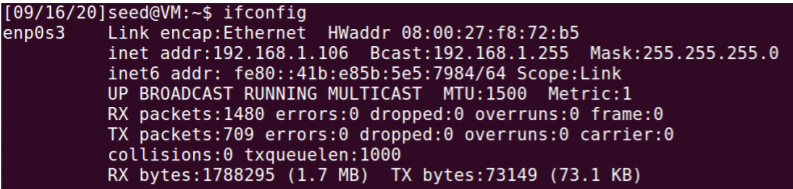
**Lab6-report**

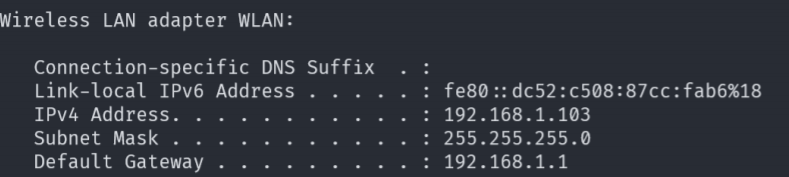
57117214 吴国铨

**Task 1: Using Firewall：**

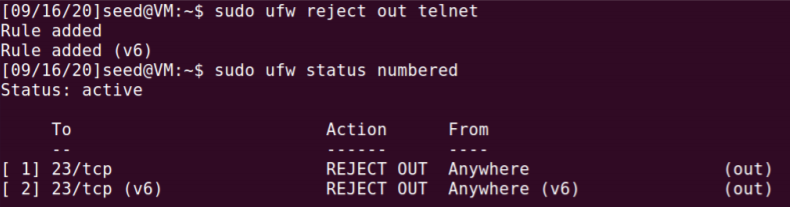
机器A:



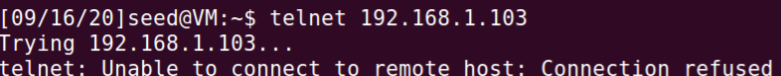
机器B：



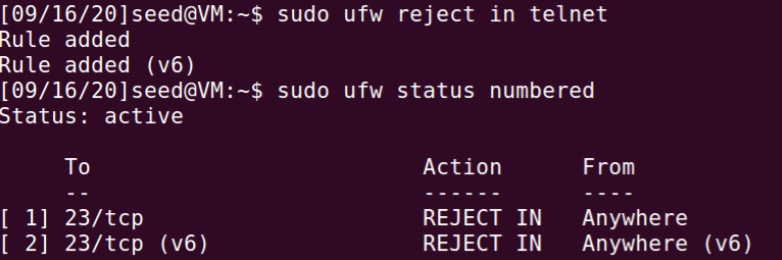
阻止A对B发起telent



检测：

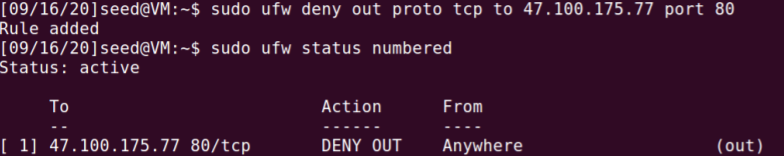


组织B对A发起telent：



阻止A访问特定外部网页：

选择ip地址为47.100.15.77的网页



使用浏览器打开，无法加载，拦截成功。

**Task 2: Implementing a Single Firewall**

使用以下代码：

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/netfilter.h>

#include <linux/netfilter\_ipv4.h>

#include <linux/ip.h>

#include <linux/tcp.h>

#include <linux/socket.h>

unsigned int outTelnetFilter(void \*priv, struct sk\_buff \*skb, const struct

nf\_hook\_state \*state) {

struct iphdr \*iph = ip\_hdr(skb);

struct tcphdr \*tcph = (void \*)iph + iph->ihl \* 4;

char ip\_src[16];

snprintf(ip\_src, 16, "%pI4", &iph->saddr);

if (strcmp(ip\_src, "192.168.1.106") == 0 && iph->protocol == IPPROTO\_TCP &&

tcph->dest == htons(23)) {

printk("DROP out telnet.\n");

return NF\_DROP;

} else {

return NF\_ACCEPT;

}

}

unsigned int inTelnetFilter(void \*priv, struct sk\_buff \*skb, const struct

nf\_hook\_state \*state) {

struct iphdr \*iph = ip\_hdr(skb);

struct tcphdr \*tcph = (void \*)iph + iph->ihl \* 4;

char ip\_dst[16];

snprintf(ip\_dst, 16, "%pI4", &iph->daddr);

if (strcmp(ip\_dst, "192.168.1.106") == 0 && iph->protocol == IPPROTO\_TCP &&

tcph->dest == htons(23)) {

printk("DROP in telnet.\n");

return NF\_DROP;

} else {

return NF\_ACCEPT;

}

}

unsigned int inSshFilter(void \*priv, struct sk\_buff \*skb, const struct

nf\_hook\_state \*state) {

struct iphdr \*iph = ip\_hdr(skb);

struct tcphdr \*tcph = (void \*)iph + iph->ihl \* 4;

char ip\_dst[16];

snprintf(ip\_dst, 16, "%pI4", &iph->daddr);

if (strcmp(ip\_dst, "192.168.1.106") == 0 && iph->protocol == IPPROTO\_TCP &&

tcph->dest == htons(22)) {

printk("DROP in ssh.\n");

return NF\_DROP;

} else {

return NF\_ACCEPT;

}

}

unsigned int evianzhangHttpsFilter(void \*priv, struct sk\_buff \*skb, const

struct nf\_hook\_state \*state) {

struct iphdr \*iph = ip\_hdr(skb);

struct tcphdr \*tcph = (void \*)iph + iph->ihl \* 4;

char ip\_src[16];

snprintf(ip\_src, 16, "%pI4", &iph->daddr);;

if (strcmp(ip\_src, "119.75.217.109") == 0 && iph->protocol == IPPROTO\_TCP &&

tcph->dest == htons(443)) {

printk("DROP connection to 119.75.217.109:443.\n");

return NF\_DROP;

} else {

return NF\_ACCEPT;

}

}

struct nf\_hook\_ops inTelnetHook;

struct nf\_hook\_ops outTelnetHook;

struct nf\_hook\_ops inSshHook;

struct nf\_hook\_ops evianzhangHook;

struct nf\_hook\_ops evianzhangHttpsHook;

static int kmodule\_init(void) {

inTelnetHook.hook = inTelnetFilter;

inTelnetHook.hooknum = NF\_INET\_POST\_ROUTING;

inTelnetHook.pf = PF\_INET;

inTelnetHook.priority = NF\_IP\_PRI\_FIRST;

outTelnetHook.hook = outTelnetFilter;

outTelnetHook.hooknum = NF\_INET\_POST\_ROUTING;

outTelnetHook.pf = PF\_INET;

outTelnetHook.priority = NF\_IP\_PRI\_FIRST;

inSshHook.hook = inSshFilter;

inSshHook.hooknum = NF\_INET\_POST\_ROUTING;

inSshHook.pf = PF\_INET;

inSshHook.priority = NF\_IP\_PRI\_FIRST;

evianzhangHook.hook = evianzhangFilter;

evianzhangHook.hooknum = NF\_INET\_POST\_ROUTING;

evianzhangHook.pf = PF\_INET;

evianzhangHook.priority = NF\_IP\_PRI\_FIRST;

evianzhangHttpsHook.hook = evianzhangHttpsFilter;

evianzhangHttpsHook.hooknum = NF\_INET\_POST\_ROUTING;

evianzhangHttpsHook.pf = PF\_INET;

evianzhangHttpsHook.priority = NF\_IP\_PRI\_FIRST;

nf\_register\_hook(&inTelnetHook);

nf\_register\_hook(&outTelnetHook);

nf\_register\_hook(&inSshHook);

nf\_register\_hook(&evianzhangHook);

nf\_register\_hook(&evianzhangHttpsHook);

return 0;

}

static void kmodule\_exit(void) {

nf\_unregister\_hook(&inTelnetHook);

nf\_unregister\_hook(&outTelnetHook);

nf\_unregister\_hook(&inSshHook);

nf\_unregister\_hook(&evianzhangHook);

nf\_unregister\_hook(&evianzhangHttpsHook);

}

module\_init(kmodule\_init);

module\_exit(kmodule\_exit);

MODULE\_LICENSE("GPL");

分别实现以下功能：

禁止本机向外发起telent

禁止外部向本机发起telent

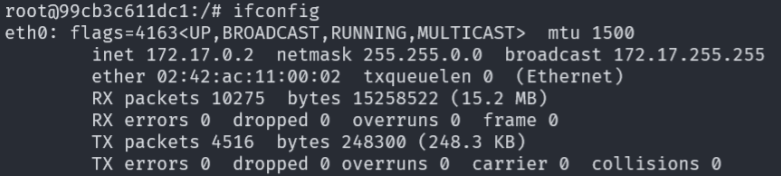
禁止外部向本机发起ssh

禁止本机访问[www.baidu.com](http://www.baidu.com)

编译后运行，载入内核，执行操作，效果与task1一样，均被阻断。

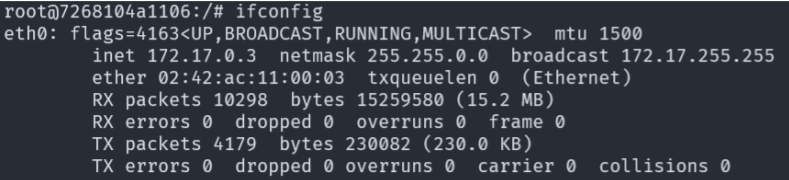
**Task 3: Evading Egress Filtering**

使用三个容器

容器A：

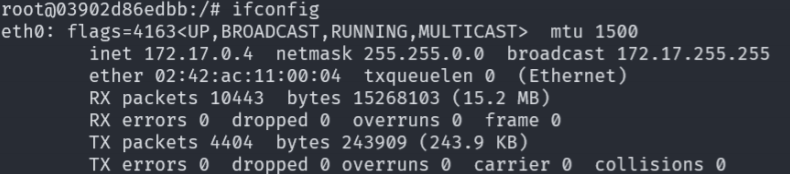
作为墙内机器

容器B：



作为墙外机器

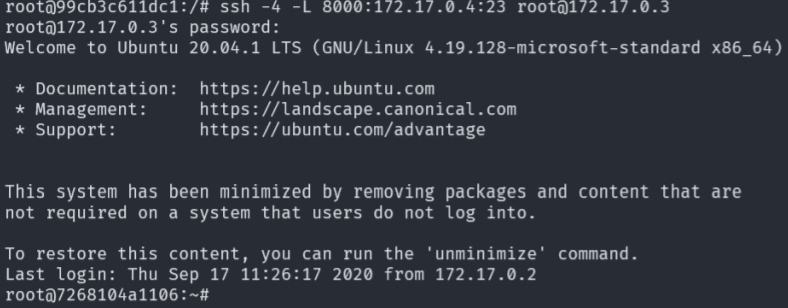
容器C：



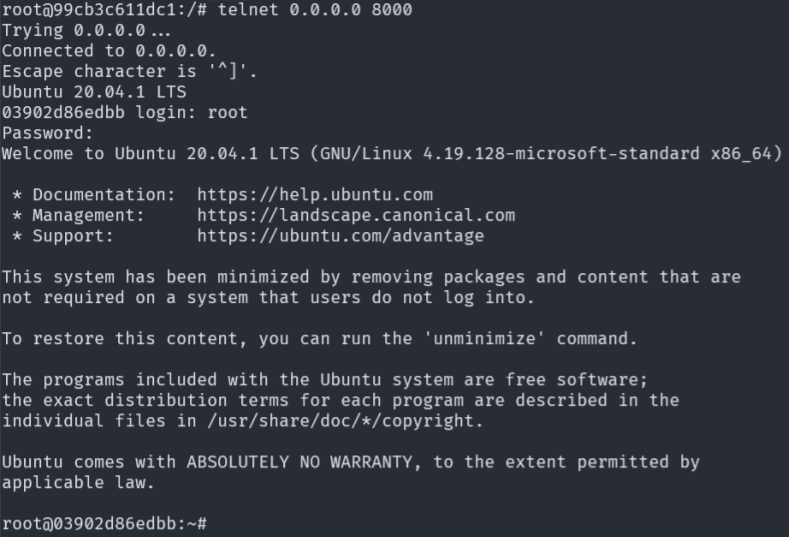
作为telnet服务器

Task 3.1: Telnet to Machine B through the firewall：

在容器A中向容器B发起ssh请求，以B为跳板访问容器C：

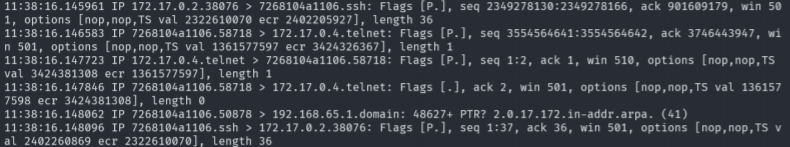


此时另外在容器A中开一个shell，对自身的8000端口发起telnet连接：



成功连接到了容器C。

在容器B中架起tcpdump可以观察到连接的情况：



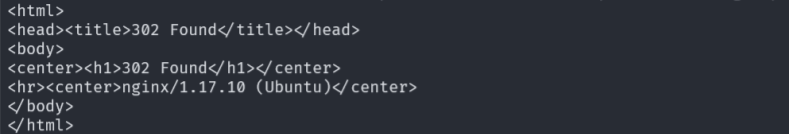
当在容器A中输入命令时，会先通过ssh将命令发送给容器B，然后容器B将命令发送给容器C的telnet，容器C的telnet返回给容器B之后，容器B再把返回结果给容器A。

Task 3.b: Connect to Facebook using SSH Tunnel

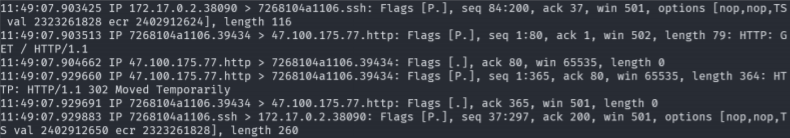
再容器A中使用

ssh -D 9000 -C [root@172.17.0.3](mailto:root@172.17.0.3)

然后再容器A的另一个shell连接中，使用9000端口作为socksv5的代理，请求连接：



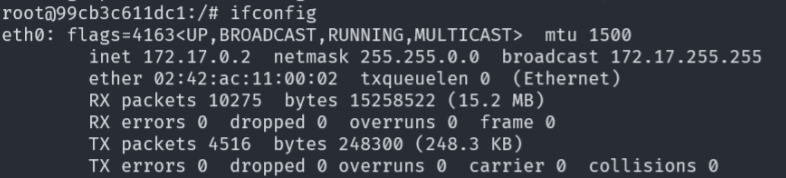
访问成功后，此时容器中显示：



将ssh关闭，再次使用curl请求数据时则会发生错误。

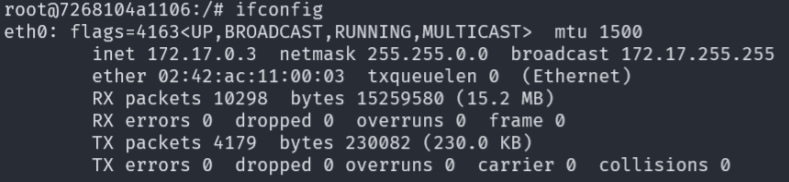
**Task 4: Evading Ingress Filtering**

容器A：



作为机器A

容器B：



作为机器B

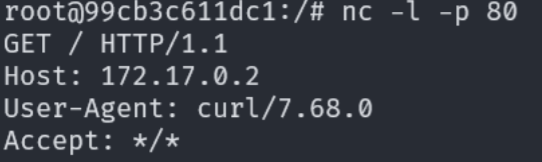
再容器A中阻断外部向内部发起的80端口和22端口的连接，同时用nc再80端口监听

此时在容器A中使用：

ssh -fCNR 172.17.0.3:2333:172.17.0.2:2334 [root@172.17.0.3](mailto:root@172.17.0.3)

对容器B的2333端口发起反向ssh隧道

在容器B中，向自身的2333端口发出TCP连接请求，在容器A中可以观察到：



说明容器A虽然阻断了外部访问80端口的请求，但是容器B通过建立反向ssh隧道，成功访问到了A内部的80端口