Lab 5

Firewall Evasion Lab: Bypassing Firewalls using VPN

Name：程昊

Student Number：57117128

Task1: Using Firewall

◆ 实验流程：

这里将 192.168.114.129 设置为 A，192.168.114.130 设置为 B。

修改默认的设置：

```
# Set the default input policy to ACCEPT, DROP, or REJECT. Please note that if
# you change this you will most likely want to adjust your rules.
DEFAULT_INPUT_POLICY="ACCEPT"
```

1. Prenvent A from doing telnet B：

首先检查此时 A 是可以连通 B 的 telnet 的：

```
[09/07/20]seed@VM:~/.../lab5$ telnet 192.168.114.130
Trying 192.168.114.130...
Connected to 192.168.114.130.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login:
```

输入以下命令：

```
[09/07/20]seed@VM:~$ sudo ufw deny out from 192.168.114.129 to 192.168.114.130 p
ort 23
Rules updated
```

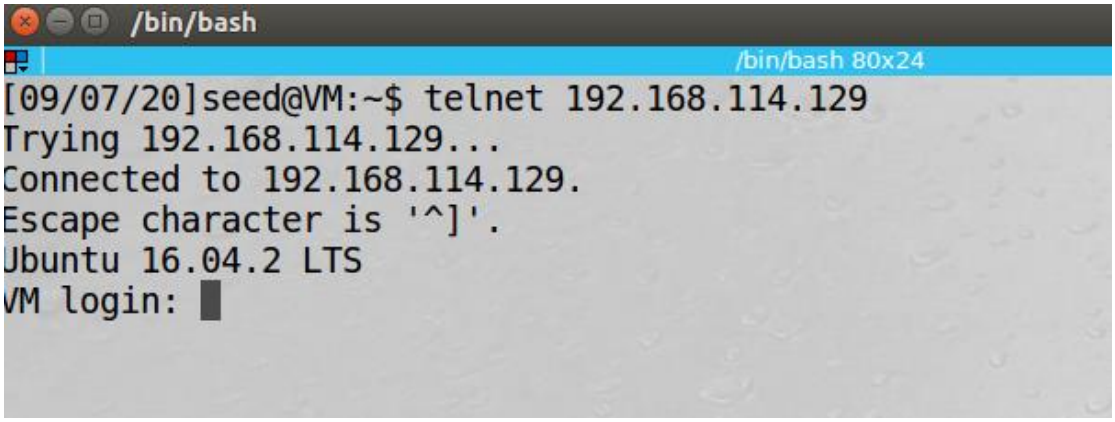
然后再次尝试连接到 B：

```
[09/07/20]seed@VM:~$ telnet 192.168.114.130
Trying 192.168.114.130...
```

此时终端正在不断尝试发起向 B 机器的 telnet 连接。

2. Prevent B from doing A telnet：

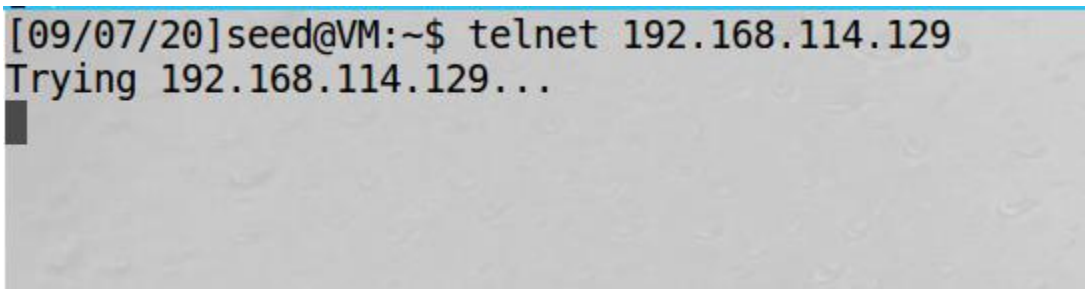
首先检查在设置防火墙规则前是否能从 B 连接到 A 的 telnet 服务器：



在 A 上添加如下防火墙规则：

```
[09/07/20]seed@VM:~$ sudo ufw deny from 192.168.114.130 to 192.168.114.129 port
23
Rule added
```

此时再尝试从 B 连接 A 的 telnet 服务器：



B 一直尝试向 A 发起 telnet 连接，不过包全被 ufw 过滤了。

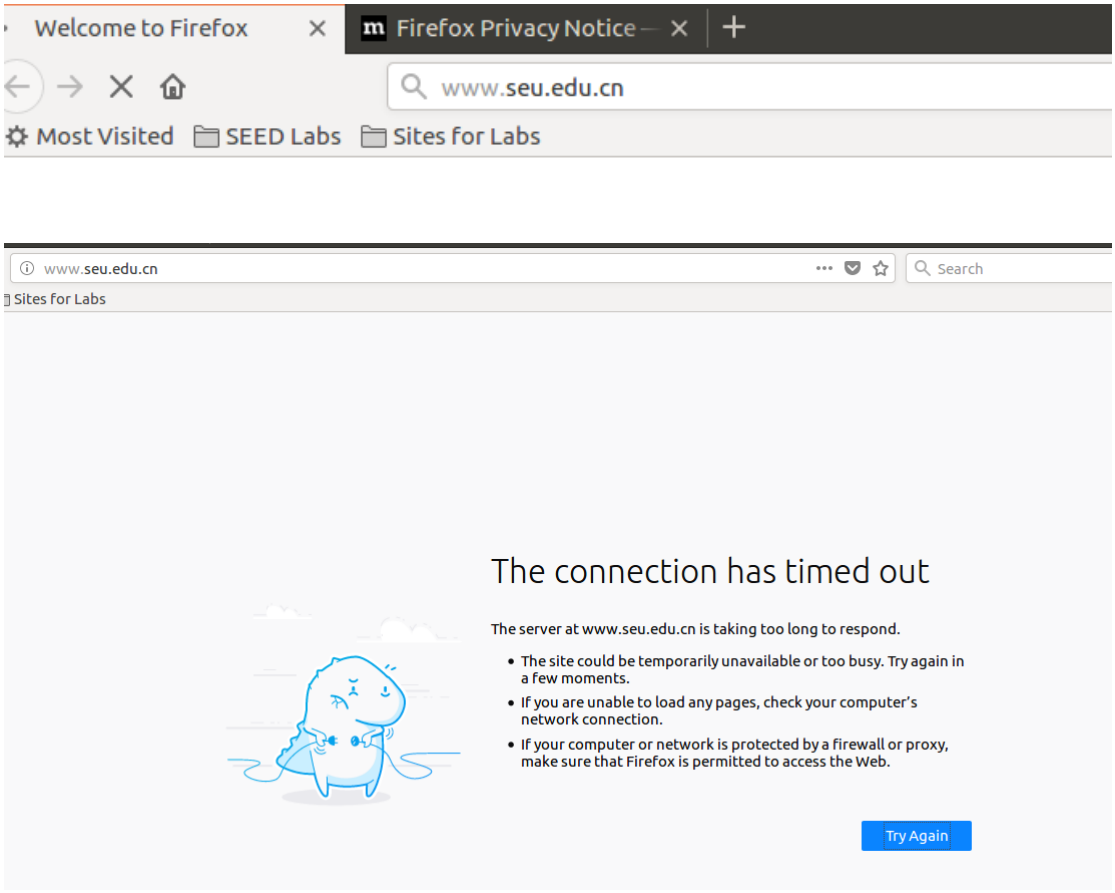
3. Prevent A from visiting an external web site：

我们阻止 A 对 seu 主页的访问，首先利用 Ping 命令查看一下 seu 的 ip：

```
[09/07/20]seed@VM:~$ ping www.seu.edu.cn
PING seu-ipv6.cache.saaswaf.com (121.194.14.142) 56(84) bytes of data.
^C64 bytes from 121.194.14.142: icmp_seq=1 ttl=128 time=65.6 ms
```

添加如下规则：

```
[09/07/20]seed@VM:~$ sudo ufw deny out from 192.168.114.129 to 121.194.14.142 po
rt 80
Rule added
```



此时一直卡在 firefox 浏览器的初始主页面。迟迟无法加载至东大的官方网站，这说明防火墙规则生效了。

Task2: Implementing a Simple Firewall

◆ 实验流程：

首先禁用 ufw，防止干扰实验 2 的进行：

```
[09/07/20]seed@VM:~$ sudo ufw disable
Firewall stopped and disabled on system startup
```

接着编写基于 Netfilter 的过滤代码：

这里共有 5 条规则：

A→B telnet：

```
// tcp
if (ip_header->protocol == 6)
{
   tcp_header = (struct tcphdr *)((__u32 *)ip_header + ip_header->ihl);
   src_port = (unsigned int)ntohs(tcp_header->source);
   dst_port = (unsigned int)ntohs(tcp_header->dest);
   // filter 2: B telnet A
   if (dst_port == 23)
   {
      print_address(ip_header);
      if (!check_address_src(ip_header, 192, 168, 114, 130))
      {
         printk(KERN_INFO "filter 2: src not match\n");
         return NF_ACCEPT;
      }
      if (!check_address_dst(ip_header, 192, 168, 114, 129))
      {
         printk(KERN_INFO "filter 2: dst not match\n");
         return NF_ACCEPT;
      }
      printk(KERN_INFO "filter 2: B telnet A\n");
      printk(KERN_INFO "filter 2: SRC_PORT: %d DST_PORT: %d\n", src_port, dst_port);
      return NF_DROP;
   }
}
return NF_ACCEPT;
```

B→A：telnet

```
  // filter 1: A telnet B
  if (dst_port == 23)
  {
     print_address(ip_header);
     if (!check_address_src(ip_header, 192, 168, 114, 129))
     {
        printk(KERN_INFO "filter 1: src not match\n");
        return NF_ACCEPT;
     }
     if (!check_address_dst(ip_header, 192, 168, 114, 130))
     {
        printk(KERN_INFO "filter 1: dst not match\n");
        return NF_ACCEPT;
     }
     printk(KERN_INFO "filter 1: A telnet B\n");
     printk(KERN_INFO "filter 1: SRC_PORT: %d DST_PORT: %d\n", src_port, dst_port);
     return NF_DROP;
  }
```

Prevent A from browsing SEU home web site:

```
// filter 3: A http www.seu.edu.cn
if (dst_port == 80)
{
   print_address(ip_header);
   if (!check_address_src(ip_header, 192, 168, 114, 129))
   {
      printk(KERN_INFO "filter 3: src not match\n");
      return NF_ACCEPT;
   }
   if (!check_address_dst(ip_header, 121, 194, 14, 142))
   {
      printk(KERN_INFO "filter 3: dst not match\n");
      return NF_ACCEPT;
   }
   printk(KERN_INFO "filter 3: A http fudan.edu.cn\n");
   printk(KERN_INFO "filter 3: SRC_PORT: %d DST_PORT: %d\n", src_port, dst_port);
   return NF_DROP;
}
```

A ping B：

```
// filter 4: A ping B
if (icmp_header->type == 8)
{
   print_address(ip_header);
   if (!check_address_src(ip_header, 192, 168, 114, 129))
   {
      printk(KERN_INFO "filter 4: src not match\n");
      return NF_ACCEPT;
   }
   if (!check_address_dst(ip_header, 192, 168, 114, 130))
   {
      printk(KERN_INFO "filter 4: dst not match\n");
      return NF_ACCEPT;
   }
   printk(KERN_INFO "filter 4: A ping B\n");
   printk(KERN_INFO "filter 4: SRC_PORT: %d DST_PORT: %d\n", src_port, dst_port);
   return NF_DROP;
}
```

A ssh B：

```
// filter 5: A ssh B
if (dst_port == 22)
{
   print_address(ip_header);
   if (!check_address_src(ip_header, 192, 168, 114, 129))
   {
      printk(KERN_INFO "filter 5: src not match\n");
      return NF_ACCEPT;
   }
   if (!check_address_dst(ip_header, 192, 168, 114, 130))
   {
      printk(KERN_INFO "filter 5: dst not match\n");
      return NF_ACCEPT;
   }
   printk(KERN_INFO "filter 5: A ssh B\n");
   printk(KERN_INFO "filter 5: SRC_PORT: %d DST_PORT: %d\n", src_port, dst_port);
   return NF_DROP;
}
```

编写 makefile 文件：

```
obj-m += task2.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
~
```

在当前工作目录下进行编译：

```
[09/07/20]seed@VM:~$ cd Desktop/lab5
[09/07/20]seed@VM:~/.../lab5$ cd task2
[09/07/20]seed@VM:~/.../task2$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Desktop/lab5/task2 modu
les
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/seed/Desktop/lab5/task2/task2.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Desktop/lab5/task2/task2.mod.o
  LD [M]  /home/seed/Desktop/lab5/task2/task2.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[09/07/20]seed@VM:~/.../task2$
```

再利用 insmod 命令加载该模块：

```
[09/07/20]seed@VM:~/.../task2$ sudo insmod task2.ko
```

这时候尝试连接 telnet 服务器(位于 VM B)：

```
[09/07/20]seed@VM:~/.../task2$ telnet 192.168.114.130
Trying 192.168.114.130...
```

可以看到过滤代码生效了，利用 dmesg 看看内核消息：

```
[62678.808944] filter 1: A telnet B
[62678.808945] filter 1: SRC_PORT: 48728 DST_PORT: 23
[62819.289538] filter SRC: 192.168.114.129
[62819.289540] filter DST: 192.168.114.130
[62819.289540] filter 1: A telnet B
[62819.289541] filter 1: SRC_PORT: 48730 DST_PORT: 23
[62820.301338] filter SRC: 192.168.114.129
[62820.301432] filter DST: 192.168.114.130
[62820.301443] filter 1: A telnet B
[62820.301466] filter 1: SRC_PORT: 48730 DST_PORT: 23
[62822.316864] filter SRC: 192.168.114.129
[62822.316899] filter DST: 192.168.114.130
[62822.316902] filter 1: A telnet B
[62822.316905] filter 1: SRC_PORT: 48730 DST_PORT: 23
[62826.477175] filter SRC: 192.168.114.129
[62826.477209] filter DST: 192.168.114.130
[62826.477213] filter 1: A telnet B
[62826.477216] filter 1: SRC_PORT: 48730 DST_PORT: 23
[62834.669254] filter SRC: 192.168.114.129
[62834.669286] filter DST: 192.168.114.130
[62834.669290] filter 1: A telnet B
```

可以看到确实是由于新加载的内核模块起到了防火墙的作用。

再尝试从 B 上 telnetA：

```
[09/07/20]seed@VM:~$ telnet 192.168.114.129
Trying 192.168.114.129...
```

```
[62930.263308] filter 2: B telnet A
[62930.263331] filter 2: SRC_PORT: 37056 DST_PORT: 23
[62931.272776] filter SRC: 192.168.114.130
[62931.272807] filter DST: 192.168.114.129
[62931.272811] filter 2: B telnet A
[62931.272814] filter 2: SRC_PORT: 37056 DST_PORT: 23
[62933.289209] filter SRC: 192.168.114.130
[62933.289258] filter DST: 192.168.114.129
[62933.289263] filter 2: B telnet A
[62933.289267] filter 2: SRC_PORT: 37056 DST_PORT: 23
[62937.352871] filter SRC: 192.168.114.130
[62937.352909] filter DST: 192.168.114.129
[62937.352913] filter 2: B telnet A
[62937.352916] filter 2: SRC_PORT: 37056 DST_PORT: 23
[62945.545004] filter SRC: 192.168.114.130
[62945.545038] filter DST: 192.168.114.129
[62945.545042] filter 2: B telnet A
[62945.545045] filter 2: SRC_PORT: 37056 DST_PORT: 23
```

尝试从 A ping B：

```
[09/07/20]seed@VM:~/.../task2$ ping 192.168.114.130
PING 192.168.114.130 (192.168.114.130) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
```

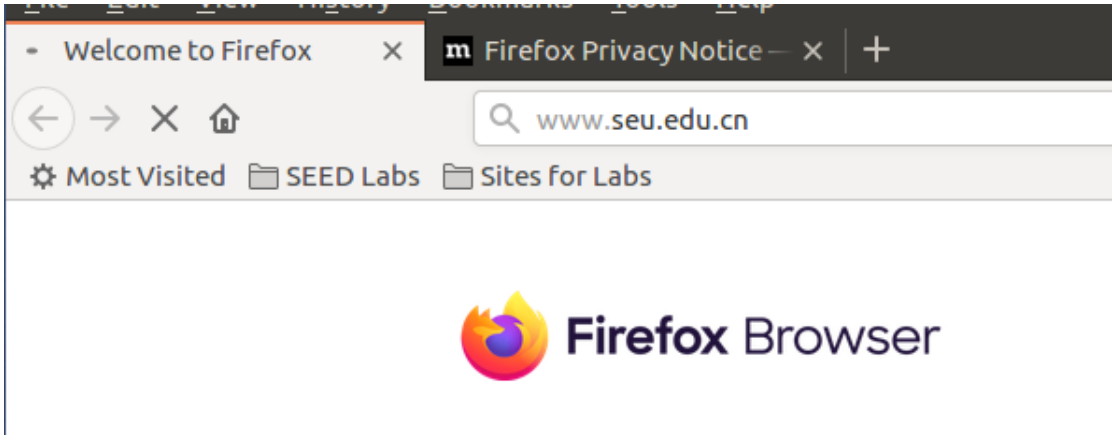
```
[62997.490795] filter 4: A ping B
[62997.490796] filter 4: SRC_PORT: 37056 DST_PORT: 23
[62998.509340] filter SRC: 192.168.114.129
[62998.509342] filter DST: 192.168.114.130
[62998.509343] filter 4: A ping B
[62998.509343] filter 4: SRC_PORT: 37056 DST_PORT: 23
[62999.533020] filter SRC: 192.168.114.129
[62999.533022] filter DST: 192.168.114.130
[62999.533022] filter 4: A ping B
[62999.533023] filter 4: SRC_PORT: 37056 DST_PORT: 23
[63000.557410] filter SRC: 192.168.114.129
[63000.557412] filter DST: 192.168.114.130
[63000.557412] filter 4: A ping B
```

再尝试 A ssh B:

```
[09/07/20]seed@VM:~/.../task2$ ssh 192.168.114.130
```

```
[63056.647309] filter 5: A ssh B
[63056.647310] filter 5: SRC_PORT: 59002 DST_PORT: 22
[63057.677139] filter SRC: 192.168.114.129
[63057.677172] filter DST: 192.168.114.130
[63057.677176] filter 5: A ssh B
[63057.677179] filter 5: SRC_PORT: 59002 DST_PORT: 22
[63059.692727] filter SRC: 192.168.114.129
[63059.692762] filter DST: 192.168.114.130
[63059.692766] filter 5: A ssh B
[63059.692769] filter 5: SRC_PORT: 59002 DST_PORT: 22
[63063.789327] filter SRC: 192.168.114.129
[63063.789361] filter DST: 192.168.114.130
[63063.789365] filter 5: A ssh B
```

最后尝试访问 SEU 主页：




```
[63128.301290] filter 3: A http fudan.edu.cn
[63128.301293] filter 3: SRC_PORT: 48134 DST_PORT: 80
[63128.556981] filter SRC: 192.168.114.129
[63128.557012] filter DST: 121.194.14.142
[63128.557016] filter 3: A http fudan.edu.cn
[63128.557019] filter 3: SRC_PORT: 48136 DST_PORT: 80
[63128.557030] filter SRC: 192.168.114.129
[63128.557033] filter DST: 121.194.14.142
[63128.557036] filter 3: A http fudan.edu.cn
[63128.557039] filter 3: SRC_PORT: 48138 DST_PORT: 80
[63128.812642] filter SRC: 192.168.114.129
[63128.812673] filter DST: 121.194.14.142
[63128.812677] filter 3: A http fudan.edu.cn
[63128.812680] filter 3: SRC_PORT: 48140 DST_PORT: 80
```

以上所有内核模块代码经过验证均可生效。

Task3: Evading Egress Filtering

◆ 实验流程：

首先利用 rmmod 命令移除之前的可加载内核模块 task2.ko，接着删除掉之前在 ufw 中设置的所有规则：

```
[09/07/20]seed@VM:~/.../task2$ sudo rmmod task2.ko
[09/07/20]seed@VM:~/.../task2$ sudo ufw delete 1
Deleting:
 deny out from 192.168.114.129 to 192.168.114.130 port 23
Proceed with operation (y|n)? y
Rules updated
[09/07/20]seed@VM:~/.../task2$ sudo ufw delete 1
Deleting:
 deny from 192.168.114.130 to 192.168.114.131 port 23
Proceed with operation (y|n)? y
Rules updated
[09/07/20]seed@VM:~/.../task2$ sudo ufw delete 1
Deleting:
 deny from 192.168.114.130 to 192.168.114.129 port 23
Proceed with operation (y|n)? y
Rules updated
[09/07/20]seed@VM:~/.../task2$ sudo ufw delete 1
Deleting:
 deny out from 192.168.114.129 to 121.194.14.142 port 80
Proceed with operation (y|n)? y
Rules updated
```

接着我们在 ufw 下加入两条规则：

```
[09/07/20]seed@VM:~/.../task2$ sudo ufw deny out from 192.168.114.129 to any por
t 23
Rule added
[09/07/20]seed@VM:~/.../task2$ sudo ufw deny out from 192.168.114.129 to 121.194
.14.142 port 80
Rule added
```

此时 A(192.168.114.129)已经无法向外 telnet 以及访问 www.seu.edu.cn 网站了。

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

这里令 VM C(192.168.114.131)作为 telnet 通讯的目标。利用 B(192.168.114.130)开辟一个 ssh 隧道：

```
[09/07/20]seed@VM:~/.../task2$ ssh -L 8000:192.168.114.131:23 192.168.114.130
The authenticity of host '192.168.114.130 (192.168.114.130)' can't be establishe
d.
ECDSA key fingerprint is SHA256:p1zAio6c1bI+8HDp5xa+eKRi561aFDaPE1/xq1eYzCI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.114.130' (ECDSA) to the list of known hosts.
seed@192.168.114.130'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 Sep  7 13:27:02 2020 from 192.168.114.129
```

再另一个终端上 telnet localhost 8000，在观察 wireshark：

1	2020-09-07 18:00:56.1086001…	192.168.114.130	192.168.114.131	TCP	74	38152 → 23 [SYN] Seq=2307841101 Win=29200 Len=0 MSS=1
26	2020-09-07 18:02:01.6471833…	192.168.114.130	192.168.114.129	SSH	134	Server: Encrypted packet (len=68)
27	2020-09-07 18:02:01.6472240…	192.168.114.129	192.168.114.130	TCP	66	59188 → 22 [ACK] Seq=3597890852 Ack=1165254476 Win=31
38	2020-09-07 18:02:31.8753986…	192.168.114.129	192.168.114.130	SSH	166	Client: Encrypted packet (len=100)
39	2020-09-07 18:02:31.8760485…	192.168.114.130	192.168.114.131	TCP	74	38156 → 23 [SYN] Seq=2929382483 Win=29200 Len=0 MSS=1
40	2020-09-07 18:02:31.9197261…	192.168.114.130	192.168.114.129	TCP	66	22 → 59188 [ACK] Seq=1165254476 Ack=3597890952 Win=27

可以看到 B(192.168.114.130)确实起到了一个类似于网桥的作用。

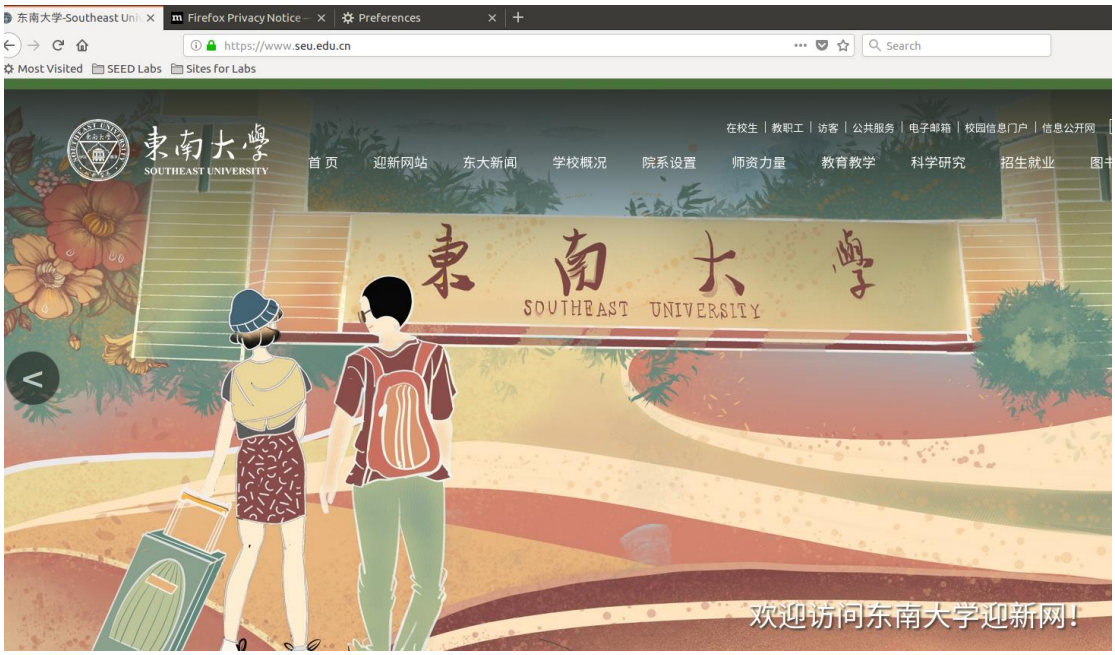
```
[09/07/20]seed@VM:~$ telnet localhost 8000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sun Sep  6 16:15:24 EDT 2020 from 192.168.114.130 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * 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.

[09/07/20]seed@VM:~$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:79:2e:a5
          inet addr:192.168.114.131  Bcast:192.168.114.255  Mask:255.255.255.0
```

3.a task: Connect to SEU using SSH Tunnel

输入以下命令：

```
[09/07/20]seed@VM:~$ ssh -D 9090 -C seed@192.168.114.131
The authenticity of host '192.168.114.131 (192.168.114.131)' can't be
d.
```

修改 firefox 的代理为 127.0.0.1：9090，然后在浏览器尝试访问：



此时可以正常浏览 SEU 主页了。

◆ 实验结论：

通过构建 SSH 隧道可以有效规避某些防火墙的过滤规则。

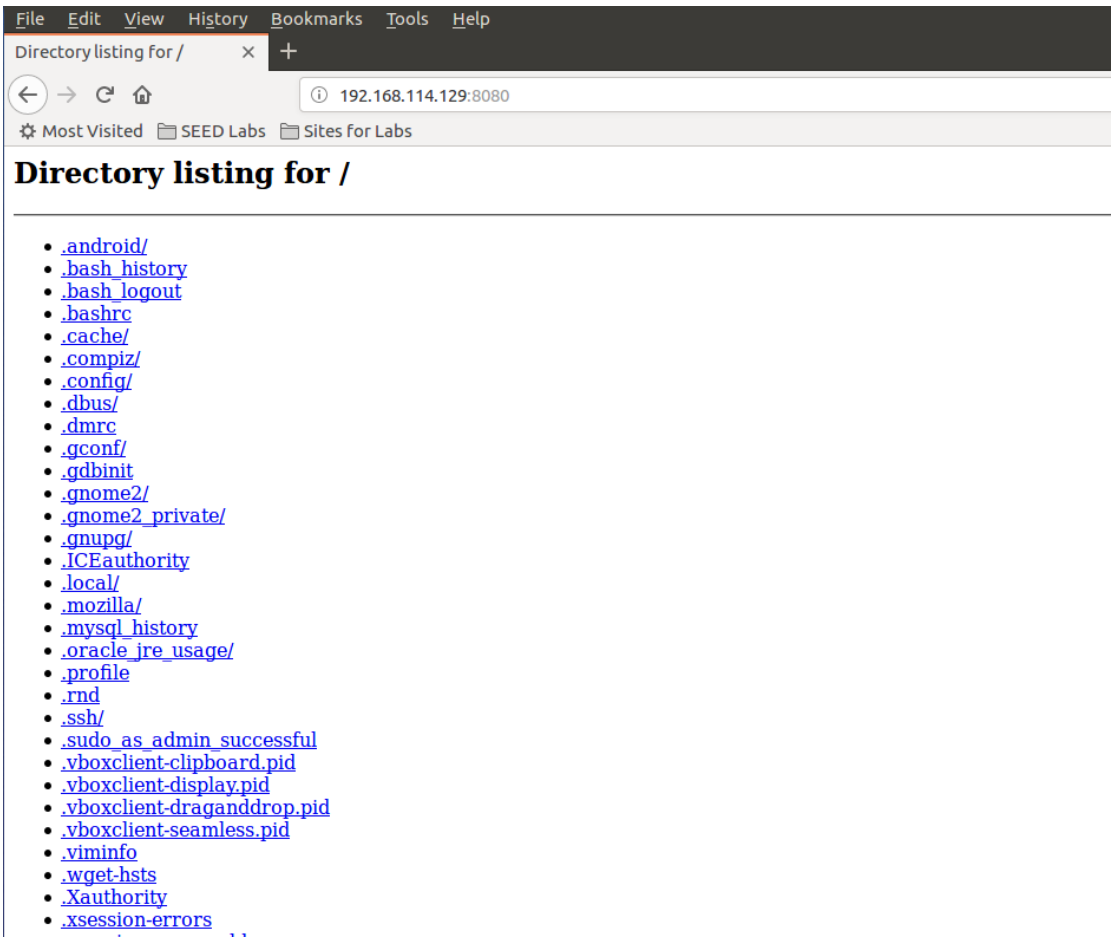
Task4: Evading Ingress Filtering

◆ 实验流程：

这里我们首先在机器 A(192.168.114.129)上开启一个简易的 httpserver：

```
[09/07/20]seed@VM:~$ sudo python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

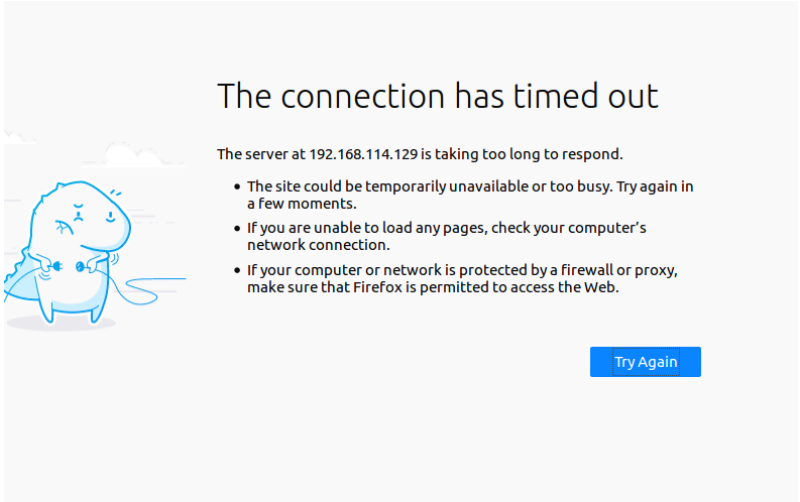
然后通过机器 C(192.168.114.131)进行访问：



可以正常访问，此时我们在 A 上添加一条规则：

```
[09/07/20]seed@VM:~$ sudo ufw deny in 8080
Rule added
Rule added (v6)
```

这条规则的作用是禁止任何外部的 8080 端口访问。此时我们发现 C 已经无法访问 A 的 8080 端口了：



此外同时禁止 A 机器的 ssh 访问：

```
[09/07/20]seed@VM:~$ sudo ufw deny in ssh
Rule added
Rule added (v6)
```

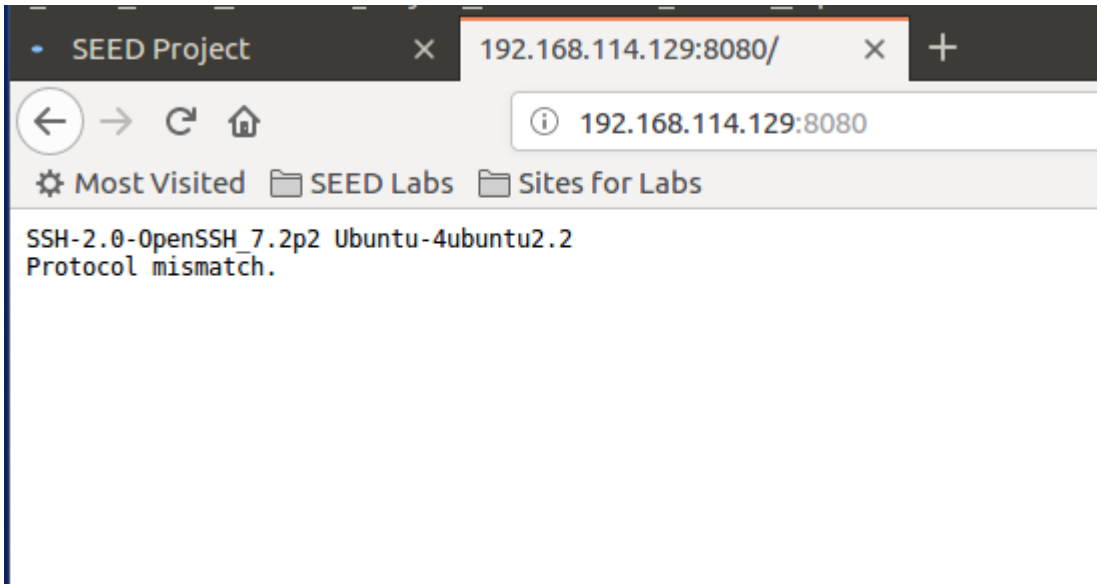
接下来我们在 A 上设置反向隧道：

```
[09/07/20]seed@VM:~$ ssh -fNR 7000:localhost:22 seed@192.168.114.131
seed@192.168.114.131's password:
```

在机器 C 上登陆 ssh：

```
[09/07/20]seed@VM:~$ ssh -p 7000 seed@localhost
ssh: connect to host localhost port 7000: Connection refused
[09/07/20]seed@VM:~$ ssh -p 7000 seed@localhost
The authenticity of host '[localhost]:7000 ([127.0.0.1]:7000)' can't be estab
hed.
ECDSA key fingerprint is SHA256:p1zAio6c1bI+8HDp5xa+eKRi561aFDaPE1/xq1eYzCI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:7000' (ECDSA) to the list of known hos
.
seed@localhost'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: Sat Sep  5 18:21:07 2020 from 10.0.2.128
```

此时可以访问 A 机器的 8080 端口了（不过页面发生变化了）：



◆ 实验结论：

反向 SSH 可以穿透内网。

Firewall Evasion Lab: Bypassing Firewalls using VPN

Task1: VM Setup

◆ 实验流程：

以之前配置好的实验环境作为本次实验的环境即可。

VM1(VPN Client):192.168.114.129(网卡为 ens33)

VM2(VPN Server):192.168.114.132(网卡为 ens38)

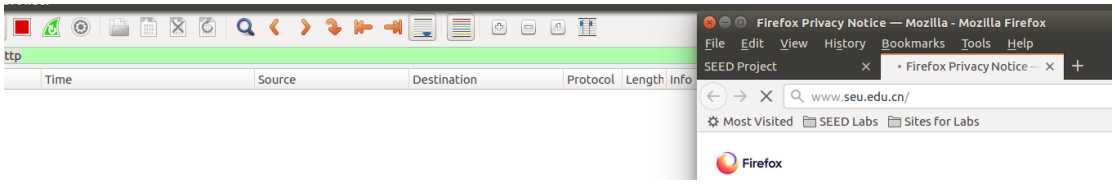
Task2: Setup Firewall

◆ 实验流程：

这里在 VM1 上配置防火墙：

```
[09/07/20]seed@VM:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
121.194.14.0/24            DENY OUT    Anywhere on ens33
```

尝试访问 SEU 的主页：



已经无法连通，并且 wireshark 也并没有抓到任何 http 请求报文。说明数据包已经被防火墙过滤了。

◆ 实验结论：

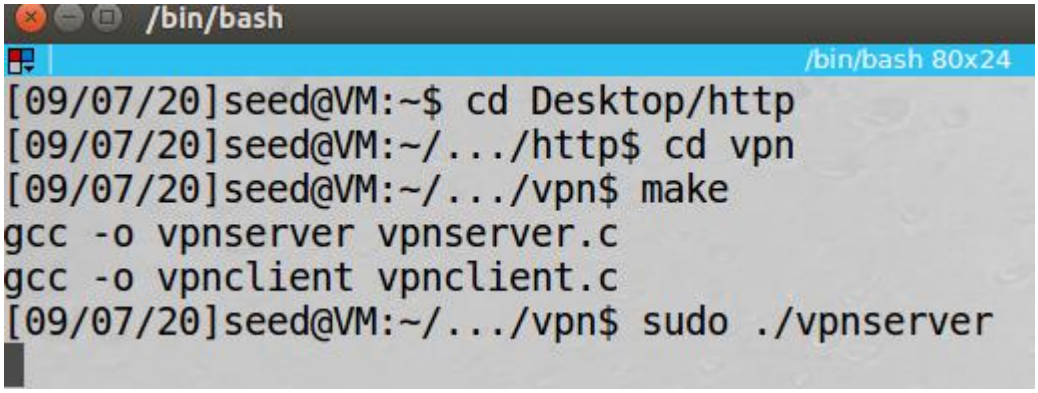
VM1 上设置的 ufw 规则已经成功扮演了一个防火墙的角色。

Task3: Bypassing Firewall using VPN

◆ 实验流程：

Step1 Run the VPN server:

在 VM2 上运行 VPN 服务端：



利用 ifconfig -a 命令可以看到虚拟接口 tun0 还未被配置，此时我们为他配置一个 ip 地址(192.168.53.1)：

```
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-
-00
          POINTOPOINT NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

[09/07/20]seed@VM:~$ sudo ifconfig tun0 192.168.53.1/24 up
```

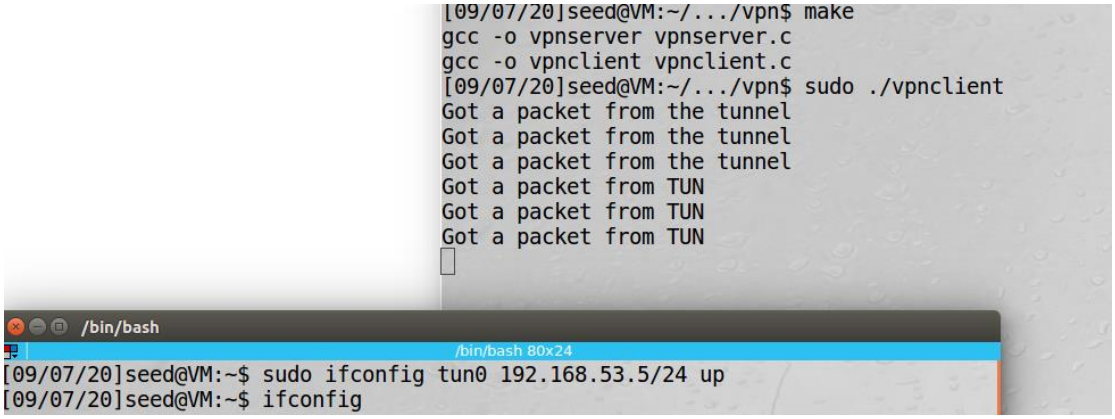
```
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-
-00
          inet addr:192.168.53.1  P-t-P:192.168.53.1  Mask:255.255.255.
          inet6 addr: fe80::fb1e:ffc1:ac1b:382f/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

同时要打开路由转发功能：

```
[09/07/20]seed@VM:~$  sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Step2 Run VPN Client：

如图所示，运行 VPN 客户端，同时配置 IP：



Step3 Set up Routing：

将 SEU 主页所在的 IP 子网引流至虚拟接口 tun0：

```
[09/07/20]seed@VM:~$ sudo route add -net 121.194.14.0/24 tun0
```

在客户机和服务器上均是如此。

Step 4 Set Up NAT on Server VM：

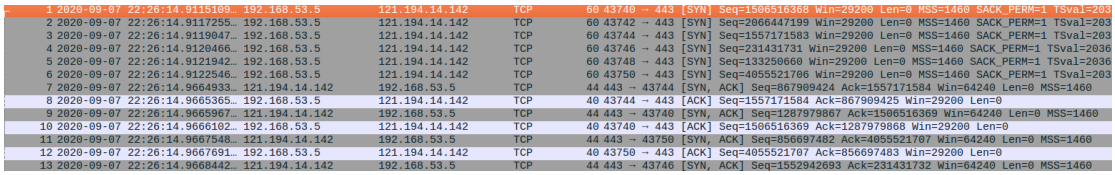
在数据包最终到达用户前，它会先被转发至 VPN 服务器，因为原本的数据请求是从 VPN 服务器发出的，数据包会通过原路由反向传播。

按如下命令在 VM Server 下配置：

```
09/07/20]seed@VM:~$ sudo iptables -t nat -F
09/07/20]seed@VM:~$ sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o eth8
09/07/20]seed@VM:~$ sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o ens38
```

验证：

为了验证我们从 VM1 上成功访问 SEU 主页是通过 VPN 进行的，我们将通过 Wireshark 抓包来确认这一点：



1	2020-09-07 22:26:14.9115109…	192.168.53.5	121.194.14.142	TCP	60 43740 → 443 [SYN] Seq=1506516368 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=203
2	2020-09-07 22:26:14.9117255…	192.168.53.5	121.194.14.142	TCP	60 43742 → 443 [SYN] Seq=2066447199 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=203
3	2020-09-07 22:26:14.9119047…	192.168.53.5	121.194.14.142	TCP	60 43744 → 443 [SYN] Seq=1557171583 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=203
4	2020-09-07 22:26:14.9120466…	192.168.53.5	121.194.14.142	TCP	60 43746 → 443 [SYN] Seq=231431731 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2036
5	2020-09-07 22:26:14.9121942…	192.168.53.5	121.194.14.142	TCP	60 43748 → 443 [SYN] Seq=133250660 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2036
6	2020-09-07 22:26:14.9122546…	192.168.53.5	121.194.14.142	TCP	60 43750 → 443 [SYN] Seq=4055521706 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=203
7	2020-09-07 22:26:14.9664933…	121.194.14.142	192.168.53.5	TCP	44 443 → 43744 [SYN, ACK] Seq=867909424 Ack=1557171584 Win=64240 Len=0 MSS=1460
8	2020-09-07 22:26:14.9665365…	192.168.53.5	121.194.14.142	TCP	40 43744 → 443 [ACK] Seq=1557171584 Ack=867909425 Win=29200 Len=0
9	2020-09-07 22:26:14.9665967…	121.194.14.142	192.168.53.5	TCP	44 443 → 43740 [SYN, ACK] Seq=1287979867 Ack=1506516369 Win=64240 Len=0 MSS=1460
10	2020-09-07 22:26:14.9666102…	192.168.53.5	121.194.14.142	TCP	40 43740 → 443 [ACK] Seq=1506516369 Ack=1287979868 Win=29200 Len=0
11	2020-09-07 22:26:14.9667548…	121.194.14.142	192.168.53.5	TCP	44 443 → 43750 [SYN, ACK] Seq=856697482 Ack=4055521707 Win=64240 Len=0 MSS=1460
12	2020-09-07 22:26:14.9667691…	192.168.53.5	121.194.14.142	TCP	40 43750 → 443 [ACK] Seq=4055521707 Ack=856697483 Win=29200 Len=0
13	2020-09-07 22:26:14.9668442…	121.194.14.142	192.168.53.5	TCP	44 443 → 43746 [SYN, ACK] Seq=1552942693 Ack=231431732 Win=64240 Len=0 MSS=1460

这是对虚拟接口 tun0 的监控(VM1 的 tun0)，可以看到，由于路由表的作用，121.194.14.142 的流量被定向至 VPN 的网段(192.168.53.5)，这里由 VPN 完成了对 SEU 的访问。