SETUP)

Machine A : 10.0.2.15

Machine B: 10.0.2.4

Task 1) Using Firewall

All of these are run on VM A (10.0.2.15). We use `sudo ufw disable` command to stop reload the firewall and disable firewall on boot. And we also use `sudo iptables -F` command to flush all iptables rules.

```
[10/26/21]seed@VM:~$ sudo ufw disable
Firewall stopped and disabled on system startup
[10/26/21]seed@VM:~$ sudo iptables -F
```

Prevent A from telnetting to B.
We ran this command below on the Machine A(10.0.2.15). The -d 10.0.2.4 means the destination IP address is Machine B(10.0.2.4) and –dport 23 means the port is 23 since the telnet's port is 23. This screen continued trying to connect until it was eventually dropped.

```
[10/26/21]seed@VM:~$ sudo iptables -A OUTPUT -s 10.0.2.15/32 -d 10.0.2.4/32 -o eth1 -p tcp
m tcp --dport 23 -j DROP
[10/26/21]seed@VM:~$ telnet 10.0.2.4
Trying 10.0.2.4...
```

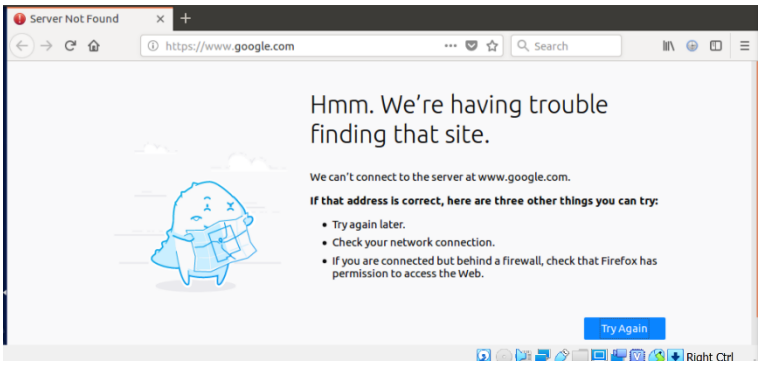
Prevent B from telnetting A.
We ran this first one on the Machine A(10.0.2.15). The -s 10.0.2.4 means the source IP address is Machine B(10.0.2.4) and –dport 23 means the port is 23 since the telnet's port is 23.
Then we tried to telnet into machine A from Machine B and did not work.

```
[10/26/21]seed@VM:~$ sudo iptables -A INPUT -s 10.0.2.4/32 -d 10.0.2.15/32 -p tcp -m tcp --d
port 23 -j DROP
```

```
→  ~ telnet 10.0.2.15
Trying 10.0.2.15...
```

Prevent A from visiting outside web page. We block any traffic to 8.8.8.8 on port 80 (http) and 443 (https) from machine A.

```
[10/26/21]seed@VM:~$ sudo iptables -A OUTPUT -s 10.0.2.15/32 -d 8.8.8.8/32 -p tcp -m tcp --d
port 80 -j DROP
[10/26/21]seed@VM:~$ sudo iptables -A OUTPUT -s 10.0.2.15/32 -d 8.8.8.8/32 -p tcp -m tcp --d
port 443 -j DROP
```



And we check to make sure our VM can't get to google. Our firewall worked.

Task 2) Implementing a Simple Firewall

We write this simple firewall program that implements 5 different rules with LKM and Netfilter.

Rule 1: Prevent A from telnetting to B.

We will prevent A from doing telnet to Machine B like in task 1 by using Netfilter tool. Our goal is to block the packets that are going out to port 23, preventing Machine A(10.0.2.15) from using telnet to connect to Machine B(10.0.2.4).

```
// rule for task 1.1: Prevent A from doing `telnet` to Machine B
unsigned int telnetFilter_1(void *priv, struct sk_buff *skb,
                            const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

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

    if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && eq_daddr(iph, "10.0.2.4") && eq_saddr(iph,
"10.0.2.15"))
    {
        printk(KERN_INFO "Dropping telnet from %pI4 packet to %pI4\n", &iph->saddr, &iph->daddr);
        return NF_DROP;
    }
    else
    {
        return NF_ACCEPT;
    }
}
```

Rule 2: Prevent B from telnetting to A.

We will prevent B from doing telnet to Machine A like in task 1 by using Netfilter tool. Our goal is to block the packets that are going out to port 23, preventing Machine B(10.0.2.4) from using telnet to connect to Machine A(10.0.2.15).

```c
// Prevent B from telnet A
unsigned int telnetFilter_2(void *priv, struct sk_buff *skb,
                            const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

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

    if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && eq_daddr(iph, "10.0.2.15") && eq_saddr(iph,
"10.0.2.4"))
    {
        printk(KERN_INFO "Dropping telnet from %pI4 packet to %pI4\n", &iph->saddr, &iph->daddr);
        return NF_DROP;
    }
    else
    {
        return NF_ACCEPT;
    }
}
```

Rule 3: Don't allow A to visit 8.8.8.8.

We will prevent A from visiting an external web site like in task 1 by using Netfilter tool. Our goal is to block the packets that are going out to port 80(http) and port 443(https), preventing Machine A(10.0.2.15) from visiting google DNS(8.8.8.8). So we can't visit www.google.com.

```c
//disallow A from visiting 8.8.8.8
unsigned int block_google(void *priv, struct sk_buff *skb,
                          const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;
    iph = ip_hdr(skb);
    tcph = (void *)iph + iph->ihl * 4;
    if ((tcph->dest == htons(80) || tcph->dest == htons(443))
    && (eq_daddr(iph, "8.8.8.8") && eq_saddr(iph, "10.0.2.15"))
    {
        printk(KERN_INFO "Dropping http/https from %pI4 packet to %pI4\n", &iph->saddr, &iph->daddr);
        return NF_DROP;
    }
    else
    {
        return NF_ACCEPT;
    }
}
```

Rule 4: Don't allow A to visit facebook.com (157.240.229.35)

We will prevent A from visiting an external web site www.facebook.com using Netfilter tool. Our goal is to block the packets that are going out to port 80(http) and port 443(https), preventing Machine A(10.0.2.15) from visiting www.facebook.com(157.240.229.35).

```c
//disallow A from visiting facebook.com
unsigned int block_face(void *priv, struct sk_buff *skb,
                        const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;
    iph = ip_hdr(skb);
    tcph = (void *)iph + iph->ihl * 4;
    if ((tcph->dest == htons(80) || tcph->dest == htons(443))
    && (eq_daddr(iph, "157.240.229.35") && eq_saddr(iph, "10.0.2.15"))
    {
        printk(KERN_INFO "Dropping http/https from %pI4 packet to %pI4\n", &iph->saddr, &iph->daddr);
        return NF_DROP;
    }
    else
    {
        return NF_ACCEPT;
    }
}
```

Rule 5: Don't allow A or B to visit Linkedin (13.107.42.14)

We will prevent A and B from visiting an external web site www.linkedin.com using Netfilter tool. Our goal is to block the packets that are going out to port 80(http) and port 443(https), preventing Machine A(10.0.2.15) and Machine B(10.0.2.4)from visiting www.linkedin.com (13.107.42.14).

```c
//disallow A from visiting linkedin
unsigned int block_link(void *priv, struct sk_buff *skb,
                        const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;
    iph = ip_hdr(skb);
    tcph = (void *)iph + iph->ihl * 4;
    if ((tcph->dest == htons(80) || tcph->dest == htons(443))
    && (eq_daddr(iph, "13.107.42.14") && (eq_saddr(iph, "10.0.2.15") || eq_saddr(iph,"10.0.2.4"))
    {
        printk(KERN_INFO "Dropping http/https from %pI4 packet to %pI4\n", &iph->saddr, &iph->daddr);
        return NF_DROP;
    }
    else
    {
        return NF_ACCEPT;
    }
}
```

All of these make this program.

We can see the code below that we have add 5 filter rules mentioned above. It then uses `nf_register_hook()` to do the final registration.

The macros `module_init ()` and `module_exit ()` point to functions that are to be executed while the kernel module is being inserted and removed from the kernel respectively.

```
int setUpFilter(void)
{
    int i;
    printk(KERN_INFO "Registering filters.\n");
    FilterHookRule[0] = (struct nf_hook_ops){.hook = telnetFilter_1, .hooknum = NF_INET_LOCAL_OUT, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};
    FilterHookRule[1] = (struct nf_hook_ops){.hook = telnetFilter_2, .hooknum = NF_INET_LOCAL_IN, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};
    FilterHookRule[2] = (struct nf_hook_ops){.hook = block_google, .hooknum = NF_INET_LOCAL_OUT, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};
    FilterHookRule[3] = (struct nf_hook_ops){.hook = block_face, .hooknum = NF_INET_LOCAL_OUT, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};
    FilterHookRule[4] = (struct nf_hook_ops){.hook = block_link, .hooknum = NF_INET_LOCAL_OUT, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};

    // set the amount of filter rules
    regist_num = 5;

    for (i = 0; i < regist_num; i++)
        nf_register_hook(&FilterHookRule[i]);
    return 0;
}

void removeFilter(void)
{
    int i;
    printk(KERN_INFO "Filters are being removed.\n");
    //unregist hooks one by one
    for (i = 0; i < regist_num; i++)
        nf_unregister_hook(&FilterHookRule[i]);
    regist_num = 0;
}

module_init(setUpFilter);
module_exit(removeFilter);

MODULE LICENSE("GPL");
```

To compile a module, we had to make a makefile that was pretty complicated. We used https://tldp.org/LDP/lkmpg/2.6/html/x181.html for help on this.

In this makefile, we compile the `simple_firewall.c` file, and finally generate the `simple_firewall.ko` file.

`obj-m` means to compile and generate a loadable module.

`-C` specifies the location of the kernel source code.

`M=$(PWD)`: The source file address of the module to be compiled.

`all, clean` are pseudo-targets in makefiles.

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

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

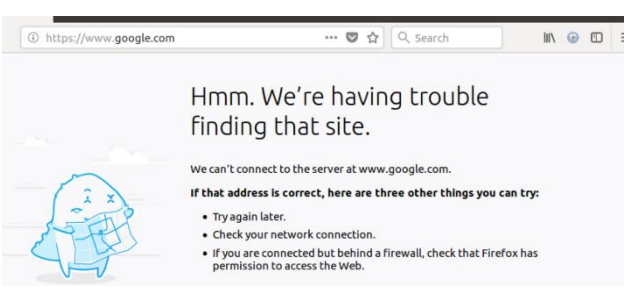
Then we “make” in the command line and get the module to run by issuing a module command to install.

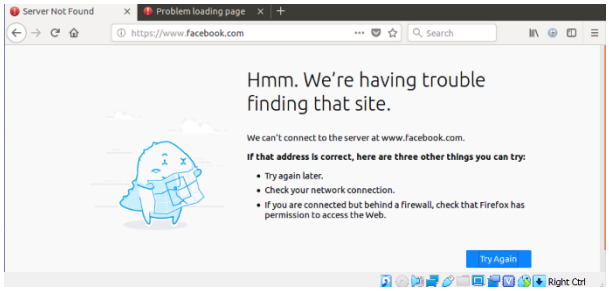
```
[10/26/21]seed@VM:~/Lab6$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Lab6 modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/seed/Lab6/simple_firewall.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Lab6/simple_firewall.mod.o
  LD [M]  /home/seed/Lab6/simple_firewall.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[10/26/21]seed@VM:~/Lab6$ sudo insmod simple_firewall.ko
[10/26/21]seed@VM:~/Lab6$
```

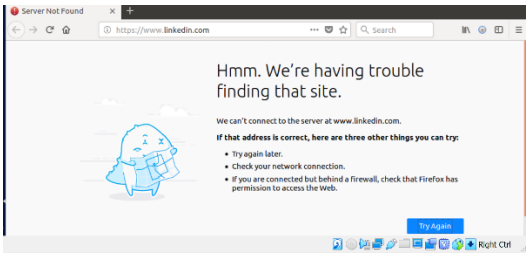
We check all of our rules. **In order**, you can see that telnetting 10.0.2.15 to 10.0.2.4 doesn't work. telnetting 10.0.2.4 to 10.0.2.15 doesn't work. 10.0.2.15 cannot visit google. 10.0.2.15 cannot visit facebook. 10.0.2.15 NOR 10.0.2.4 can visit Linkedin. Success!

```
[10/26/21]seed@VM:~/Lab6$ telnet 10.0.2.4
Trying 10.0.2.4...
```

```
→  ~ telnet 10.0.2.15
Trying 10.0.2.15...
```

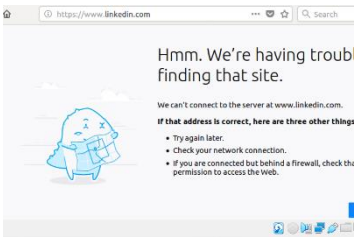






<- Machine A

Machine B ->



Task 3) Evading Egress Filtering

We disable our firewall from task 2.

```
[10/26/21]seed@VM:~/Lab6$ sudo rmmod simple_firewall
```

We rewrite the rules from task 2 to block all outgoing traffic to any telnet servers.

Our goal is to block all the packets that are going out to port 23. After adding this filter rule, the Machine A is blocked behind a firewall that rejected all telnet to 10.0.2.15.

```
unsigned int telnetFilter_1(void *priv, struct sk_buff *skb,
                            const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

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

    if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23))
    {
        printk(KERN_INFO "Dropping telnet from %pI4 packet to %pI4\n", &iph->saddr, &iph->daddr);
        return NF_DROP;
    }
    else
    {
        return NF_ACCEPT;
    }
}
```

We write a rule that disallows all visiting to facebook.

Our goal is to block the packets that are going out to port 80(http) and port 443(https), preventing Machine A(10.0.2.15) from visiting www.facebook.com(157.240.229.35). After adding this filter rule, the Machine A should not be able to see Facebook pages.

```
//DISALLOW ALL FACEBOOK.COM VISITATIONS

unsigned int block_face(void *priv, struct sk_buff *skb,
                        const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;
    iph = ip_hdr(skb);
    tcph = (void *)iph + iph->ihl * 4;
    if ((tcph->dest == htons(80) || tcph->dest == htons(443))
    && eq_daddr(iph, "157.240.229.35"))
    {
        printk(KERN_INFO "Dropping http/https from %pI4 packet to %pI4\n", &iph->saddr, &iph->daddr);
        return NF_DROP;
    }
    else
    {
        return NF_ACCEPT;
    }
}
```

Then we adjust our SetupFilter() function below to only include those two rules.

So our firewall has been set up already, which will block all the outgoing traffic to both the telnet servers and the Facebook pages.

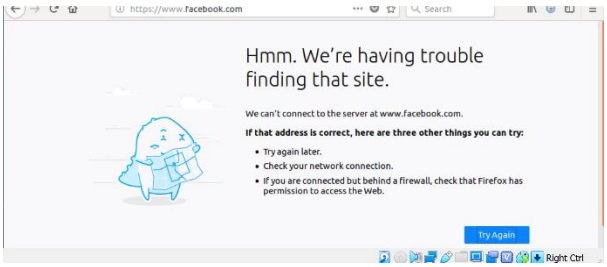
```
int setUpFilter(void)
{
    int i;
    printk(KERN_INFO "Registering filters.\n");
    FilterHookRule[0] = (struct nf_hook_ops){.hook = telnetFilter_1, .hooknum = NF_INET_LOCAL_OUT, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};
    //FilterHookRule[1] = (struct nf_hook_ops){.hook = telnetFilter_2, .hooknum = NF_INET_LOCAL_IN, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};
    //FilterHookRule[2] = (struct nf_hook_ops){.hook = block_google, .hooknum = NF_INET_LOCAL_OUT, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};
    FilterHookRule[1] = (struct nf_hook_ops){.hook = block_face, .hooknum = NF_INET_LOCAL_OUT, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};
    //FilterHookRule[4] = (struct nf_hook_ops){.hook = block_link, .hooknum = NF_INET_LOCAL_OUT, .pf =
PF_INET, .priority = NF_IP_PRI_FIRST};

    // set the amount of filter rules
    regist_num = 2;

    for (i = 0; i < regist_num; i++)
        nf_register_hook(&FilterHookRule[i]);
    return 0;
}
```

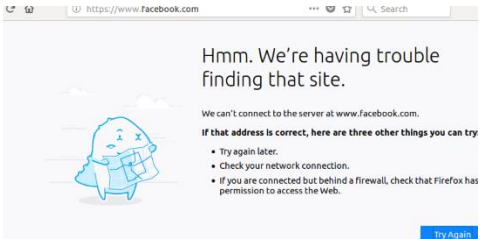
We make and install our module.

```
[10/26/21]seed@VM:~/Lab6$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Lab6 modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/seed/Lab6/simple_firewall.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Lab6/simple_firewall.mod.o
  LD [M]  /home/seed/Lab6/simple_firewall.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[10/26/21]seed@VM:~/Lab6$ sudo insmod simple_firewall.ko
```

Then we test to make sure. In order, you can see that it worked. A cannot telnet to B and neither B NOR A can get to facebook.com.

```
[10/26/21]seed@VM:~/Lab6$ telnet 10.0.2.4
Trying 10.0.2.4...
```





Task 3a) Machine C is 10.0.2.5

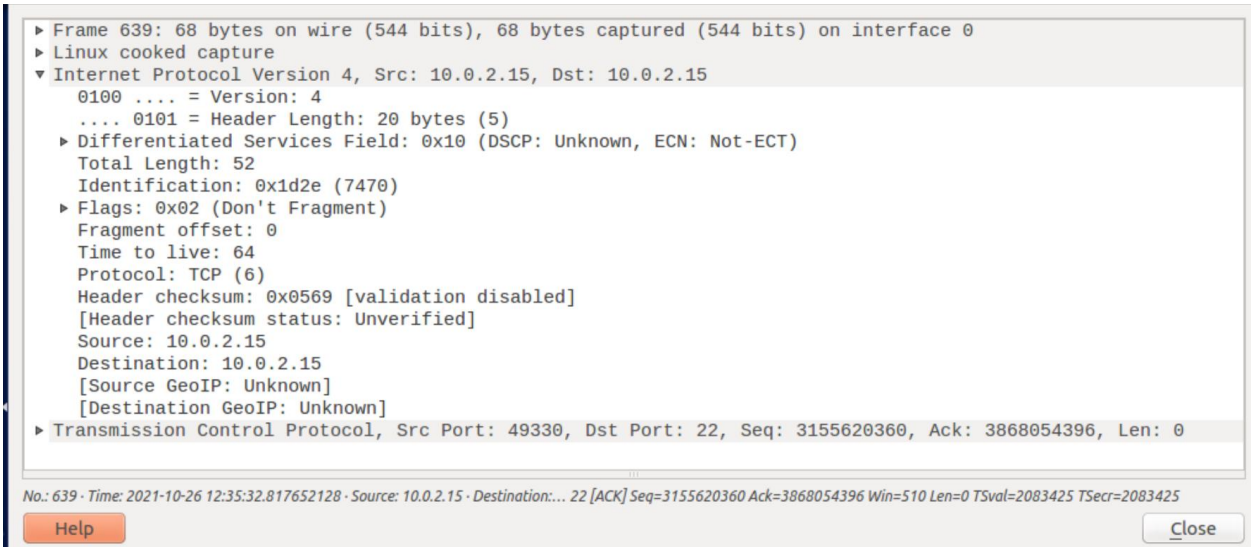
We bypass the firewall by setting up an ssh tunnel between A and B through port 23.

```
[10/26/21]seed@VM:~/Lab6$ ssh -L 8000:10.0.2.4:23 seed@10.0.2.15
The authenticity of host '10.0.2.15 (10.0.2.15)' can't be established.
ECDSA key fingerprint is SHA256:p1zAio6c1bI+8HDp5xa+eKRi561aFDaPE1/xq1eYzCI.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added '10.0.2.15' (ECDSA) to the list of known hosts.
seed@10.0.2.15'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
```

And now we cann telnet using the tunnel on localhost 8000

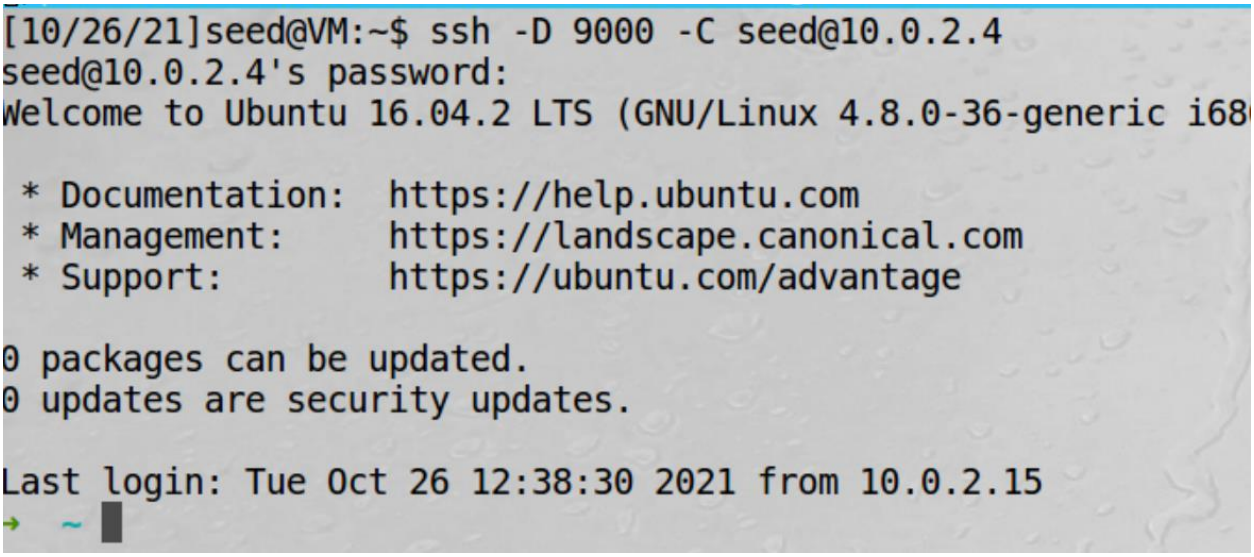
```
[10/26/21]seed@VM:~$ telnet localhost 8000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

The wireshark packet is below. Basically, we are putting ssh tunnel through B's local host 8000 that connects our machine A to machine B's local host 8000. Then, localhost will forward the packets from machine B to machine C's telnet server. By doing this, Machine A is not actually the one using the telnet server as appearing to the firewall. We are using a true middle man to do the connection.

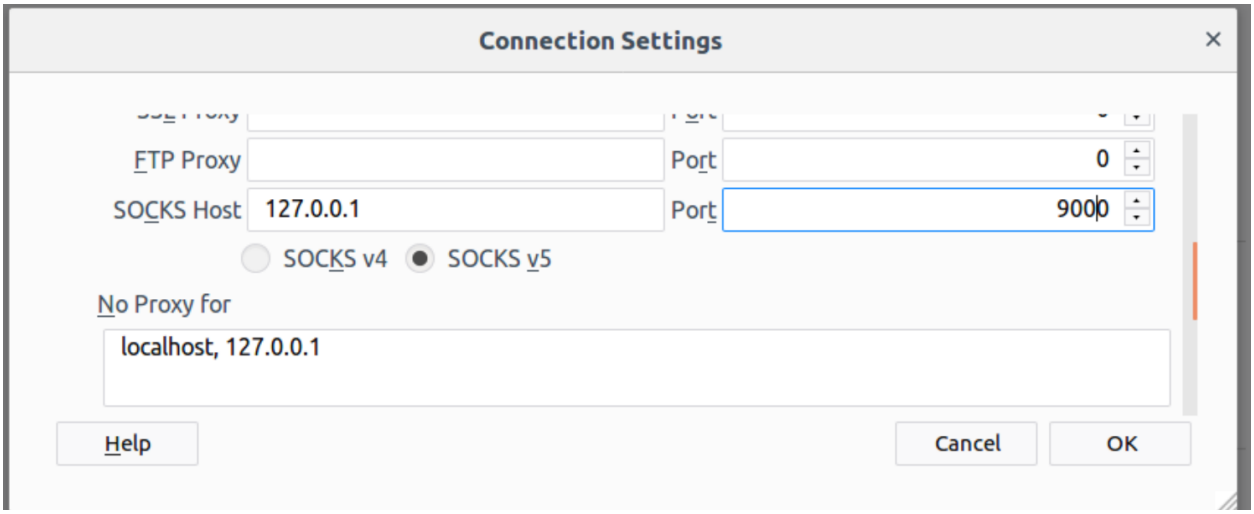


Task 3b)

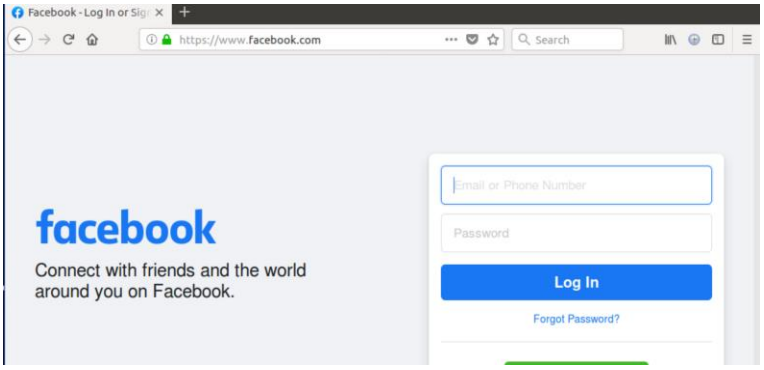
We tunnel the ssh connection.



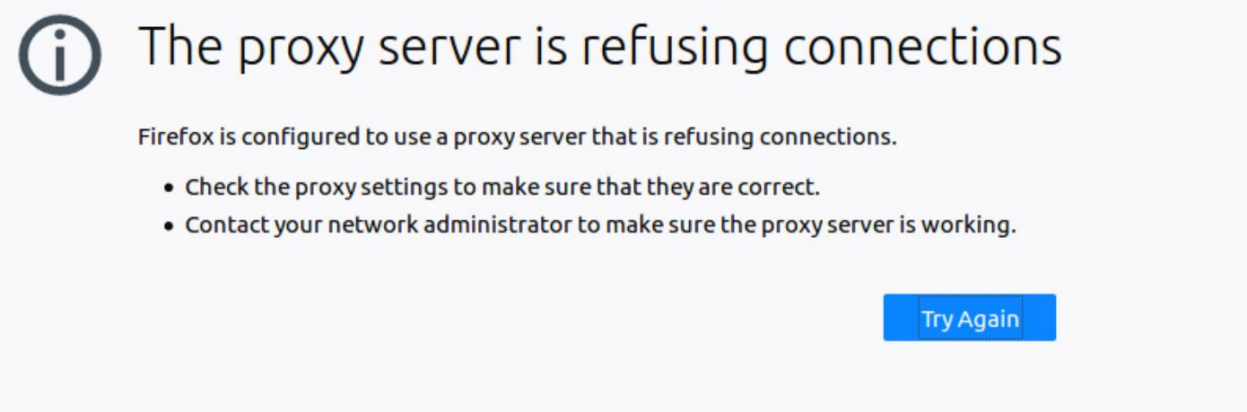
We set the preferences in firefox



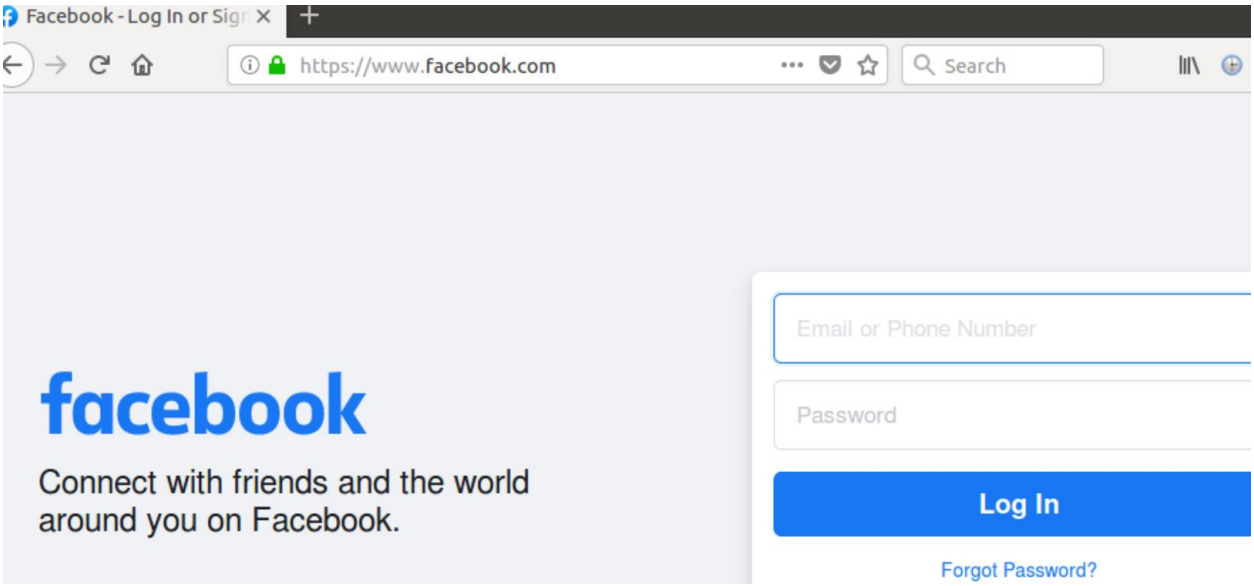
We visit facebook and it works.



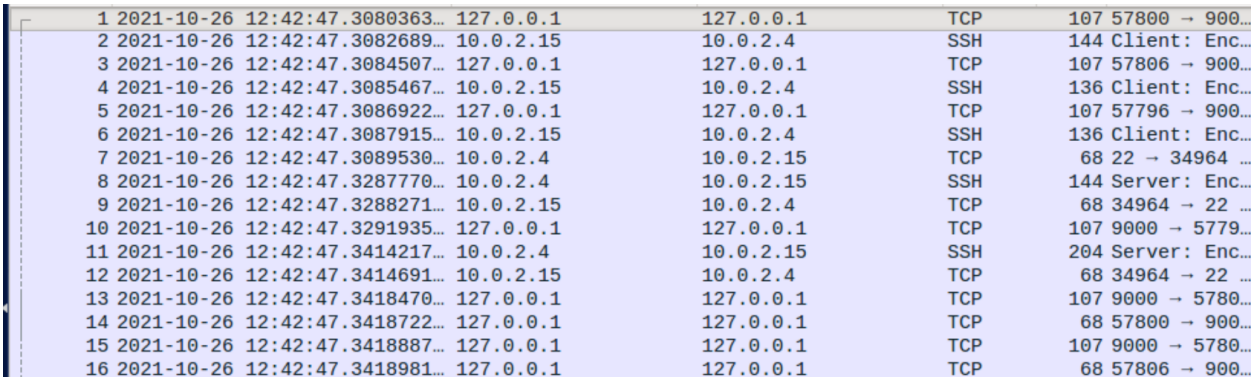
Then we break the tunnel and get this error message.



Then we reestablish the ssh tunnel and it works!



Here are the wireshark packets when the ssh tunnel is working. As you can see, the 10.0.2.15 is connecting to 127.0.0.1 over and over. 10.0.2.4 and 10.0.2.15 are connecting, as well through the ssh tunnel. So basically, firefox requires the local proxy to go through our ssh tunnel every time it needs to connect to a web server. Which allows us to bypass the filters.



1	2021-10-26 12:42:47.3080363…	127.0.0.1	127.0.0.1	TCP	107	57800 → 900…
2	2021-10-26 12:42:47.3082689…	10.0.2.15	10.0.2.4	SSH	144	Client: Enc…
3	2021-10-26 12:42:47.3084507…	127.0.0.1	127.0.0.1	TCP	107	57806 → 900…
4	2021-10-26 12:42:47.3085467…	10.0.2.15	10.0.2.4	SSH	136	Client: Enc…
5	2021-10-26 12:42:47.3086922…	127.0.0.1	127.0.0.1	TCP	107	57796 → 900…
6	2021-10-26 12:42:47.3087915…	10.0.2.15	10.0.2.4	SSH	136	Client: Enc…
7	2021-10-26 12:42:47.3089530…	10.0.2.4	10.0.2.15	TCP	68	22 → 34964 …
8	2021-10-26 12:42:47.3287770…	10.0.2.4	10.0.2.15	SSH	144	Server: Enc…
9	2021-10-26 12:42:47.3288271…	10.0.2.15	10.0.2.4	TCP	68	34964 → 22 …
10	2021-10-26 12:42:47.3291935…	127.0.0.1	127.0.0.1	TCP	107	9000 → 5779…
11	2021-10-26 12:42:47.3414217…	10.0.2.4	10.0.2.15	SSH	204	Server: Enc…
12	2021-10-26 12:42:47.3414691…	10.0.2.15	10.0.2.4	TCP	68	34964 → 22 …
13	2021-10-26 12:42:47.3418470…	127.0.0.1	127.0.0.1	TCP	107	9000 → 5780…
14	2021-10-26 12:42:47.3418722…	127.0.0.1	127.0.0.1	TCP	68	57800 → 900…
15	2021-10-26 12:42:47.3418887…	127.0.0.1	127.0.0.1	TCP	107	9000 → 5780…
16	2021-10-26 12:42:47.3418981…	127.0.0.1	127.0.0.1	TCP	68	57806 → 900…

Task 4)

Block all incoming ssh connections using iptables.

```
[10/26/21]seed@VM:~$ sudo iptables -A INPUT -s 10.0.2.4/32 -d 10.0.2.15/32 -p tc
p -m tcp --dport 22 -j DROP
```

We create a reverse ssh tunnel on machine A that connects to machine B. It goes through 8000 on local host and uses port 80 (http). This is outbound traffic so it is not blocked.

```
[10/26/21]seed@VM:~$ ssh -R 8000:localhost:80 seed@10.0.2.4
seed@10.0.2.4'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

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

Last login: Tue Oct 26 12:41:46 2021 from 10.0.2.15
→  ~
```

Now we can access the localhost:8000 on machine B's firefox. It works!!

It means we have already set up a reverse SSH tunnel on Machine A successfully. Machine B can still access the protected web server on A outside.