Firewall Lab

2022 年 12 月 28 日

姓　　名：	李晨漪
学　　号：	202000210103
教　　师：	郭山清
学　　院：	山东大学网络空间安全学院
班　　级：	20 级网安 2 班

目录

1 第一部分：Firewall Exploration Lab

2 实验目的

这个实验室的学习目标有两个方面：学习防火墙是如何工作的，以及为一个网络建立一个简单的防火墙。学生将首先实现一个简单的无状态数据包过滤防火墙，它检查数据包，并根据防火墙规则决定是否删除还是转发数据包。通过这个实现任务，学生可以得到关于防火墙如何工作的基本想法。实际上，Linux 已经有了一个内置的防火墙，也是基于网络过滤器的。学生将被赋予一个简单的网络拓扑，并被要求使用 iptables 来建立防火墙规则来保护网络。学生们还将接触到其他一些有趣的应用程序。本实验室涵盖以下主题：

1. 防火墙
2. 网络过滤器
3. 可加载的内核模块
4. 使用 iptables 建立防火墙规则
5.iptables 各种应用程序

3 实验原理

3.1 iptables 简介

iptables 是集成在 Linux 内核中的包过滤防火墙系统。使用 iptables 可以添加、删除具体的过滤规则，iptables 默认维护着 4 个表和 5 个链，所有的防火墙策略规则都被分别写入这些表与链中。

“四表”是指 iptables 的功能，默认的 iptable s 规则表有 filter 表（过滤规则表）、nat 表（地址转换规则表）、mangle（修改数据标记位规则表）、raw（跟踪数据表规则表）：

filter 表：控制数据包是否允许进出及转发，可以控制的链路有 INPUT、FORWARD 和 OUTPUT。

nat 表：控制数据包中地址转换，可以控制的链路有 PREROUTING、INPUT、OUTPUT 和 POSTROUTING。

mangle：修改数据包中的原数据，可以控制的链路有 PREROUTING、INPUT、OUTPUT、FORWARD 和 POSTROUTING。

raw：控制 nat 表中连接追踪机制的启用状况，可以控制的链路有 PREROUTING、OUTPUT。

“五链”是指内核中控制网络的 NetFilter 定义的 5 个规则链。每个规则表中包含多个数据链：INPUT（入站数据过滤）、OUTPUT（出站数据过滤）、FORWA RD（转发数据过滤）、PREROUTING（路由前过滤）和 POSTROUTING（路由后过滤），防火墙规则需要写入到这些具体的数据链中。

3.2 有状态防火墙

状态防火墙只定义出站规则就好，不需要定义入站规则。当用户打开浏览器访问那个 web 服务时，当数据包到达防火墙时，状态检测引擎会检测到这是一个发起连接的初始数据包 (由 SYN 标志)，然后它就会把这个数据包中的信息与防火墙规则作比较，如果没有相应规则允许，防火墙就会拒绝这次连接，当然在这里它会发现有一条规则允许用户访问外部 WEB 服务，于是它允许数据包外出并且在状态表中新建一条会话，通常这条会话会包括此连接的源地址、源端口、目标地址、目标端口、连接时间等信息，对于 TCP 连接（只针对 TCP，对 UDP 创建模拟的虚拟连接, 对 ICMP 无效），它还应该会包含序列号和标志位等信息。当后续数据包到达时，如果这个数据包不含 SYN

标志，也就是说这个数据包不是发起一个新的连接时，状态检测引擎就会直接把它的信息与状态表中的会话条目进行比较，如果信息匹配，就直接允许数据包通过，这样不再去接受规则的检查，提高了效率，如果信息不匹配，数据包就会被丢弃或连接被拒绝，并且每个会话还有一个超时值，过了这个时间，相应会话条目就会被从状态表中删除掉 (这也就是为什么有一些客户端软件去访问防火墙背后的服务端时，如果客户端不定时去发一条信息连一下服务器的话，服务器就无法再给客户端主动发消息了，因为状态会话已经超时被删除了)。

就外部 WEB 网站对用户的响应包来说，由于状态检测引擎会检测到返回的数据包属于 WEB 连接的会话，所以它会动态打开端口以允许返回包进入，传输完毕后又动态地关闭这个端口，这样就避免了普通包过滤防火墙那种静态地开放所有高端端口的危险做法，同时由于有会话超时的限制，它也能够有效地避免外部的 DoS 攻击，并且外部伪造的 ACK 数据包也不会进入，因为它的数据包信息不会匹配状态表中的会话条目。

4 实验准备

实验环境：Ubuntu20.04

4.1 Docker 容器

基本操作简介：

```
// Aliases for the Compose commands above
$ dcbuild # Alias for: docker-compose build
$ dcup # Alias for: docker-compose up
$ dcdown # Alias for: docker-compose down

$ dockps // Alias for: docker ps --format "{{.ID}} {{.Names}}"
$ docksh <id> // Alias for: docker exec -it <id> /bin/bash
```

4.2 DNS 缓存操作

```
# rndc dumpdb -cache // Dump the cache to the specified file
# rndc flush // Flush the DNS cache
```

5 实验步骤及运行结果

5.1 Task 1: Implementing a Simple Firewall

5.1.1 Task 1.A: Implement a Simple Kernel Module

在命令行执行以下命令：

```
$ sudo insmod hello.ko (inserting a module)
$ lsmod | grep hello (list modules)
$ sudo rmmod hello (remove the module)
```

```
$ dmesg (check the messages)
```

hello.ko 是一个简单的可加载的内核模块。当模块加载时它打印出了“Hello World!”；当模块从内核中移除时，它会打印出“Bye-bye World!”。

- 运行截图

```
[ 4283.609661] Hello World!
[ 4307.245262] Bye-bye World!.
```

图 1:

5.1.2 Task 1.B: Implement a Simple Firewall Using Netfilter

```c
//seedFilter.c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/icmp.h>
#include <linux/if_ether.h>
#include <linux/inet.h>

static struct nf_hook_ops hook1, hook2,hook3,hook4,hook5,hook6,hook7
    ,hook8;

//blocking ping vm:10.9.0.1
unsigned int blockICMP(void *priv, struct sk_buff *skb,
                       const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct icmphdr *icmph;

    //u16  port   = 53;
    char ip[16] = "10.9.0.1";
    u32  ip_addr;

    if (!skb) return NF_ACCEPT;

    iph = ip_hdr(skb);
    // Convert the IPv4 address from dotted decimal to 32-bit binary
```

```
30      in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);
31
32      if (iph->protocol == IPPROTO_ICMP) {
33          icmph = icmp_hdr(skb);
34          if (iph->daddr == ip_addr && icmph->type==ICMP_ECHO){
35              printk(KERN_WARNING "*** Dropping %pI4 (ICMP)\n", &(iph
                    ->daddr));
36              return NF_DROP;
37           }
38      }
39      return NF_ACCEPT;
40  }
41
42  unsigned int telnetFilter(void *priv, struct sk_buff *skb,
43                            const struct nf_hook_state *state)
44  {
45      struct iphdr *iph;
46      struct tcphdr *tcph;
47
48      iph = ip_hdr(skb);
49      tcph=(void *)iph+iph->ihl*4;
50
51      if (iph->protocol == IPPROTO_TCP&&tcph->dest==htons(23)) {
52          printk(KERN_INFO "*** Dropping telnet packet to %d.%d.%d.%d\n
                ",
53          ((unsigned char*)&iph->daddr)[0],
54          ((unsigned char*)&iph->daddr)[1],
55          ((unsigned char*)&iph->daddr)[2],
56          ((unsigned char*)&iph->daddr)[3]);
57          return NF_DROP;
58           }
59      else{
60      return NF_ACCEPT;}
61  }
62
63  unsigned int blockUDP(void *priv, struct sk_buff *skb,
64                            const struct nf_hook_state *state)
65  {
66      struct iphdr *iph;
67      struct udphdr *udph;
68
69      u16  port    = 53;
70      char ip[16] = "8.8.8.8";
```

```
71      u32  ip_addr;
72
73      if (!skb) return NF_ACCEPT;
74
75      iph = ip_hdr(skb);
76      // Convert the IPv4 address from dotted decimal to 32-bit binary
77      in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);
78
79      if (iph->protocol == IPPROTO_UDP) {
80          udph = udp_hdr(skb);
81          if (iph->daddr == ip_addr && ntohs(udph->dest) == port){
82              printk(KERN_WARNING "*** Dropping %pI4 (UDP), port %d\n"
                    , &(iph->daddr), port);
83              return NF_DROP;
84           }
85      }
86      return NF_ACCEPT;
87  }
88
89  unsigned int printInfo(void *priv, struct sk_buff *skb,
90                  const struct nf_hook_state *state)
91  {
92      struct iphdr *iph;
93      char *hook;
94      char *protocol;
95
96      switch (state->hook){
97        case NF_INET_LOCAL_IN:     hook = "LOCAL_IN";     break;
98        case NF_INET_LOCAL_OUT:    hook = "LOCAL_OUT";    break;
99        case NF_INET_PRE_ROUTING:  hook = "PRE_ROUTING";  break;
100       case NF_INET_POST_ROUTING: hook = "POST_ROUTING"; break;
101       case NF_INET_FORWARD:      hook = "FORWARD";      break;
102       default:                   hook = "IMPOSSIBLE";   break;
103     }
104     printk(KERN_INFO "*** %s\n", hook); // Print out the hook info
105
106     iph = ip_hdr(skb);
107     switch (iph->protocol){
108       case IPPROTO_UDP:  protocol = "UDP";   break;
109       case IPPROTO_TCP:  protocol = "TCP";   break;
110       case IPPROTO_ICMP: protocol = "ICMP";  break;
111       default:           protocol = "OTHER"; break;
112
```

```
113     }
114     // Print out the IP addresses and protocol
115     printk(KERN_INFO "    %pI4  -> %pI4 (%s)\n",
116                      &(iph->saddr), &(iph->daddr), protocol);
117 
118     return NF_ACCEPT;
119 }
120 
121 int registerFilter(void) {
122     printk(KERN_INFO "Registering filters.\n");
123 
124     hook1.hook = printInfo;
125     hook1.hooknum = NF_INET_LOCAL_OUT;
126     hook1.pf = PF_INET;
127     hook1.priority = NF_IP_PRI_FIRST;
128     nf_register_net_hook(&init_net, &hook1);
129 
130     hook3.hook = printInfo;
131     hook3.hooknum = NF_INET_POST_ROUTING;
132     hook3.pf = PF_INET;
133     hook3.priority = NF_IP_PRI_FIRST;
134     nf_register_net_hook(&init_net, &hook3);
135 
136     hook4.hook = printInfo;
137     hook4.hooknum = NF_INET_LOCAL_IN;
138     hook4.pf = PF_INET;
139     hook4.priority = NF_IP_PRI_FIRST;
140     nf_register_net_hook(&init_net, &hook4);
141 
142     hook5.hook = printInfo;
143     hook5.hooknum = NF_INET_FORWARD;
144     hook5.pf = PF_INET;
145     hook5.priority = NF_IP_PRI_FIRST;
146     nf_register_net_hook(&init_net, &hook5);
147 
148     hook6.hook = printInfo;
149     hook6.hooknum = NF_INET_PRE_ROUTING;
150     hook6.pf = PF_INET;
151     hook6.priority = NF_IP_PRI_FIRST;
152     nf_register_net_hook(&init_net, &hook6);
153 
154     hook2.hook = blockUDP;
155     hook2.hooknum = NF_INET_POST_ROUTING;
```

```
156     hook2.pf = PF_INET;
157     hook2.priority = NF_IP_PRI_FIRST;
158     nf_register_net_hook(&init_net, &hook2);
159 
160     hook7.hook = blockICMP;
161     hook7.hooknum = NF_INET_POST_ROUTING;
162     hook7.pf = PF_INET;
163     hook7.priority = NF_IP_PRI_FIRST;
164     nf_register_net_hook(&init_net, &hook7);
165 
166     hook8.hook = telnetFilter;
167     hook8.hooknum = NF_INET_POST_ROUTING;
168     hook8.pf = PF_INET;
169     hook8.priority = NF_IP_PRI_FIRST;
170     nf_register_net_hook(&init_net, &hook8);
171 
172     return 0;
173 }
174 
175 void removeFilter(void) {
176     printk(KERN_INFO "The filters are being removed.\n");
177     nf_unregister_net_hook(&init_net, &hook1);
178     nf_unregister_net_hook(&init_net, &hook2);
179     nf_unregister_net_hook(&init_net, &hook3);
180     nf_unregister_net_hook(&init_net, &hook4);
181     nf_unregister_net_hook(&init_net, &hook5);
182     nf_unregister_net_hook(&init_net, &hook6);
183     nf_unregister_net_hook(&init_net, &hook7);
184     nf_unregister_net_hook(&init_net, &hook8);
185 }
186 
187 module_init(registerFilter);
188 module_exit(removeFilter);
189 
190 MODULE_LICENSE("GPL");
```

- **解释**

task1：hook2 阻止 UDP 数据包。

task2：hook1，hook3，hook4，hook5，hook6 将 printInfo 函数连接到所有的网络过滤器钩子上。下面是钩子号的宏。

```
1 NF_INET_PRE_ROUTING
2 NF_INET_LOCAL_IN
```

```
3 NF_INET_FORWARD
4 NF_INET_LOCAL_OUT
5 NF_INET_POST_ROUTING
```

task3：hook7 防止其他计算机 ping 虚拟机；hook8 防止其他计算机远程网络进入虚拟机。且 hook7 和 hook8 都连接 NF_INET_POST_ROUTING。

- **运行截图**

task1. 阻止 UDP 数据包，dig @8.8.8.8 www.example.com 失败。

```
[12/07/22]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

图 2:

此时可加载内核模块的 printInfo 信息。

```
[12/12/22]seed@VM:~/.../packet_filter$ dmesg
[ 2458.261118] Registering filters.
[ 2540.569176] *** LOCAL_OUT
[ 2540.569177]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 2540.571285] *** LOCAL_OUT
[ 2540.571287]     10.0.2.36  --> 8.8.8.8 (UDP)
[ 2540.571294] *** Dropping 8.8.8.8 (UDP), port 53
[ 2545.566736] *** LOCAL_OUT
[ 2545.566738]     10.0.2.36  --> 8.8.8.8 (UDP)
[ 2545.566749] *** Dropping 8.8.8.8 (UDP), port 53
[ 2550.565962] *** LOCAL_OUT
[ 2550.565963]     10.0.2.36  --> 8.8.8.8 (UDP)
[ 2550.565975] *** Dropping 8.8.8.8 (UDP), port 53
[ 2571.387437] The filters are being removed.
```

图 3:

注：task2 截图见上述解释。

task3. 以下命令失败。

```
1 ping 10.9.0.1
2 telnet 10.9.0.1
```

```
[12/12/22]seed@VM:~$ ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 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
^C
--- 10.9.0.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3072ms

[12/12/22]seed@VM:~$ telnet 10.9.0.1
Trying 10.9.0.1...
telnet: Unable to connect to remote host: Connection timed out
```

图 4:

此时可加载内核模块的 printInfo 信息。

```
[17417.719935] *** Dropping 10.9.0.1 (ICMP)
[17440.716610] *** LOCAL_OUT
[17440.716613]     10.9.0.1  --> 10.9.0.1 (TCP)
[17440.716626] *** Dropping telnet packet to 10.9.0.1
[17441.746478] *** LOCAL_OUT
[17441.746562]     10.9.0.1  --> 10.9.0.1 (TCP)
[17441.746648] *** Dropping telnet packet to 10.9.0.1
```

图 5:

注：task2 截图见上述 task1、task3 中 printInfo 信息。

- **task2 在什么条件下将被调用每个钩子函数**

(1)NF_INET_PRE_ROUTING: 除了混杂模式, 所有数据包都将经过这个钩子点。它上面注册的钩子函数在路由判决之前被调用。

(2) NF_INETLOCAL_IN: 数据包要进行路由判决, 以决定需要被转发还是发往本机。前一种情况下, 数据包将前往转发路径; 而后一种情况下, 数据包将通过这个钩子点, 之后被发送到网络协议栈, 并最终被主机接收。

(3) NF_INET_FORWARD: 需要被转发的数据包会到达这个钩子点。这个钩子点对于实现一个防火墙是十分重要的。

(4) NF_INET_LOCAL_OUT: 这是本机产生的数据包到达的第一个钩子点。

(5) NF_INET_POST_ROUTING: 需要被转发或者由本机产生的数据包都会经过这个钩子点。源网络地址转换 (source network translation, SNAT) 就是用这个钩子点实现的。

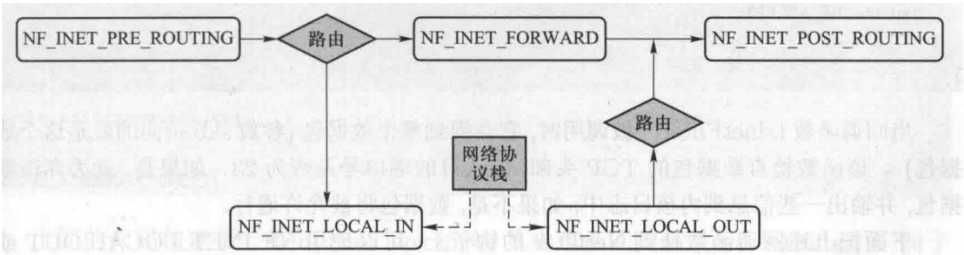

图 14.3 IPv4 中的 Netfilter 钩子函数

图 6:

5.2 Task 2: Experimenting with Stateless Firewall Rules

5.2.1 Task 2.A: Protecting the Router

Step1：执行以下 iptables 命令，然后尝试从 10.9.0.5 访问它。

```
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
iptables -P OUTPUT DROP #Set default rule for OUTPUT
iptables -P INPUT DROP #Set default rule for INPUT
```

Step2：清除下 iptables 命令。

```
iptables -F
iptables -P OUTPUT ACCEPT
iptables -P INPUT ACCEPT
```

- **解释**

1. 规则 1 接收 INPUT（入站数据过滤）处 icmp-echo-request 数据包。
2. 规则 2 接收 OUTPUT（出站数据过滤）处 icmp-echo-reply 数据包。
3. 丢弃 INPUT 处不符合规则 1 的数据包。
4. 丢弃 OUTPUT 处不符合规则 2 的数据包。

5.ping 和 ping 回复包符合规则 1、2 可被接收过滤。所以可以 ping，而 telnet 属于 TCP/IP 协议不能被过滤，故失败。

- **运行截图**

在 10.9.0.5 上对 router 操作。

ping 成功。

```
root@fc07867ad31b:/# ping seed-router
PING seed-router (10.9.0.11) 56(84) bytes of data.
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.185
 ms
```

图 7:

telnet 失败。

```
root@fc07867ad31b:/# telnet seed-router
Trying 10.9.0.11...
^C
```

图 8:

5.2.2 Task 2.B: Protecting the Internal Network

在 FORWAERD 上实现：1. 外部主机无法 ping 内部主机。2. 外部主机可以 ping 路由器。3. 内部主机可以在主机外部进行 ping 操作。4. 应阻止内部和外部网络之间的所有其他数据包。

```
root@53f7f54e22b2:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reques
t -j DROP
```

图 9:

```
root@53f7f54e22b2:/# iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-reques
t -j ACCEPT
root@53f7f54e22b2:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply
-j ACCEPT
```

图 10:

```
root@53f7f54e22b2:/# iptables -P FORWARD DROP
root@53f7f54e22b2:/#
```

图 11:

iptables 表如图。

```
root@53f7f54e22b2:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            0.0.0.0/0           icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0           icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0           icmptype 0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@53f7f54e22b2:/#
```

图 12:

- **解释**

1.eth0 是连接外部主机的接口，eth1 是连接内部主机的接口。

2. 规则 1 丢弃进入 eth0 的 icmp-echo-request 数据包。外部主机的 ping 请求包无法进入 eth0 到达内部主机。即实现：外部主机无法 ping 内部主机、外部主机可以 ping 路由器。

3. 规则 2 接收进入 eth1 的 icmp-echo-request 数据包。规则 3 接收进入 eth0 的 icmp-echo-reply 数据包。可实现要求 3。

4. 规则 4 丢弃 FORWARD 处不符合规则 2、3 的数据包。可实现要求 4。

- **运行截图**

1. 外部主机无法 ping 内部主机。2. 外部主机可以 ping 路由器。4. 应阻止内部和外部网络之间的所有其他数据包。（telnet 失败）

```
12/19/22]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@8520c671f1ff:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
24 packets transmitted, 0 received, 100% packet loss, time 23617ms

root@8520c671f1ff:/# ping seed-router
PING seed-router (10.9.0.11) 56(84) bytes of data.
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=1 ttl=64 time=0.164
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=2 ttl=64 time=0.054
 ms
64 bytes from seed-router.net-10.9.0.0 (10.9.0.11): icmp_seq=3 ttl=64 time=0.055
 ms
^C
--- seed-router ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2042ms
rtt min/avg/max/mdev = 0.054/0.091/0.164/0.051 ms
root@8520c671f1ff:/# telnet 192.168.60.5
Trying 192.168.60.5...
```

图 13:

3. 内部主机可以在主机外部进行 ping 操作。

```
root@28830aa870eb:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.187 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.068 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.067 ms
^C
--- 10.9.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2052ms
rtt min/avg/max/mdev = 0.067/0.107/0.187/0.056 ms
root@28830aa870eb:/#
```

图 14:

5.2.3 Task 2.C: Protecting Internal Servers

要求:

1. 所有的内部主机都运行一个 telnet 服务器（侦听端口 23）。外部主机只能访问 192.168.60.5 上的 telnet 服务器，而不能访问其他内部主机。

2. 外部主机无法访问其他内部服务器。

3. 内部主机可以访问所有的内部服务器。

4. 内部主机无法访问外部服务器。

5. 在此任务中，不允许使用连接跟踪机制。它将在以后的任务中使用。

```
root@53f7f54e22b2:/# iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport
23 -j ACCEPT
root@53f7f54e22b2:/# iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport
23 -j ACCEPT
root@53f7f54e22b2:/# iptables -P FORWARD DROP
root@53f7f54e22b2:/#
```

图 15:

iptables 表如图。

```
root@53f7f54e22b2:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            192.168.60.5         tcp dpt:23
ACCEPT     tcp  --  192.168.60.5         0.0.0.0/0            tcp spt:23

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@53f7f54e22b2:/#
```

图 16:

- **解释**

1.eth0 是连接外部主机的接口，eth1 是连接内部主机的接口。

2. 规则 1 接收进入 eth0 的目的 IP 地址为 192.168.60.5 目的端口为 23 的 tcp 数据包。规则 2 接收源 IP 地址为 192.168.60.5 源端口为 23 的 tcp 数据包。可实现要求 1。

3. 规则 3 丢弃 FORWARD 处不符合规则 1、2 的数据包。即需要通过路由跨内部、外部网络的数据包都将无法通行。可实现要求 2、3、4。

- **运行截图**

1. 所有的内部主机都运行一个 telnet 服务器（侦听端口 23）。外部主机只能访问 192.168.60.5 上的 telnet 服务器，而不能访问其他内部主机。

```
root@8520c671f1ff:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
28830aa870eb login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

图 17:

```
root@8520c671f1ff:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@8520c671f1ff:/# telnet 192.168.60.7
Trying 192.168.60.7...
^C
root@8520c671f1ff:/#
```

图 18:

2. 外部主机无法访问其他内部服务器。

```
root@8520c671f1ff:/# nc 192.168.60.6 9090
ddddd
[12/19/22]seed@VM:~/.../Labsetup$ docksh de
root@de11fb26c40d:/# nc -lt 9090
```

图 19:

3. 内部主机可以访问所有的内部服务器。

```
root@28830aa870eb:/# nc -lt 9090
cccccc
```

```
root@de11fb26c40d:/# nc 192.168.60.5 9090
cccccc
```

图 20:

4. 内部主机无法访问外部服务器。

```
root@8520c671f1ff:/# nc -lt 9090
```

```
root@de11fb26c40d:/# nc 10.9.0.5 9090
dddd
```

图 21:

5.3 Task 3: Connection Tracking and Stateful Firewall

5.3.1 Task 3.A: Experiment with the Connection Tracking

ICMP 实验：

```
root@53f7f54e22b2:/#  conntrack -L
icmp     1 27 src=192.168.60.11 dst=192.168.60.5 type=8 code=0 id=63 src=192.168
.60.5 dst=192.168.60.11 type=0 code=0 id=63 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@53f7f54e22b2:/#
```

图 22:

UDP 实验：

```
root@dbbbed61f386:/#  conntrack -L
udp      17 25 src=10.9.0.5 dst=192.168.60.5 sport=36890 dport=9090 [UNREPLIED]
src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=36890 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

图 23:

TCP 实验：

```
root@dbbbed61f386:/#  conntrack -L
tcp      6 431995 ESTABLISHED src=10.9.0.5 dst=192.168.60.5 sport=55664 dport=90
90 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=55664 [ASSURED] mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
```

图 24:

注：一个状态防火墙会监视在一段时间内进出网络的数据包，通过它们之间的关系，把已经存在的网络连接找出来。有了这些连接的信息，防火墙就能设置基于这些连接的规则。这里“连接”的含义是一个广义的概念，它不同于 TCP 的连接，实际上指的是数据流。因此这个概念不仅仅适用于面向连接的协议 (如 TCP)，也适用于无连接协议 (如 UDP 和 ICMP) 。

- **ICMP 连接状态要保持多久？**

ICMP 连接：ICMP 同样不建立连接。然而，一些类型的 ICMP 消息有请求和应答模式。一对请求与应答被视为一个连接。例如，ICMP Echo 请求与应答。当一个状态防火墙看到一个请求消息时，它会认为这是一个新的连接；当它看到相应的应答消息时，会认为连接已经建立。因为 ICMP 消息仅有一轮请求与应答，当防火墙看到应答时，会认为连接被建立并同时终止。

建立连接：一对 ICMP 请求与应答。维持连接：ICMP 仅有一轮请求应答，应答出现，连接被建立并同时终止。

- **UDP 连接状态要保持多久？**

UDP 连接：UDP 不是面向连接的协议，因此不存在建立连接和终止连接的步取，这使得维持连接状态十分困难。然而，当一个 UDP 客户端和服务器开始交换数据包时，状态防火墙就认为连接已经建立了。如果一段时间内没有数据包交换，状态防火墙就认为连接已经终止。

建立连接：UDP 客户端和服务器开始交换数据包。维持连接：是否有数据包交换。

- **TCP 连接状态要保持多久？**

TCP 连接：TCP 是一个面向连接的协议。TCP 的通信双方必须首先使用三次握手协议来建立连接，当数据传输完毕之后，通信双方需要使用另一个协议来终止连接。状态防火墙可以监视这些协议并记录连接状态。

建立连接：三次握手。终止连接：终止协议。

5.3.2 Task 3.B: Setting Up a Stateful Firewall

重写 task2.c。添加一个规则，允许内部主机访问任何外部服务器。

```
root@dbbbed61f386:/# iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport
23 --syn -m conntrack --ctstate NEW -j ACCEPT
root@dbbbed61f386:/# iptables -A FORWARD -i eth1 -p tcp --syn -m conntrack --cts
tate NEW -j ACCEPT
root@dbbbed61f386:/# iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,E
STABLISHED -j ACCEPT
root@dbbbed61f386:/# iptables -A FORWARD -p tcp -j DROP
root@dbbbed61f386:/# iptables -P FORWARD ACCEPT
root@dbbbed61f386:/#
```

图 25:

- **运行截图**

允许内部主机访问任何外部服务器。

```
root@ec2517fce4e7:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
6247d3a4955b login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

图 26:

- **比较有状态防火墙和无状态防火墙这两种不同的方法，并解释每种方法的优缺点。**

无状态防火墙的优点：它可以独立地检查每个数据包，不会遗漏特别的数据包。

无状态防火墙的缺点：独立处理它们不会考虑到数据包的上下文，因此可能导致不准确、不安全或复杂的防火墙规则。例如，如果我们想让 TCP 数据包进入我们的网络只有首先连接，我们不能实现容易使用无状态包过滤器，因为当防火墙检查每个 TCP 包，它不知道包是否属于现有的连接，除非防火墙维护一些每个连接的状态信息。

有状态防火墙的优点：因为数据包通常不是独立的；它们可能是 TCP 连接的一部分，也可能是由其他数据包触发的 ICMP 数据包。正是根据连接状态实现过滤防护，考虑上下文状态而不是独立处理数据包，更为高效和灵活。

有状态防火墙的缺点：同样不是万能的，由于其更加灵活，需要根据具体场景应用。

5.4 Task 4: Limiting Network Traffic

执行以下两条命令。

```
root@b1f1fad535d5:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute
--limit-burst 5 -j ACCEPT
root@b1f1fad535d5:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@b1f1fad535d5:/#
```

图 27:

- **解释**

1. 第一条规则是限制可以通过防火墙的数据包的数量。具体如下：

```
1 —limit avg max: average match rate: default 3/hour[Packets per
      second unless followed by
2 /sec /minute /hour /day postfixes]
3 —limit-burst number: number to match in a burst, default 5
```

2. 第二条规则是对来自 10.9.0.5 的不符合规则 1 的包进行丢弃。

- **运行截图**

可以观察到收到的 ping 回复不是连续的。可以看到对于突发包每 5 条才收到一个 echo-reply、且每分钟最多接收 10 条。

```
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.155 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.067 ms
64 bytes from 192.168.60.5: icmp_seq=18 ttl=63 time=0.069 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.068 ms
64 bytes from 192.168.60.5: icmp_seq=24 ttl=63 time=0.068 ms
64 bytes from 192.168.60.5: icmp_seq=25 ttl=63 time=0.065 ms
64 bytes from 192.168.60.5: icmp_seq=30 ttl=63 time=0.217 ms
64 bytes from 192.168.60.5: icmp_seq=31 ttl=63 time=0.070 ms
```

图 28:

- **请对有或没有第二条规则进行实验，然后解释是否需要第二条规则，以及为什么。**

需要有第二条规则否则防火墙不起限制作用。未加第二条规则效果：接收所有来自 10.9.0.5 的数据包，无论是否有第一条规则。即接收的 ping 回复序列将是连续的。所以必须加上第二条规则。

5.5 Task 5: Load Balancing

Using the nth mode (round-robin).

```
root@6f94662c0940:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080
root@6f94662c0940:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 2 --packet 0 -j DNAT --to-destination 192.168.60.6:8080
root@6f94662c0940:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode nth --every 1 --packet 0 -j DNAT --to-destination 192.168.60.7:8080
root@6f94662c0940:/#
```

图 29:

Using the random mode.

```
root@6f94662c0940:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.333 -j DNAT --to-destination 192.168.60.5:8080
root@6f94662c0940:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.333 -j DNAT --to-destination 192.168.60.6:8080
root@6f94662c0940:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statis
tic --mode random --probability 0.333 -j DNAT --to-destination 192.168.60.7:8080
root@6f94662c0940:/#
```

图 30:

- **解释：**

1. 根据不符合规则的数据包将继续在他们的旅程中运行；它们不会被修改或阻止。分别设置 3 个规则：“对于每 3 个数据包，选择数据包 0（即第一个），更改其目的地址为服务器 1”、“对于每 2 个数据包，选择数据包 0（即第一个），更改其目的地址为服务器 2”、“对于每 1 个数据包，选择数据包 0（即第一个），更改其目的地址为服务器 3”。这样每 3 个数据包一组，就能分别去往 3 个服务器。

2. 随机模式：该规则将选择一个概率为 P 的匹配数据包。为实现负载平衡规则，以便每个内部服务器获得的流量大致相同，应该使到达 3 个服务器的概率大致相同为 1/3。

- **运行截图**

Using the nth mode：依次发送 abcabc，3 个服务器分别接收到 aa、bb、cc。

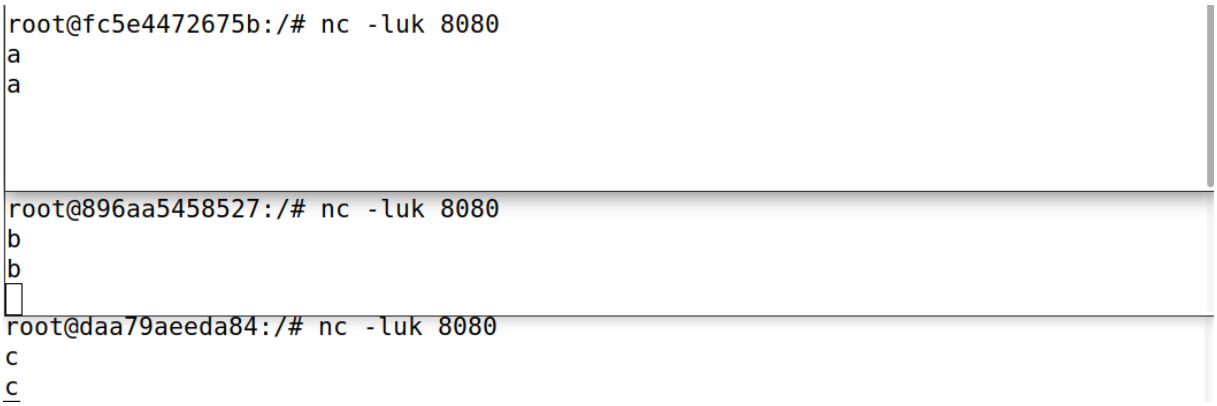

图 31:

Using the random mode：每次发送连续的 11 个 c，3 个服务器分别接收到 c 的数量为 4 个、4 个、3 个，大致相同。

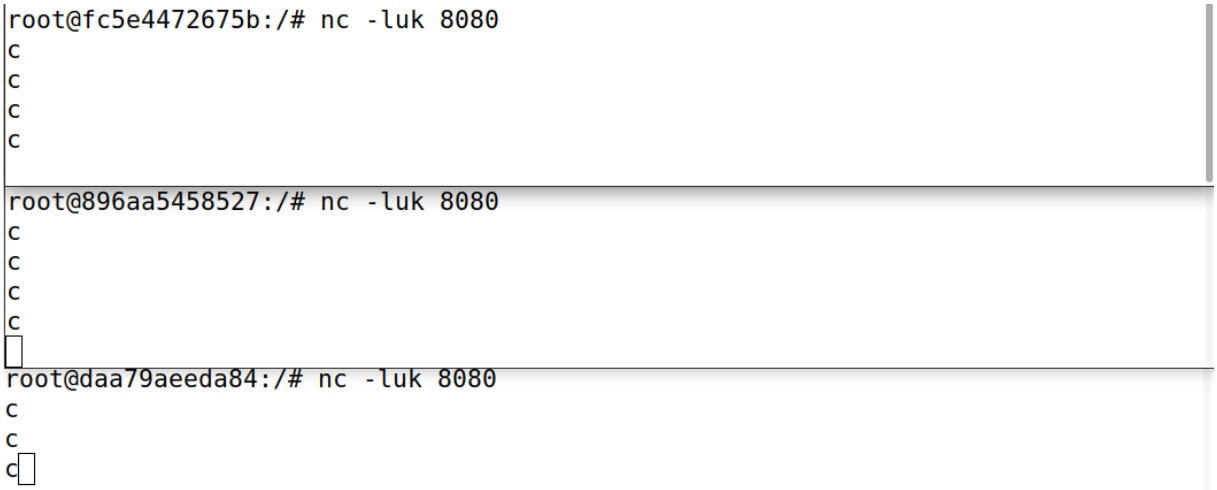

图 32:

6 第二部分：Firewall Evasion Lab

7 实验目的

有些情况下，防火墙的限制太严格，使用户不方便。例如，许多公司和学校强制执行出口过滤，从而阻止其网络内的用户接触到某些网站或互联网服务，如游戏和社交网站。有很多方法可以避开防火墙。一种典型的方法是使用隧道技术，它隐藏了网络流量的真正目的。修建隧道的方法有很多种。两种最常见的隧道传输技术分别是虚拟专用网（VPN）和端口转发。本实验室的目标是帮助学生获得这两种隧道技术的实践经验。本实验室将涵盖以下主题：

1. 防火墙渗透

2.VPN

3. 端口转发

4.SSH 隧道

8 实验原理

8.1 SSH

Secure Shell(SSH) 是由 IETF(The Internet Engineering Task Force) 制定的建立在应用层基础上的安全网络协议。它是专为远程登录会话 (甚至可以用 Windows 远程登录 Linux 服务器进行文件互传) 和其他网络服务提供安全性的协议，可有效弥补网络中的漏洞。通过 SSH，可以把所有传输的数据进行加密，也能够防止 DNS 欺骗和 IP 欺骗。还有一个额外的好处就是传输的数据是经过压缩的，所以可以加快传输的速度。目前已经成为 Linux 系统的标准配置。SSH 只是一种协议，存在多种实现，既有商业实现，也有开源实现。

SSH 之所以能够保证安全，原因在于它采用了非对称加密技术 (RSA) 加密了所有传输的数据。传统的网络服务程序，如 FTP、Pop 和 Telnet 其本质上都是不安全的；因为它们在网络上用明文传送数据、用户帐号和用户口令，很容易受到中间人（man-in-the-middle）攻击方式的攻击。就是存在另一个人或者一台机器冒充真正的服务器接收用户传给服务器的数据，然后再冒充用户把数据传给真正的服务器。

但并不是说 SSH 就是绝对安全的，因为它本身提供两种级别的验证方法：

第一种级别（基于口令的安全验证）：只要你知道自己帐号和口令，就可以登录到远程主机。所有传输的数据都会被加密，但是不能保证你正在连接的服务器就是你想连接的服务器。可能会有别的服务器在冒充真正的服务器，也就是受到“中间人攻击”这种方式的攻击。

第二种级别（基于密钥的安全验证）：你必须为自己创建一对密钥，并把公钥放在需要访问的服务器上。如果你要连接到 SSH 服务器上，客户端软件就会向服务器发出请求，请求用你的密钥进行安全验证。服务器收到请求之后，先在该服务器上你的主目录下寻找你的公钥，然后把它和你发送过来的公钥进行比较。如果两个密钥一致，服务器就用公钥加密“质询”(challenge) 并把它发送给客户端软件。客户端软件收到“质询”之后就可以用你的私钥在本地解密再把它发送给服务器完成登录。与第一种级别相比，第二种级别不仅加密所有传输的数据，也不需要在网络上传送口令，因此安全性更高，可以有效防止中间人攻击。

9 实验准备

实验环境：Ubuntu20.04

9.1 Docker 容器

基本操作简介：

```
// Aliases for the Compose commands above
$ dcbuild # Alias for: docker-compose build
$ dcup # Alias for: docker-compose up
$ dcdown # Alias for: docker-compose down

$ dockps // Alias for: docker ps --format "{{.ID}} {{.Names}}"
$ docksh <id> // Alias for: docker exec -it <id> /bin/bash
```

10 实验步骤及运行结果

10.1 Task 0: Get Familiar with the Lab Setup

请阻止另外两个网站，并将防火墙规则添加到设置文件中。

```
[12/20/22]seed@VM:~$ docksh 1868
root@18685ec3fe64:/# iptables -A FORWARD -i eth1 -d 39.156.66.14 -j DROP
root@18685ec3fe64:/# iptables -A FORWARD -i eth1 -d 211.86.56.247 -j DROP
root@18685ec3fe64:/#
```

图 33:

规则 1 阻止访问 www.baidu.com，规则 2 阻止访问 sdu.edu.cn。

- 运行截图

设置之前可以 ping 通。

```
root@04a33f2fbdf3:/# ping sdu.edu.cn
PING sdu.edu.cn (211.86.56.247) 56(84) bytes of data.
^C64 bytes from 211.86.56.247: icmp_seq=1 ttl=41 time=68.6 ms

--- sdu.edu.cn ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 68.618/68.618/68.618/0.000 ms
root@04a33f2fbdf3:/# ping www.baidu.com
PING www.a.shifen.com (39.156.66.14) 56(84) bytes of data.
^C64 bytes from 39.156.66.14: icmp_seq=1 ttl=46 time=53.2 ms

--- www.a.shifen.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 53.238/53.238/53.238/0.000 ms
root@04a33f2fbdf3:/#
```

图 34:

设置之后访问失败。

```
root@04a33f2fbdf3:/# ping www.baidu.com
PING www.a.shifen.com (39.156.66.14) 56(84) bytes of data.
^C
--- www.a.shifen.com ping statistics ---
23 packets transmitted, 0 received, 100% packet loss, time 22527ms

root@04a33f2fbdf3:/# ping sdu.edu.cn
PING sdu.edu.cn (211.86.56.247) 56(84) bytes of data.
```

图 35:

10.2 Task 1: Static Port Forwarding

请使用静态端口转发在外部网络和内部网络之间创建一个隧道，这样我们就可以直接进入 B1 上的服务器。请证明您可以从主机 A、A1 和 A2 进行这样的网络。

在主机 A 建立与主机 B 的 SSH 隧道，A、B 之间通过 SSH 服务进行通讯，通过 B 将数据以及命令转发到 B1。

```
root@217430f3c530:/# ssh -4NT -L 10.8.0.99:7000:192.168.20.5:23 seed@192.168.20.
99
seed@192.168.20.99's password:
```

```
root@b4a18fc60696:/# telnet 10.8.0.99 7000
Trying 10.8.0.99...
Connected to 10.8.0.99.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
04a33f2fbdf3 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

图 36:

上图演示了以主机 A1 为例使用该 SSH 隧道连接 B1。

- **(1) 整个过程中涉及多少 TCP 连接。您应该运行 wireshark 或 tcpdump 来捕获网络流量，然后从捕获的流量中指出所有涉及的 TCP 连接。**

主机 A 与主机 B 之间：

13	2022-12-27 07:4…	10.8.0.99	192.168.20.99	TCP	76 47396 → 22 [SYN] Seq=3518300477 Win=64240 Len=0 MSS=1460 SACK…
14	2022-12-27 07:4…	10.8.0.99	192.168.20.99	TCP	76 [TCP Out-Of-Order] 47396 → 22 [SYN] Seq=3518300477 Win=64240 …
22	2022-12-27 07:4…	10.8.0.99	192.168.20.99	TCP	76 [TCP Out-Of-Order] 47396 → 22 [SYN] Seq=3518300477 Win=64240 …
23	2022-12-27 07:4…	10.8.0.99	192.168.20.99	TCP	76 [TCP Out-Of-Order] 47396 → 22 [SYN] Seq=3518300477 Win=64240 …

图 37:

客户主机 A1 与主机 A 之间：

244	2022-12-27 07:4…	10.8.0.5	10.8.0.99	TCP	76 51538 → 7000 [SYN] Seq=1336161064 Win=64240 Len=0 MSS=1460 SA…
245	2022-12-27 07:4…	10.8.0.5	10.8.0.99	TCP	76 [TCP Out-Of-Order] 51538 → 7000 [SYN] Seq=1336161064 Win=6424…
246	2022-12-27 07:4…	10.8.0.99	10.8.0.5	TCP	76 7000 → 51538 [SYN, ACK] Seq=3879159982 Ack=1336161065 Win=651…
247	2022-12-27 07:4…	10.8.0.99	10.8.0.5	TCP	76 [TCP Out-Of-Order] 7000 → 51538 [SYN, ACK] Seq=3879159982 Ack…
248	2022-12-27 07:4…	10.8.0.5	10.8.0.99	TCP	68 51538 → 7000 [ACK] Seq=1336161065 Ack=3879159983 Win=64256 Le…
249	2022-12-27 07:4…	10.8.0.5	10.8.0.99	TCP	68 [TCP Dup ACK 248#1] 51538 → 7000 [ACK] Seq=1336161065 Ack=387…

图 38:

主机 B 与主机 B1 之间：

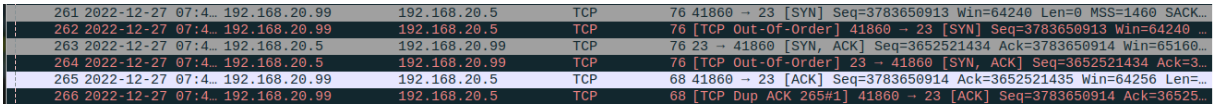

图 39:

- **(2) 为什么这个隧道能成功地帮助用户逃避在实验室设置中指定的防火墙规则？**

因为防火墙设置允许 SSH 数据包通过。

所以在主机 A 和主机 B 之间建立了一条 SSH 隧道。隧道在主机 A 的一端将接收从 telnet 客户程序发来的 TCP 数据包, 会把这些包中的数据通过隧道传输到主机 B 端。在主机 B 上运行的 SSH 会把数据放入另一个新创建的 TCP 数据包并发往 B1。防火墙只能检测到主机 A 发往主机 B 的 SSH 数据, 而检测不到主机 B 发往主机 B1 的 telnet 数据包。并且, SSH 数据是加密的, 因此防火墙无法得知其中的内容。

10.3 Task 2: Dynamic Port Forwarding

10.3.1 Task 2.1: Setting Up Dynamic Port Forwarding

请证明您可以在内部网络上使用来自主机 B、B1 和 B2 的 curl 访问所有被屏蔽的网站。

```
root@e114d3ec6c92:/# ssh -4NT -D 192.168.20.99:7200 seed@10.8.0.99
The authenticity of host '10.8.0.99 (10.8.0.99)' can't be established.
ECDSA key fingerprint is SHA256:IReOZYOQZLw8BiyJpaZ6dm+aUqIIPokQDHBnjCAK2CQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.8.0.99' (ECDSA) to the list of known hosts.
seed@10.8.0.99's password:
```

图 40:

- **运行截图**

B1 使用 curl 访问 www.example.com。

```
root@04a33f2fbdf3:/# curl --proxy socks5h://192.168.20.99:7200 www.example.com
<!doctype html>
<html>
<head>
    <title>Example Domain</title>
```

图 41:

- **(1) 哪台计算机与预期的 web 服务器建立了实际的连接？(2) 这台计算机如何知道它应该连接到哪个服务器上？**

问题（1）：主机 A。

问题（2）：设置隧道指定了一个代理选项主机 B。之后 curl 命令将把其 HTTP 请求发送给代理 B，代理 B 监听端口 x。代理将在该端口接收到的数据转发到隧道的另一端（主机 A），数据将从那里进一步转发到目标网站。

10.3.2 Task 2.2: Testing the Tunnel Using Browser

配置火狐的代理设置。要进入设置页面，我们可以在 URL 字段中键入关于：首选项，或单击“首选项”菜单项。在“常规”页面上，找到“网络设置”部分，点击“设置”按钮，将弹出一个窗口。设置代理。

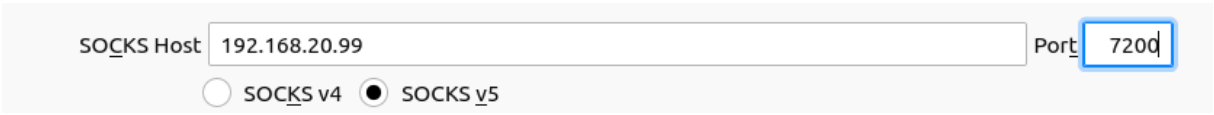

图 42:

可以在浏览器中访问 www.example.com。

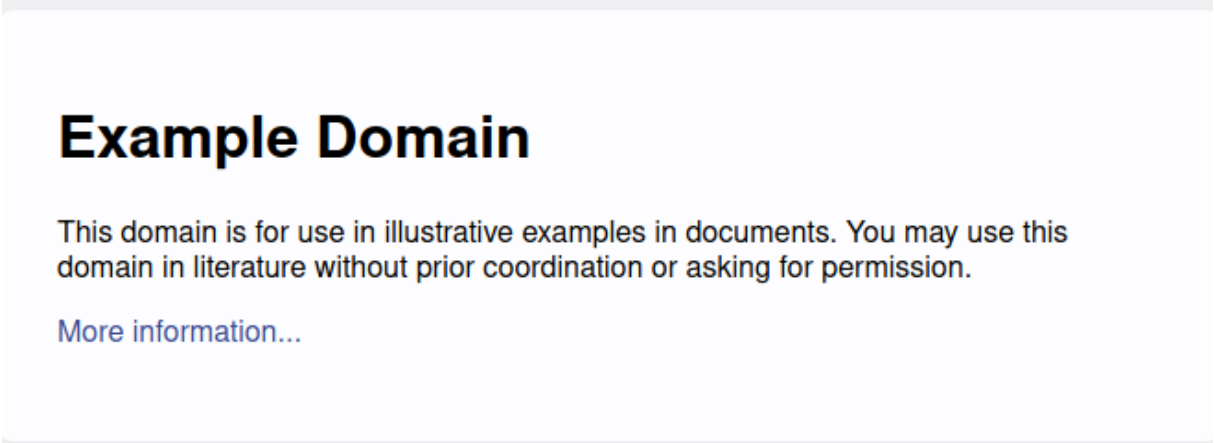

图 43:

- **(1) 指出在整个端口转发过程中所涉及的流量。(2) 打破 SSH 隧道，然后尝试浏览一个网站。描述你的观察。**

问题（1）：主机 B1 与主机 B。主机 B 与主机 A。主机 A 访问网址。
问题（2）：打破 SSH 将无法连接。

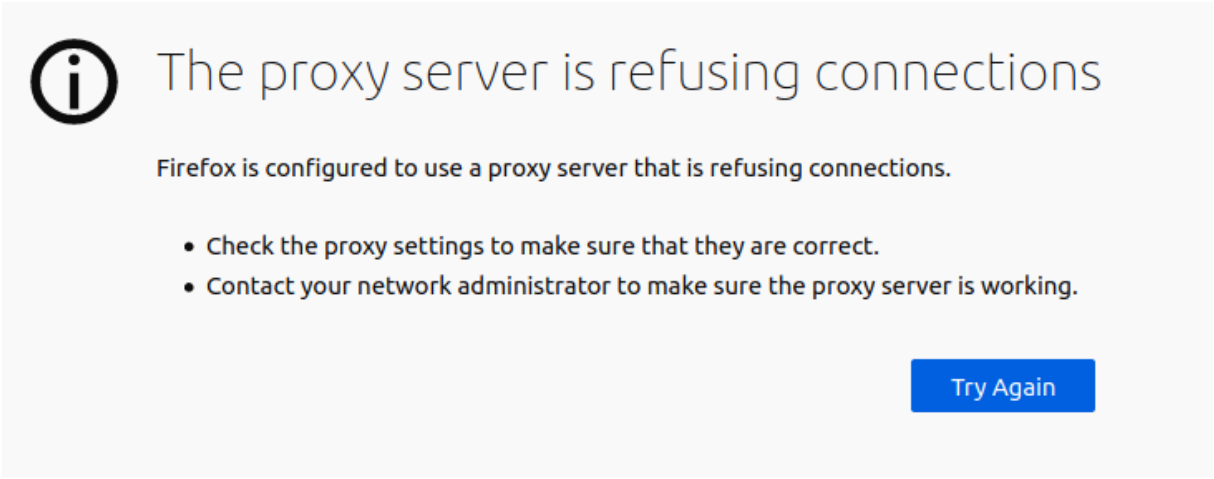

图 44:

10.3.3 Task 2.3: Writing a SOCKS Client Using Python

补全 socks 程序。B1、B2 端：

```
#!/bin/env python3
import socks
s = socks.socksocket()

# Set the proxy
s.set_proxy(socks.SOCKS5, "192.168.20.99", 7200)

# Connect to final destination via the proxy
hostname = "www.example.com"
s.connect((hostname, 80))

request = b"GET / HTTP/1.0\r\nHost: " + hostname.encode('utf-8') + b
    "\r\n\r\n"
s.sendall(request)

# Get the response
response = s.recv(2048)
while response:
  print(response.split(b"\r\n"))
  response = s.recv(2048)
```

B 端：

```
#!/bin/env python3
import socks
s = socks.socksocket()

# Set the proxy
s.set_proxy(socks.SOCKS5, "0.0.0.0", 7200)

# Connect to final destination via the proxy
hostname = "www.example.com"
s.connect((hostname, 80))

request = b"GET / HTTP/1.0\r\nHost: " + hostname.encode('utf-8') + b
    "\r\n\r\n"
s.sendall(request)

# Get the response
response = s.recv(2048)
```

```
while response:
    print(response.split(b"\r\n"))
    response = s.recv(2048)
```

- 运行截图

运行访问 www.example.com。

```
[12/20/22]seed@VM:~$ docksh 04a
root@04a33f2fbdf3:/# nano sock.py
root@04a33f2fbdf3:/# nano sock.py
root@04a33f2fbdf3:/# chmod a+x sock.py
root@04a33f2fbdf3:/# ./sock.py
```

图 45:

```
root@04a33f2fbdf3:/# ./sock.py
[b'HTTP/1.0 200 OK', b'Age: 397481', b'Cache-Control: max-age=604800', b'Content
-Type: text/html; charset=UTF-8', b'Date: Tue, 20 Dec 2022 14:05:42 GMT', b'Etag
: "3147526947+ident"', b'Expires: Tue, 27 Dec 2022 14:05:42 GMT', b'Last-Modifie
d: Thu, 17 Oct 2019 07:18:26 GMT', b'Server: ECS (sab/5798)', b'Vary: Accept-Enc
oding', b'X-Cache: HIT', b'Content-Length: 1256', b'Connection: close', b'', b'<
!doctype html>\n<html>\n<head>\n    <title>Example Domain</title>\n\n    <meta c
```

图 46:

10.4 Task 3: Virtual Private Network (VPN)

10.4.1 Task 3.1: Bypassing Ingress Firewall

请在 A 和 B 之间创建一个 VPN 隧道，其中 B 是 VPN 服务器。然后进行所有必要的配置。一旦一切都设置好了，请证明您可以从外部网络连接到 B、B1 和 B2。请捕获数据包跟踪，并解释为什么数据包没有被防火墙阻止。

```
root@405f45112449:/# ssh -w 0:0 root@192.168.20.99 -o "PermitLocalCommand=yes" -o "LocalCommand= ip addr add 192.168.53.88/24 dev tun0 && ip link set tun0 up"
-o "RemoteCommand=ip addr add 192.168.53.99/24 dev tun0 && ip link set tun0 up"
The authenticity of host '192.168.20.99 (192.168.20.99)' can't be established.
ECDSA key fingerprint is SHA256:IReOZYOQZLw8BiyJpaZ6dm+aUqIIPokQDHBnjCAK2CQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.20.99' (ECDSA) to the list of known hosts.
root@192.168.20.99's password:
```

图 47:

- 运行截图

例子：AtelnetB。（这里 telnet 192.168.53.99，因为是 VPN 接口）

```
root@bc36c0ec8fa1:/# telnet 192.168.53.99
Trying 192.168.53.99...
Connected to 192.168.53.99.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
2df8c3df47c0 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

图 48:

Wireshark 捕捉。可以查看 SSH 加密的数据包，说明 VPN 工作。

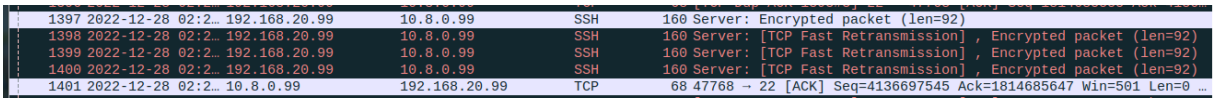

图 49:

因为使用 VPN 技术, 建立的内网到外网的隧道, IP 数据包可以通过这条隧道发送。由于隧道中的数据是加密的, 防火墙无法探知隧道中的内容, 因此不能进行过滤。

10.4.2 Task 3.2: Bypassing Egress Firewall

请在 B 和 A 之间设置一个 VPN 隧道，A 为 VPN 服务器。请证明您可以使用此 VPN 隧道从主机 B、B1 和 B2 成功到达被屏蔽的网站。请捕获数据包跟踪，并解释为什么数据包没有被防火墙阻止。

```
root@2df8c3df47c0:/# ssh -w 0:0 root@10.8.0.99 -o "PermitLocalCommand=yes" -o "L
ocalCommand= ip addr add 192.168.53.88/24 dev tun0 && ip link set tun0 up" -o "R
emoteCommand=ip addr add 192.168.53.99/24 dev tun0 && ip link set tun0 up"
root@10.8.0.99's password:
```

图 50:

VPN Server 处设置 NAT 转换。

```
[12/28/22]seed@VM:~$ docksh bc
root@bc36c0ec8fa1:/# iptables -t nat -A POSTROUTING -j MASQUERADE -o eth0
root@bc36c0ec8fa1:/#
```

图 51:

- 运行截图

先使用 B telnet 192.168.53.99，因为是 VPN 接口通过此连接 VPN 再 ping www.example.com。

```
root@bc36c0ec8fa1:/# telnet 192.168.53.99
Trying 192.168.53.99...
Connected to 192.168.53.99.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
2df8c3df47c0 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

图 52:

Wireshark 捕捉。发现 SSH 加密的数据包，说明 VPN 工作；ICMP-echo 包说明 ping 通可以连网。

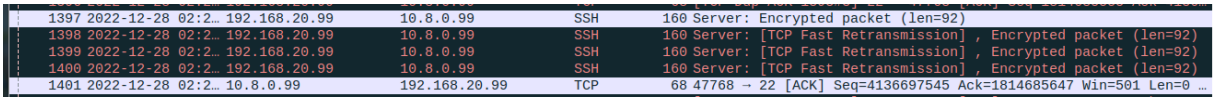

图 53:

VPN 绕过防火墙是改变经过 blocked URL 流量的路由: 去往 blocked URL 的数据包应该经过 TUN 接口, 而不是经过被防火墙阻塞的 eth1 接口。完成配置后, 可以看到数据包经过隧道成功到达 **VPN 服务器**。虽然隧道流量 (UDP 包) 仍然经过 eth1(数据包从用户机到因特网的唯一接口), 但防火墙不会阻塞这些包, 因为它们的目的地是 VPN 服务器。以下 ping 命令数据包成功绕过了防火墙, 没有出错信息, 但没有得到从 ping 返回的数据包。这可以通过在 VPN Server 中设置 NAT 解决。

10.5 Task 4: Comparing SOCKS5 Proxy and VPN

- **VPN**

绕过原理：加密无法正常通过防火墙的数据包，将其利用隧道传输；经过隧道后再对数据包解密，运输至其目的地址。

使用 IP 隧道建立一条专用的虚拟线路。通过这个隧道, 将两个私有网络组成一个虚拟专用网络。虚拟专用网络内的主机可以相互通信。

建立虚拟线路 (即隧道) 需要两台计算机, 一台是 VPN 客户端, 一台是 VPN 服务器。拥有 VPN 服务器的私有网络是主站点, 另一个是从站点, 从站点需要加入主站点来形成个虚拟专用网络。VPN 客户端和服务器的主要任务如下: (1) 相互间建立一条安全的隧道; (2) 转发由隧道一端到另一端的 IP 数据包; (3) 在隧道另一端接收到 IP 数据包后, 把数据包发送到私有网络 (物理网络) 内, 这样数据包可以到达最终的目的地。

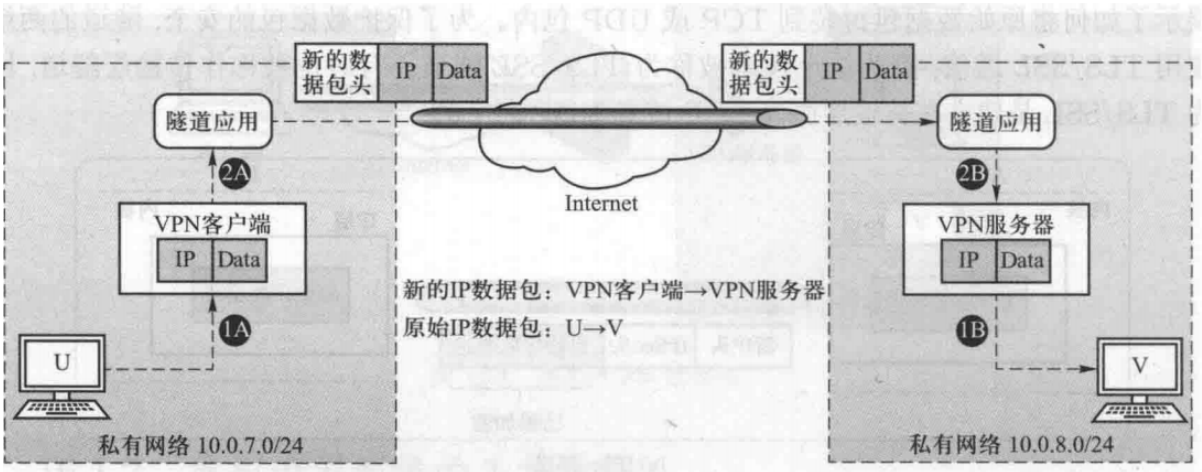

图 16.3 TLS/SSL VPN 的基本原理

图 54:

优点：功能更为强大，使用更为方便，可服务更庞大的用户群体。

缺点：严格搭建使用范围广的 VPN 站点相较 SOCK5 Proxy 更为复杂，而且需要购买 VPN 服务器节点的 IP 地址；维护费用也更高昂；同时面临着被加入黑名单的风险。

- **SOCKS5 Proxy**

绕过原理：如果一个防火墙基于数据包的目的地址进行过滤, 则可以使用 Web 代理浏览因特网, 从而绕过防火墙。只需将目的地址修改为代理服务器的地址, 让代理服务器获得网页并把结果传给即可。这样, 只要代理服务器没有被屏蔽, 就能绕过防火墙的过滤规则。

在网络中搭建一个 Web 代理, 最关键之处在于确保所有的网络流量经过代理服务器。有很多方法可以实现这个目的。可以修改系统配置来引导所有网络流量通过代理, 也可以通过修改浏览器的

网络设置使所有的 HTTP 请求都通过指定的代理, 还可以使用 iptables 直接修改 HTTP 的 IP 数据包, 将其目的 IP 地址和端口号改成代理的 IP 地址和端口号。最好的方式是不修改任何主机配置, 而把 Web 代理放在内网与外网的连接处, 这将确保所有的数据都会经过代理, 而不需要对客户端做任何修改。

优点：简单实用、容易操作。

缺点：客户机软件必须支持 SOCKS，否则不能使用 SOCKS 代理。而 SOCKS 代理不能解决前文的 telnet 问题，因为 telnet 程序不支持 SOCKS。

参考文献

[1] 杜文亮. 计算机安全导论：深度实践 [M]. 北京: 高等教育出版社,2020.4.