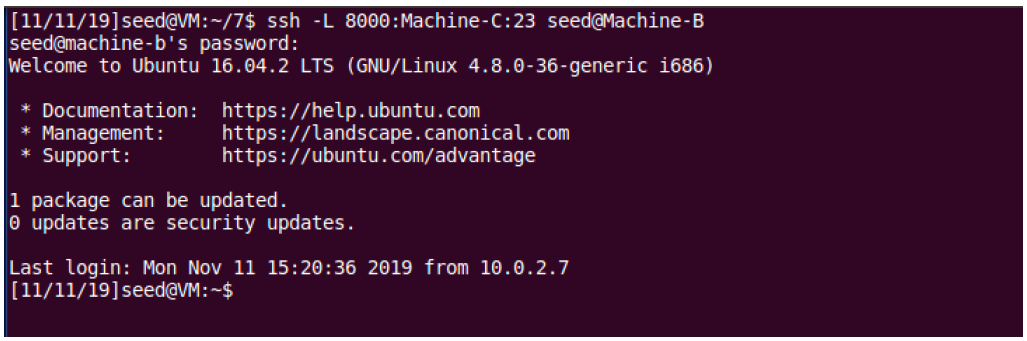


Figure 47: A - SSH to Machine and port forwarding to Machine C

Now that there is a SSH tunnel established between the port 8000 of local host and port 23 of Machine C. We can telnet to the localhost 8000 to connect to Machine C. This is shown in Figure 48.

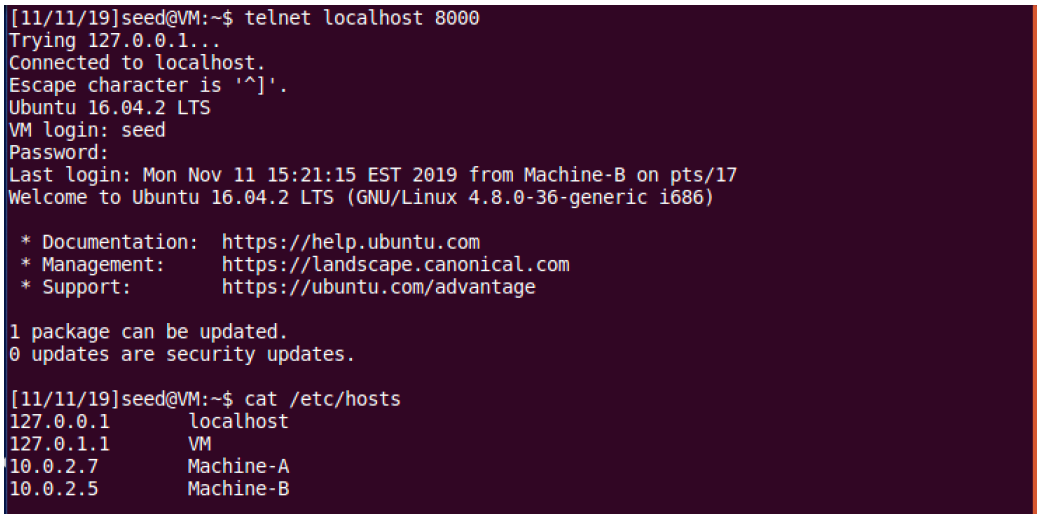


Figure 48: A - telnet to Machine C from localhost port 8000

We see that this is successful and we have connected to Machine C using telnet. This can be further confirmed using Wireshark tool. Figure 49 shows the SSH packet going from 10.0.2.7 (Machine A) to 10.0.2.5 (Machine B). We also observe that values in the SSH traffic is encrypted.

The next packet is a TELNET packet from 10.0.2.5 (Machine B) to 10.0.2.6 (Machine C). This contains the actual data passed from Machine A to Machine C. This is shown in Figure 50

The same pattern is followed for all the replies from the telnet server Machine C to the telnet client Machine A, all packets goes through Machine B through the SSh tunnel, evading detection, as shown in Figure 51.

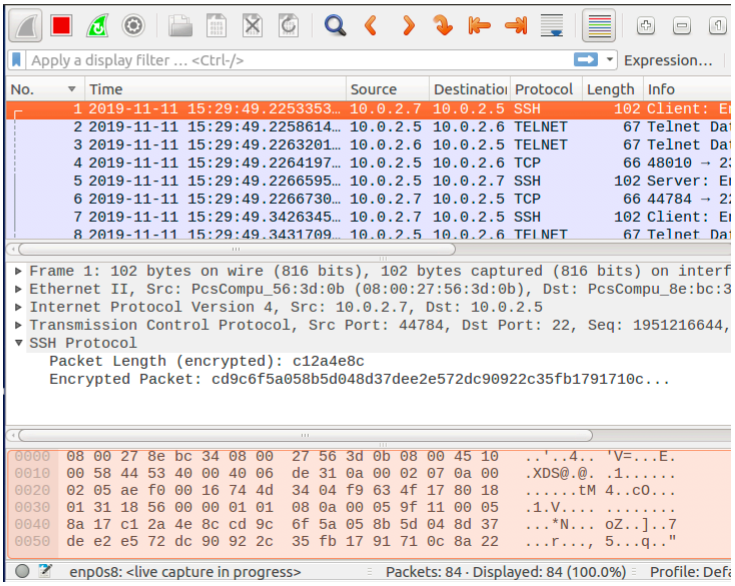


Figure 49: A - Wireshark monitoring SSH packets

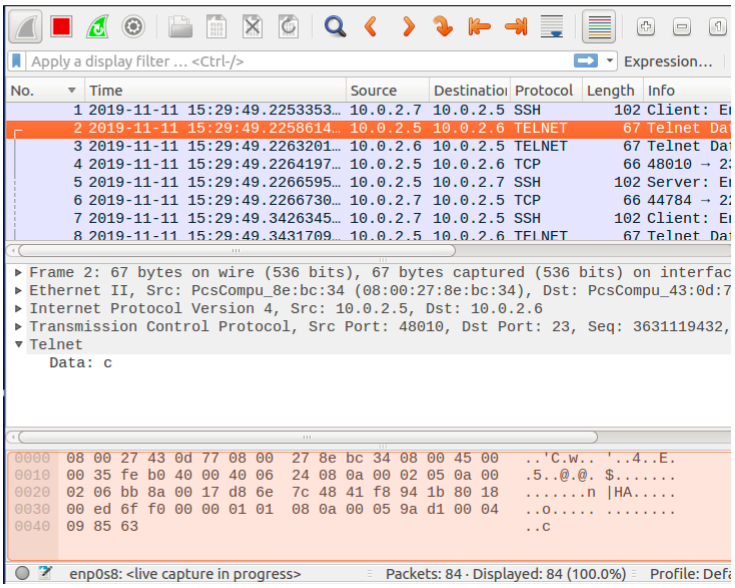


Figure 50: A - Wireshark monitoring TELNET packets

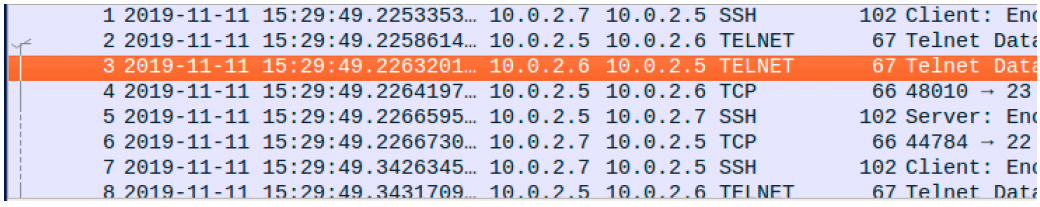


Figure 51: A - Wireshark monitoring TELNET packets

Task 3.b: Connect to Facebook using SSH Tunnel

For this task the firewall for blocking all connection to an external website like `www.facebook.com`. For demonstration purposes the website `www.syr.edu`, that uses static IP addressing, is used. By using the `dig` command we obtain the DNS record of `www.syr.edu` that contains the IP address of the website as `128.230.18.198`, and this blocked using the Netfilter module, the Figure 52 shows the zoomed view of the code related to filtering packets going to this external website.

```
}// Prevent TCP HTTP/HHTPS connection with www.syr.edu
else if(iph->protocol == IPPROTO_TCP && (tcph->dest == htons(80) || tcph->dest == htons(443))&& d1==128 && d2==230 && d3==18 && d4==198)
{
   printk(KERN_INFO "Dropping HTTPS/HTTP packet to %d.%d.%d.%d\n",
    ((unsigned char *)&iph->daddr) [0],
    ((unsigned char *)&iph->daddr) [1],
    ((unsigned char *)&iph->daddr) [2],
    ((unsigned char *)&iph->daddr) [3]
   );
   return NF_DROP;
}
else
{
   return NF_ACCEPT;
}
```

Figure 52: A - Outbound HTTP/HTTPS packet filtering

This can be confirmed by trying to access `www.syr.edu` from Machine A as shown in Figures 53 and 54 .

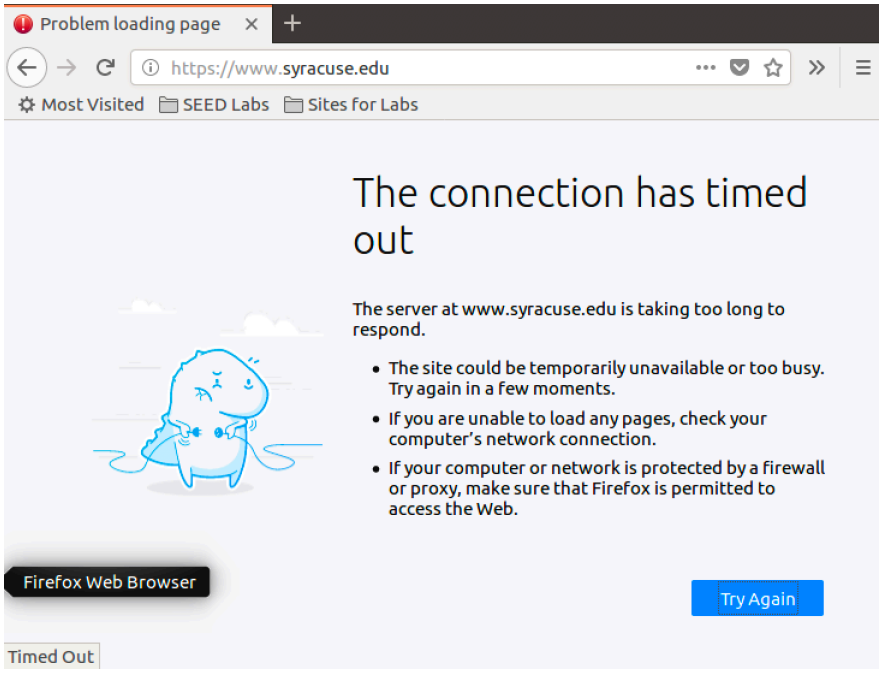


Figure 53: A - Preventing HTTP/HTTPS access to website

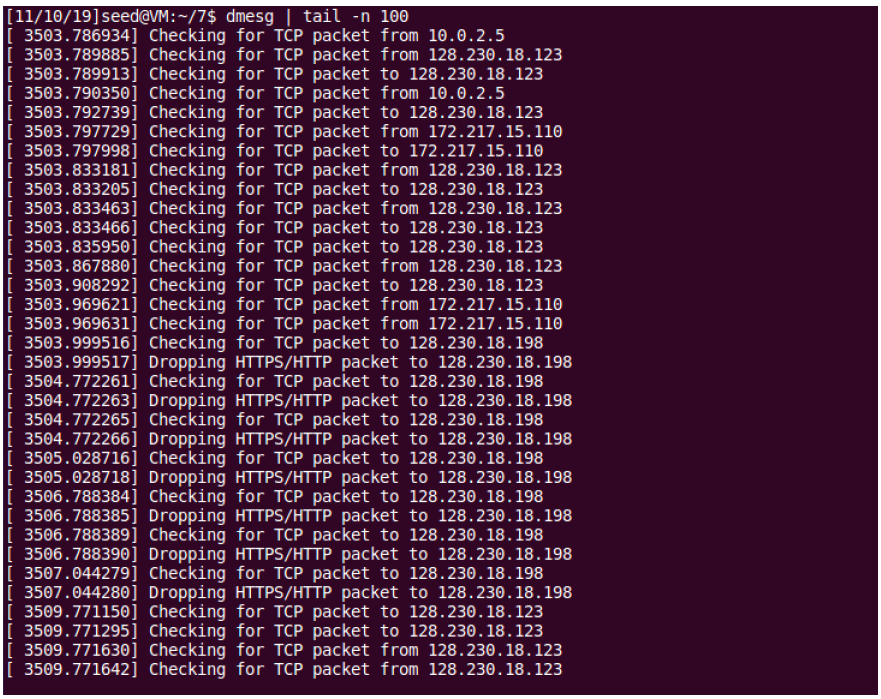


Figure 54: A - dmesg of preventing access to website

To bypass the firewall, we establish a SSH connection to Machine B, which is a system outside the firewall. We establish dynamic port forwarding with SSH tunnel from port 9000 of Machine A. The firewall hence will only see the SSH traffic. When the packet reaches Machine B, it will dynamically decide where to send the packet based on the destination information of the packet.This can be done using the `ssh` command as shown in Figure 55.

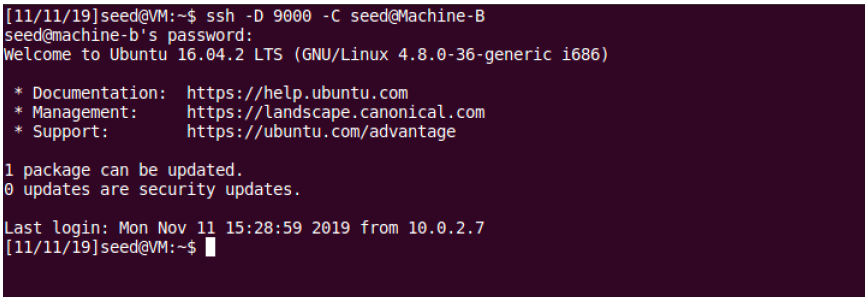


Figure 55: A - SSH dynamic port forwarding to C through B

Now we need to get Firefox to connect to the external web server through `localhost:9000` so that the traffic can go through the SSH tunnel. For this we go to preferences of Firefox, and update the Network Proxy Connection Settings. We add a manual proxy, also note since the proxy should support dynamic port forwarding, the SOCKS proxy is used. Hence the value 127.0.0.1 and 9090 were provided as the SOCKS proxy for Firefox web browser, as shown in Figure 56.

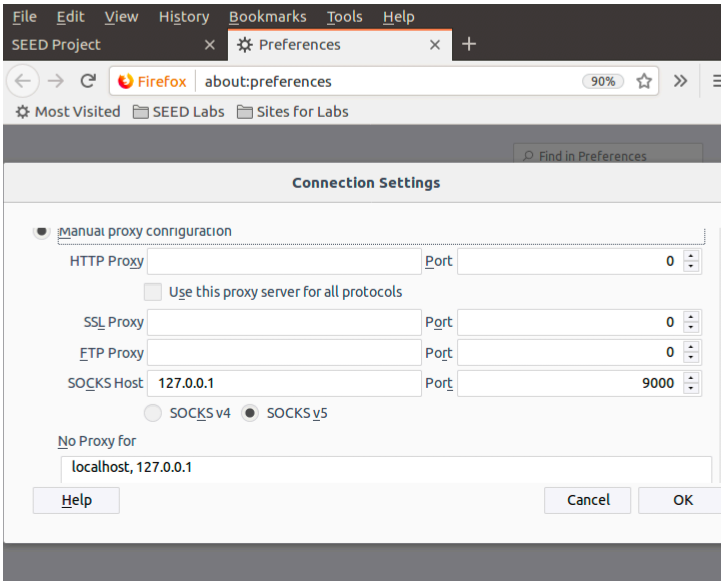


Figure 56: A - Adding Proxy to Firefox

Now visiting the website `www.syr.edu`, we see that it is accessible, as shown in Figure 57.

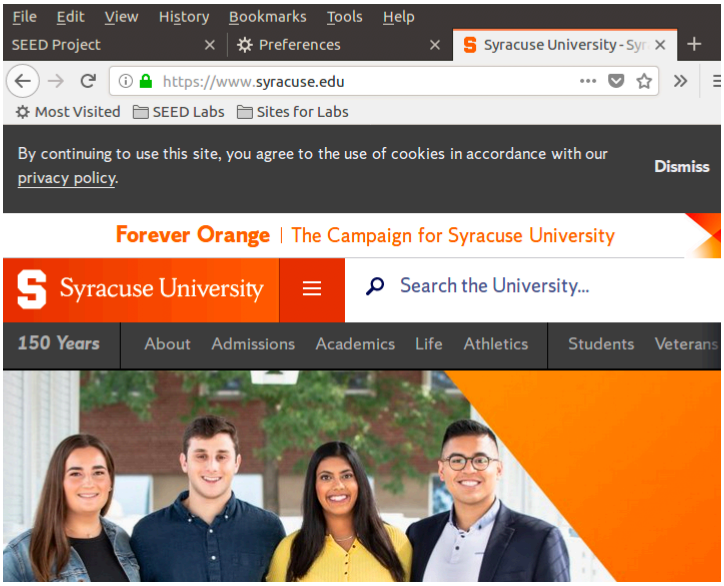


Figure 57: A - Visiting the external website through the proxy

Now we will break the SSH tunnel, as shown in Figure 58, and try the connection again. As we can see since the SSH tunnel is not present, the proxy cannot forward the packet it receives and hence the website becomes inaccessible again, as shown in Figure 59.

```
[11/11/19]seed@VM:~$ ssh -D 9000 -C seed@Machine-B
seed@machine-b's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

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

Last login: Mon Nov 11 15:28:59 2019 from 10.0.2.7
[11/11/19]seed@VM:~$ logout
```

Figure 58: A - Breaking the SSH connection

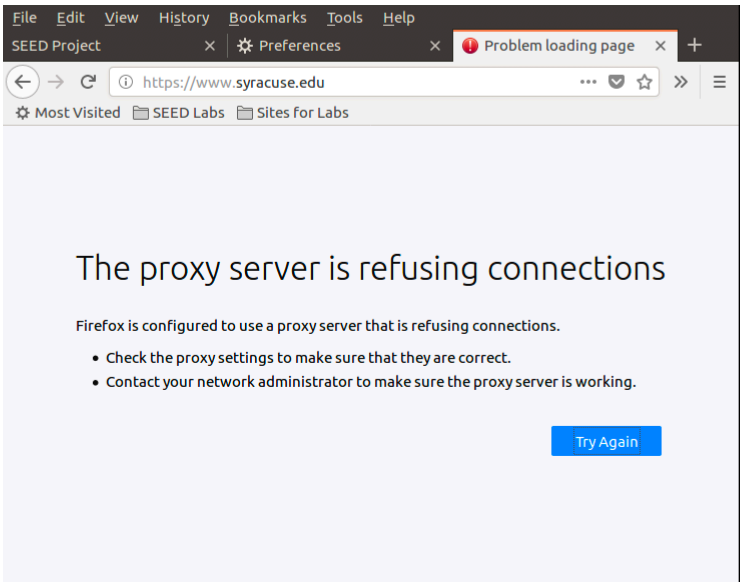


Figure 59: A - Trying to connect again

We now establish the SSH connection and try again. We see that we can access the website again. This is shown in Figures 60 and 61.

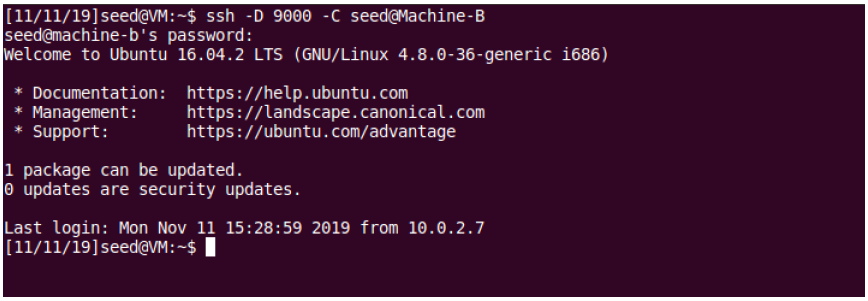


Figure 60: A - SSH dynamic port forwarding to C through B

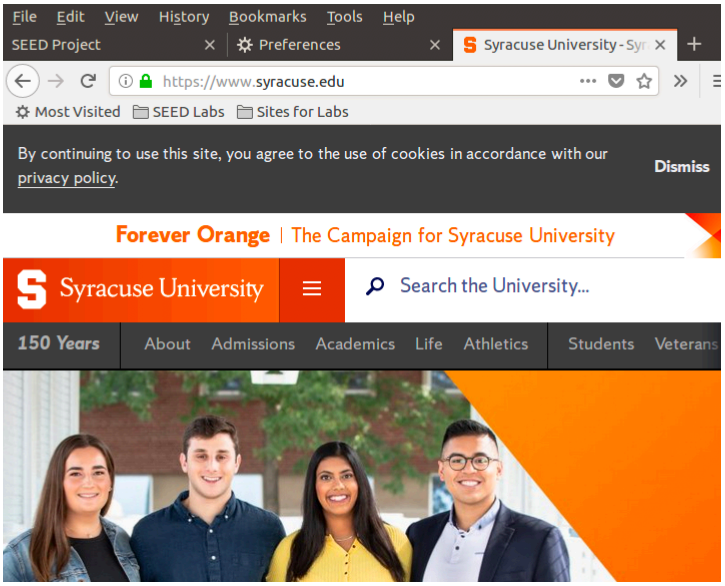


Figure 61: A - Visiting the external website through the proxy

We can confirm that the dynamic port forwarding is being used for connection by monitoring the packets using Wireshark. We see that initially a SSH packet goes from 10.0.2.7 (Machine A) to 10.0.2.5 (Machine B) which contains the encrypted packet containing the HTTPS traffic, as shown in Figure 62. This is because the proxy is active and the packets from Firefox redirects due to the proxy to Machine B. Since the SSH traffic is encrypted, the firewall doesn't drop these packet. The packets then from Machine B are forwarded to the external web server at `128.230.18.198` to the port 443 (HTTPS), as shown in Figure 63. The similar pattern is followed for the response from the web server. Because of the SSH tunnel the packets were able to evade the egress firewall, as shown in Figure 64.

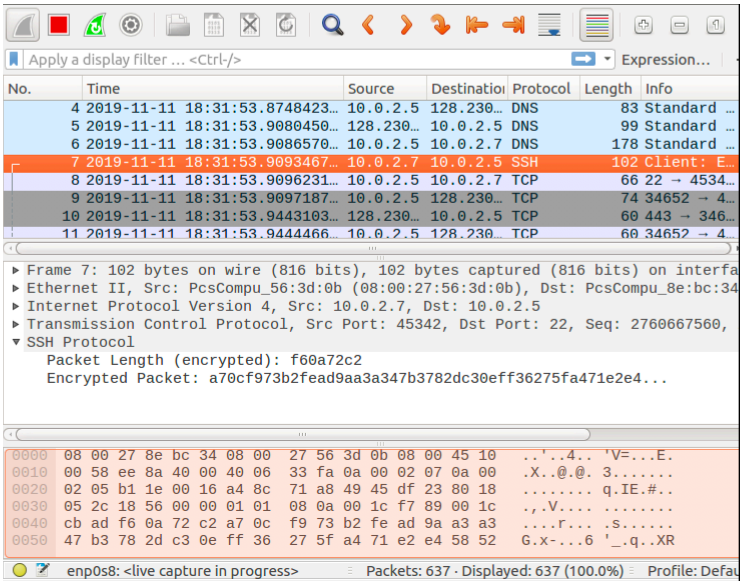


Figure 62: A - SSH traffic to B

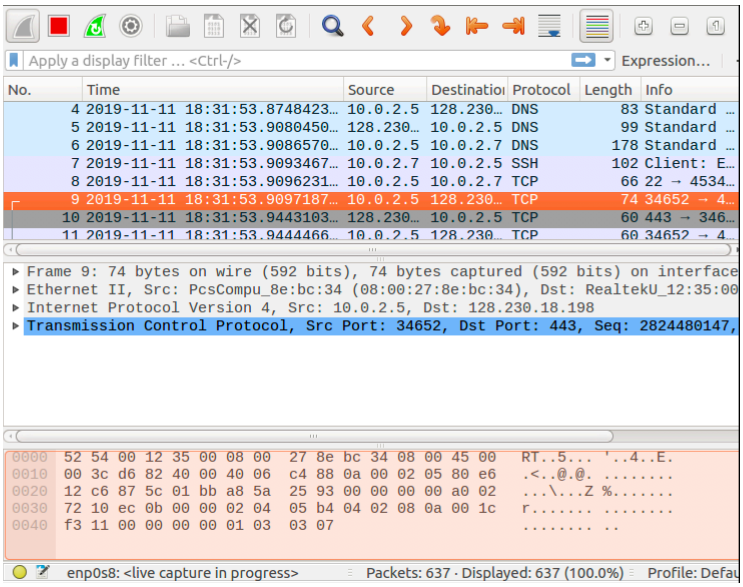


Figure 63: A - TCP packet from B to external web server

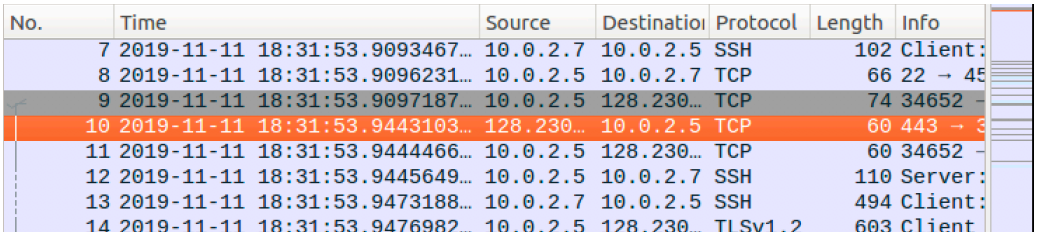


Figure 64: A - Wireshark monitoring of the packets

TASK 4: Evading Ingress Filtering

Initial Firewall Setup

The firewall is created using LKM and netfilter. The netfilter program is shown in Figure 65. The program essentially meets the requirements and prevents SSH connections and HTTP connections from Machine B or `10.0.2.5`.

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

/* This is the structure we shall use to register our function */
static struct nf_hook_ops inBoundFilterHook;


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

  unsigned int s1,s2,s3,s4;
  unsigned int d1,d2,d3,d4;

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

  s1 = ((unsigned char *)&iph->saddr)[0];
  s2 = ((unsigned char *)&iph->saddr)[1];
  s3 = ((unsigned char *)&iph->saddr)[2];
  s4 = ((unsigned char *)&iph->saddr)[3];

  d1 = ((unsigned char *)&iph->daddr)[0];
  d2 = ((unsigned char *)&iph->daddr)[1];
  d3 = ((unsigned char *)&iph->daddr)[2];
  d4 = ((unsigned char *)&iph->daddr)[3];

  printk(KERN_INFO "Checking for TCP packet from %d.%d.%d.%d\n",s1,s2,s3,s4);

   // Prevent TCP telnet connection from Machine B
   if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(22) && s1==10 && s2==0 && s3==2 && s4==5)
   {
      printk(KERN_INFO "Dropping SSH packet to %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }// Prevent TCP SSH connection from Machine B
   else if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(80) && s1==10 && s2==0 && s3==2 && s4==5)
   {
      printk(KERN_INFO "Dropping HTTP packet from %d.%d.%d.%d\n",
       ((unsigned char *)&iph->daddr) [0],
       ((unsigned char *)&iph->daddr) [1],
       ((unsigned char *)&iph->daddr) [2],
       ((unsigned char *)&iph->daddr) [3]
      );
      return NF_DROP;
   }
   else
   {
      return NF_ACCEPT;
   }
   // Prevent TCP SSH connection from Machine B

}

/* Initialization routine */
int setUpFilter(void)
{
  printk(KERN_INFO "Placing InBound Packet Filter.\n");
  inBoundFilterHook.hook = inBoundPacketFilter; /* Handler function */
  inBoundFilterHook.hooknum = NF_INET_PRE_ROUTING;
  inBoundFilterHook.pf = PF_INET;
  inBoundFilterHook.priority = NF_IP_PRI_FIRST; /* Make our function first */
  nf_register_hook(&inBoundFilterHook);

  return 0;
}

/* Cleanup routine */
void removeFilter(void)
{
  printk(KERN_INFO "Telnet filter removed.\n");
  nf_unregister_hook(&inBoundFilterHook);
}

module_init(setUpFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

Figure 65: A - task4Firewall.c

The `Makefile` is updated and using the command `make` the program is compiled as a LKM and loaded into the kernel using the command `sudo insmod task4Firewall.ko`.

For this task I have modified the `/var/www/html/index.html` in Machine A, as shown in Figure 66. This file is served by default when accessing the web server.

Figure 66: A - index.html

Before the firewall is placed, Machine B can access the web server in Machine A. He will get the default page set as shown in Figure 67.

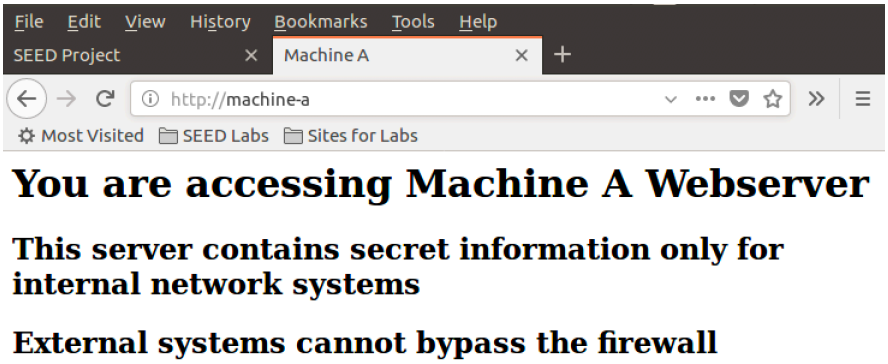


Figure 67: B - Accessing http:Machine-A before firewall

Now the LKM compiled is loaded and the firewall is placed. The code for the module blocks external HTTP and SSH connections, the zoomed picture is shown in Figure 68.

```
printk(KERN_INFO "Checking for TCP packet from %d.%d.%d.%d\n",s1,s2,s3,s4);

// Prevent TCP telnet connection from Machine B
if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(22) && s1==10 && s2==0 && s3==2 && s4==5)
{
   printk(KERN_INFO "Dropping SSH packet from %d.%d.%d.%d\n",
    ((unsigned char *)&iph->saddr) [0],
    ((unsigned char *)&iph->saddr) [1],
    ((unsigned char *)&iph->saddr) [2],
    ((unsigned char *)&iph->saddr) [3]
   );
   return NF_DROP;
}// Prevent TCP SSH connection from Machine B
else if(iph->protocol == IPPROTO_TCP && tcph->dest == htons(80) && s1==10 && s2==0 && s3==2 && s4==5)
{
   printk(KERN_INFO "Dropping HTTP packet from %d.%d.%d.%d\n",
    ((unsigned char *)&iph->saddr) [0],
    ((unsigned char *)&iph->saddr) [1],
    ((unsigned char *)&iph->saddr) [2],
    ((unsigned char *)&iph->saddr) [3]
   );
   return NF_DROP;
}
else
{
   return NF_ACCEPT;
}
// Prevent TCP SSH connection from Machine B
```

Figure 68: B - Firewall code to prevent SSH and HTTP

Hence now Machine B cannot access the web server in Machine A as shown in Figures 69 and 70

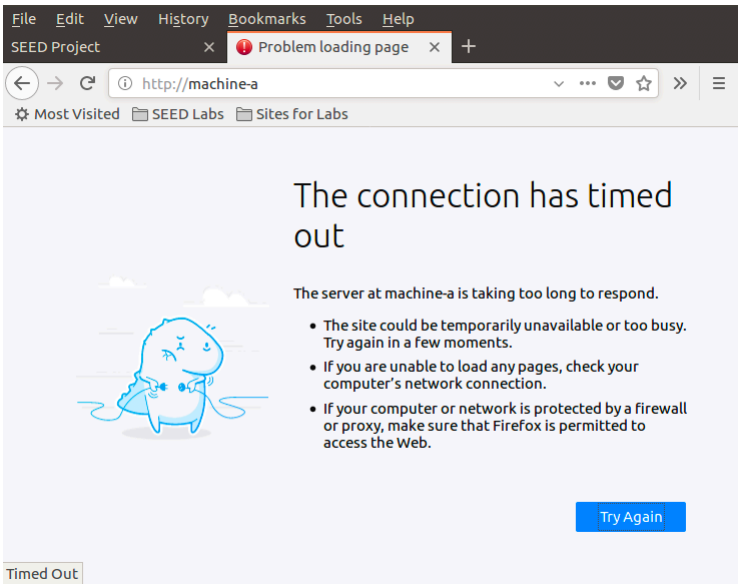


Figure 69: B - Firewall preventing access to web server in Machine A

```
[11/11/19]seed@VM:~/7$ dmesg | tail
[13987.120333] Checking for TCP packet from 10.0.2.5
[13987.120364] Dropping HTTP packet from 10.0.2.5
[13991.056561] Checking for TCP packet from 10.0.2.5
[13991.056588] Dropping HTTP packet from 10.0.2.5
[13991.312646] Checking for TCP packet from 10.0.2.5
[13991.312690] Dropping HTTP packet from 10.0.2.5
[13999.248479] Checking for TCP packet from 10.0.2.5
[13999.248510] Dropping HTTP packet from 10.0.2.5
[13999.504122] Checking for TCP packet from 10.0.2.5
[13999.504170] Dropping HTTP packet from 10.0.2.5
[11/11/19]seed@VM:~/7$
```

Figure 70: A - dmesg of Dropping packets to Machine B

Since the firewall blocks the inbound HTTP and SSH connections. Machine A will have to setup a reverse SSH tunnel. I have made my reverse SSH tunnel port forwarding from 24553 port of Machine B to port 80 of Machine A. So that Machine B can access the web server in Machine A. This is shown in Figure 71.

```
[11/11/19]seed@VM:~/7$ ssh -R 24553:localhost:80 seed@Machine-B
seed@machine-b's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

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

Last login: Mon Nov 11 19:58:40 2019 from 10.0.2.7
[11/11/19]seed@VM:~$
```

Figure 71: A - Reverse SSH port forwarding

Now the user at Machine B can access the web server at Machine A by going to localhost at port 24553. We can confirm this by going to the required IP through port 24553 as shown in Figure 72. We can also this by monitoring the packets in Wireshark. The TCP packets go through as SSH packets as shown in Figure 73.

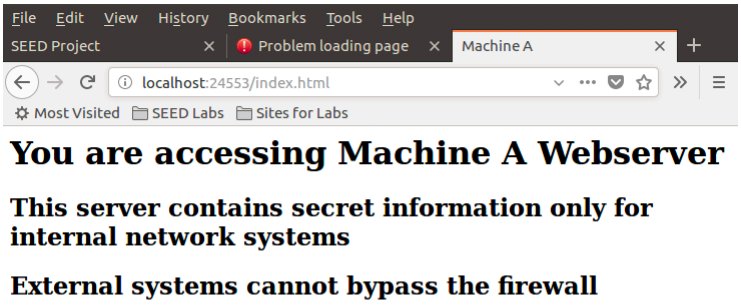


Figure 72: B - Accessing Machine A's web server

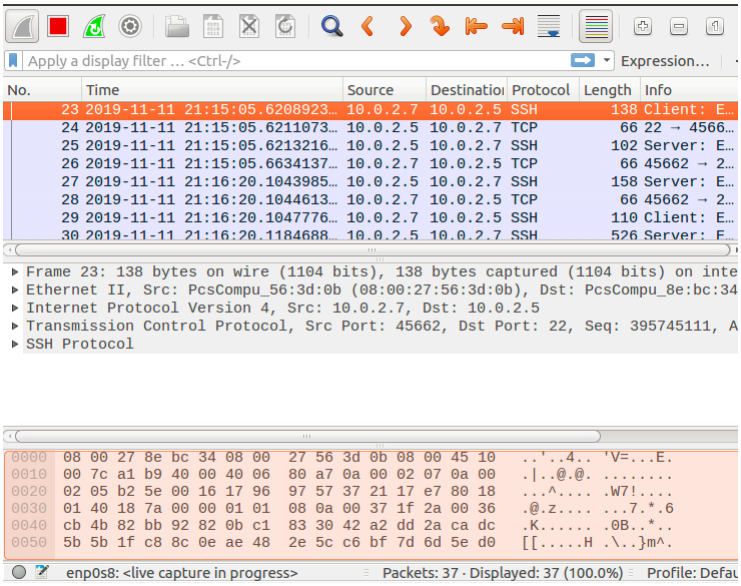


Figure 73: A - Wireshark monitoring the packets