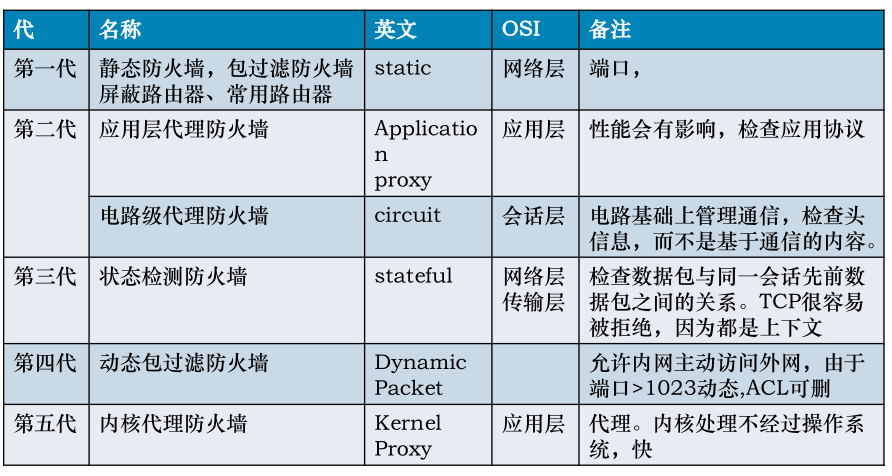
现在的软件防火墙是运行在内核模式下，通过hook进行过滤的软件。软件防火墙只会计算和分析在进程创建一个可执行的校验和。过程已经加载到内存后,它是假定保持不变,直到终止。



iptables：

iptables是个最多只能限制到第三层IP层的service。

支持ipv4，ipv6（虽然iptables的参数支持，但是我认为需要安装ip6tables才能落实）

原理：

工作在用户空间中，定义规则的工具，本身并不算是防火墙。它们定义的规则，可以让在内核空间当中的netfilter来读取，并且实现让防火墙工作。而放入内核的地方必须要是特定的位置，必须是tcp/ip的协议栈经过的地方。而这个tcp/ip协议栈必须经过的地方，可以实现读取规则的地方就叫做 netfilter.(网络过滤器)

作者一共在内核空间中选择了5个位置，

1.内核空间中：从一个网络接口进来，到另一个网络接口去的 FORWARD (转发管卡)

2.数据包从内核流入用户空间的 FORWARD (转发管卡)

3.数据包从用户空间流出的 OUTPUT(数据包出口)

4.进入/离开本机的外网接口 POSTROUTING（路由后）

5.进入/离开本机的内网接口 PREROUTING (路由前)

4.5的存在是因为我们在做NAT和DNAT的时候，目标地址转换必须在路由之前转换。所以我们必须在外网而后内网的接口处进行设置关卡。这五个位置也被称为五个钩子函数（hook functions）,也叫五个规则链。

防御手法：黑白名单。

规则的次序非常关键，谁的规则越严格，应该放的越靠前，而检查规则的时候，是按照从上往下的方式进行检查的。有一个rulenum。数数是从1开始

命令：

Table:

-t table default：filter

Chain:

-A append

-I insert [rulenum] 和-A一样后面跟着的是个chain。如果加了rulenum，那么规则就会被当做这一条插入。如果超出索引，就报错iptables: Index of insertion too big.

评价：有-I 其实也就没有必要用-A了。

-R replace rulenum 使用这个只能指定rulenum来表示被替换目标

-D delete [chain and num] 要么指定行数去删除，要么就像route del 一样自己匹配来删除一条。

-L list（-S也是种显示，-S是按照命令来显示的，相当于-L -nv）（-L 类似于wireshark格式来显示的）

-P policy 也就是默认策略

Limit:

-s source address

--sport

-d destination address

--dport 注意这里和state一样都是双-，估计因为实际上需要是-m配合的参数

-m match 如-m state --state NEW,ESTABLISHED 来指定状态

地址取反：感叹号要写在-d的前面才行，而且要用空格隔开。

iptables -A OUTPUT -o eth0 -p tcp ! -d xxx.xxx.xxx.xxx -j DROP

多个端口的配置：

1、连续端口配置-A INPUT -p tcp –-dport 21:25 -j DROP

2、使用multiport参数配置不连续端口-A INPUT -p tcp -m multiport –-dport 21:25,135:139 -j DROP

Interface：

-i eth0：从这块网卡流入的数据。流入一般用在INPUT和PREROUTING上

-o eth0：从这块网卡流出的数据。流出一般在OUTPUT和POSTROUTING上

Ptotocol:

-p protocol

--tcp-flags requires two args.匹配指定的TCP标记。有两个参数，他们都是列表，列表内部用英文的逗号作分隔符，这两个列表之间用空格分开。第一个参数指定我们要检查的标记（作用就象掩码），第二个参数指定“在第一个列表中出现过的且必须被设为1（即状态是打开的）的，所以RST打了两次。

Dispaly:

-n numeric output of addresses and ports

-v verbose -v -vv -vvv 三个对内容的显示详细程度

--line-number 显示规则在第几行

Handle:

-j target for rules

DROP 悄悄丢弃

REJECT 明示拒绝

ACCEPT 接受

REDIRECT：重定向：主要用于实现端口重定向

MASQUERADE 地址伪装，相当于自动化的snat

如下命令表示把所有10.8.0.0网段的数据包SNAT成192.168.5.3的ip然后发出去，

iptables-t nat -A POSTROUTING -s 10.8.0.0/255.255.255.0 -o eth0 -j SNAT --to-source192.168.5.3

如下命令表示把所有10.8.0.0网段的数据包SNAT成192.168.5.3/192.168.5.4/192.168.5.5等几个ip然后发出去

iptables-t nat -A POSTROUTING -s 10.8.0.0/255.255.255.0 -o eth0 -j SNAT --to-source192.168.5.3-192.168.5.5

MASQUERADE就是针对这种场景而设计的，他的作用是，从服务器的网卡上，自动获取当前ip地址来做NAT。如此配置的话，不用指定SNAT的目标ip了，不管现在eth0的出口获得了怎样的动态ip，MASQUERADE会自动读取eth0现在的ip地址然后做SNAT出去，这样就实现了很好的动态SNAT地址转换。比如下边的命令：

iptables-t nat -A POSTROUTING -s 10.8.0.0/255.255.255.0 -o eth0 -j MASQUERADE

iptables -L -n -v #查看当前定义了的规则 没想到这条命令不能将Lnv这样子合起来，不过nv这两个是可以合起来的

iptables -A OUTPUT -p tcp --tcp-flags RST RST -d IP -j DROP

封掉单个IP：

iptables -I INPUT -s 59.151.119.180 -j DROP

封IP段的命令是

iptables -I INPUT -s 211.1.0.0/16 -j DROP

iptables -I INPUT -s 211.2.0.0/16 -j DROP

iptables -I INPUT -s 211.3.0.0/16 -j DROP

使用iptables举例，该命令将所有端口重定向到443端口（该端口上有multi/handler监听处理）。技巧之一就是把SSH监听的端口设为一个比65535大的值。

iptables -t nat -A PREROUTING -p tcp –dport 1:65534 -j REDIRECT –to-ports 443

使用测试：

iptables -A INPUT -s 192.168.1.1 -j DROP 这命令成+功组织了curl的返回

iptables -D INPUT 1

root@kali:~# iptables -S

-P INPUT ACCEPT

-P FORWARD ACCEPT

-P OUTPUT ACCEPT

-A OUTPUT -s 192.168.1.1/32 -j DROP

root@kali:~# 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

DROP all -- hellovega.cn anywhere #这个显示不太好我应该加个-n

root@kali:~# iptables -D OUTPUT -s 192.168.1.1 -j DROP

root@kali:~# iptables -I OUTPUT -d 192.168.1.1 -j DROP

root@kali:~# iptables -I OUTPUT -d 192.168.1.1 -j REJECT

root@kali:~# iptables -S

-P INPUT ACCEPT

-P FORWARD ACCEPT

-P OUTPUT ACCEPT

-A OUTPUT -d 192.168.1.1/32 -j REJECT --reject-with icmp-port-unreachable

-A OUTPUT -d 192.168.1.1/32 -j DROP

root@kali:~# iptables -D OUTPUT -s 192.168.1.1 #服气

iptables: Bad rule (does a matching rule exist in that chain?).

root@kali:~# iptables -D OUTPUT -s 192.168.1.1 -j DROP #要这么详细（-S命令查看）

root@kali:~# iptables -R OUTPUT 1 -j DROP

root@kali:~# ping 192.168.1.1 -c 1

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.

From 192.168.1.141 icmp\_seq=1 Destination Port Unreachable

--- 192.168.1.1 ping statistics ---

0 packets transmitted, 0 received, +1 errors

root@kali:~# curl 192.168.1.1

curl: (7) Failed to connect to 192.168.1.1 port 80: 拒绝连接

练习题1：

只要是来自于172.16.0.0/16网段的都允许访问我本机的172.16.100.1的SSHD服务

既然采用白订单策略，那么：

先设置允许22：

定义进来的： iptables -t filter -A INPUT -s 172.16.0.0/16 -d 172.16.100.1 -p tcp --dport 22 -j ACCEPT

定义出去的： iptables -t filter -A OUTPUT -s 172.16.100.1 -d 172.16.0.0/16 -p tcp --dport 22 -j ACCEPT

再将默认策略改成DROP:

iptables -P INPUT DROP

iptables -P OUTPUT DROP

iptables -P FORWARD DROP

练习题2：

假如我们允许自己ping别人，但是别人ping自己ping不通如何实现呢？

分析：对于ping这个协议，进来的为8（ping），出去的为0(响应).我们为了达到目的，需要8出去,允许0进来

iptables -A INPUT -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT

iptables -A OUTPUT -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT

在出去的端口上：iptables -A OUTPUT -p icmp --icmp-type 8 -j ACCEPT

在进来的端口上：iptables -A INPUT -p icmp --icmp-type 0 -j ACCEPT

iptables -P INPUT DROP

iptables -P OUTPUT DROP

iptables -P FORWARD DROP

连接状态：适用于那些必须要公开的，但是又要做好防护的

对于整个TCP协议来讲，它是一个有连接的协议，三次握手中，第一次握手，我们就叫NEW连接

完整的三次握手后建立的连接，叫做已建立的连接（ESTABLISHED）

还有一种状态，比较诡异的，比如：SYN=1 ACK=1 RST=1,对于这种我们无法识别的，我们都称之为INVALID无法识别的

还有第四种，FTP这种古老的拥有的特征，每个端口都是独立的，21号和20号端口都是一去一回，他们之间是有关系的，这种关系我们称之为RELATED。

所以我们的状态一共有四种：

NEW

ESTABLISHED

RELATED

INVALID

扩展练习题1：

进来的拒绝出去的允许，进来的只允许ESTABLISHED进来，出去只允许ESTABLISHED出去。默认规则都使用拒绝

iptables -L -n --line-number ：查看之前的规则位于第几行

改写INPUT

iptables -R INPUT 2 -s 172.16.0.0/16 -d 172.16.100.1 -p tcp --dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT

iptables -R OUTPUT 1 -m state --state NEW，ESTABLISHED -j ACCEPT #你要是没有NEW的haul，是到不了ESTABLISHED

NAT：S、D是对外部机器来讲

SNAT基于原地址的转换：基于原地址的转换一般用在我们的许多内网用户通过一个外网的口上网的时候，这时我们将我们内网的地址转换为一个外网的IP，我们就可以实现连接其他外网IP的功能。

比如我们现在要将所有192.168.10.0网段的IP在经过的时候全都转换成172.16.100.1这个假设出来的外网地址：

iptables -t nat -A POSTROUTING -s 192.168.10.0/24 -j SNAT --to-source 172.16.100.1

DNAT目标地址转换：对于目标地址转换，数据流向是从外向内的，外面的是客户端，里面的是服务器端通过目标地址转换，我们可以让外面的ip通过我们对外的外网ip来访问我们服务器不同的服务器，而我们的服务却放在内网服务器的不同的服务器上。

iptables -t nat -A PREROUTING -d 192.168.10.18 -p tcp --dport 80 -j DNAT --todestination 172.16.100.2

储存与恢复：

-t table default filter

iptables-save > iptables.back

iptables-restore < iptables.back

root@kali:~# iptables-apply #也是种备份 默认文件是/etc/network/iptables.up.rules，可以使用-w来指定其他文件

Error: rulesfile not readable: /etc/network/iptables.up.rules

iptables-xml：iptables-xml — Convert iptables-save format to XML

root@kali:~# iptables-xml iptables.back > iptables.back.xml

不管设置为DROP还是REJETC，基于tcpnmap扫描都是filter。基于udp扫描都是closed。从原理上来讲，tcp应该是REJECT，udp应该是DROP。

我的路由器防护：

先使用nmap扫描tcp端口的开放：

PORT STATE SERVICE

22/tcp open ssh

23/tcp open telnet

53/tcp open domain

80/tcp open http

139/tcp open netbios-ssn

445/tcp open microsoft-ds

MAC Address: F0:B4:29:D0:F5:E2 (Xiaomi Communications)

没想到PandoraBox会开启139和445端口。首先查看了一下发现iptables里面已经有很多规则了（根据LAN口和WAN口设置了它们的Chain），所以我就插入到第一条。

udp扫描：

PORT STATE SERVICE

53/udp open|filtered domain

67/udp open|filtered dhcps

111/udp open|filtered rpcbind

137/udp open netbios-ns

138/udp open|filtered netbios-dgm

139/udp open|filtered netbios-ssn

445/udp open|filtered microsoft-ds

2049/udp open|filtered nfs

25240/udp open|filtered unknown

32777/udp open|filtered sometimes-rpc18

32778/udp open|filtered sometimes-rpc20

32779/udp open|filtered sometimes-rpc22

32780/udp open|filtered sometimes-rpc24

47624/udp open|filtered directplaysrvr

48078/udp open|filtered unknown

MAC Address: F0:B4:29:D0:F5:E2 (Xiaomi Communications)

对udp采取措施-j和采取措施DROP是一样的，都什么都不会返回，于是认为是closed。

[root@hellovega:/root]#iptables -I INPUT 1 ! -s 192.168.0.0/16 -p tcp -m multiport --dport 22,23,53,80,139,445 -j REJECT

[root@hellovega:/root]#iptables -I INPUT 2 ! -s 192.168.0.0/16 -p udp -m multiport --dports 53,67,111,137,138,139,445,2049,25240,32777,32778,32779,32780,47624,48078 -j DROP

[root@hellovega:/root]#iptables-save > iptables.back

设置完以后再扫描:

tcp的139端口与445端口还是open，这应该说明iptables对它们无效。他们在内核上更快？不知道。其他本来开着的宽口都变成filter。

我在路由器里面设置了ssh访问只能从lan端口是起到效果了，因为我nmap -p 22 虚拟端口的ip返回是closed。

升级：

iptables -I INPUT 1 ! -s 192.168.0.0/16 -p tcp -m multiport --dport 1:1000 -j REJECT

不进行udp iptables过滤

话说我直接iptables -L显示会很慢。但是-L -nv或者-S就一下出来了。可能是涉及到有个转换。

服务防护：

PandoraBox基于BusyBox

/usr/sbin/dropbear -F -P /var/run/dropbear.1.pid -p 192.168.1.1:22 -p fd23:522d:e38a::1:22 -K 300

/usr/sbin/nmbd -D

/usr/sbin/smbd -D

不支持ps -aux，直接ps，因为ps -ww被别名过成ps了。

通过ping -I pppoe-vwan10 -c 1 -W 2 -q 115.239.210.27。还有114.114.115.115，115.239.211.112这几个地址来验证自己虚拟端口是否需要重拨。

于是killall nmbd & killall smbd

odhcp6c是Openwrt中DHCPv6-client ，还有个odhcpd。dhcp服务端是靠dnsmasq，TCP、UDP53端口dns，UDP67端口dhcp。

如果有人连接ssh：

tcp 0 0 192.168.1.1:22 0.0.0.0:\* LISTEN 2265/dropbear

tcp 0 0 192.168.1.1:22 192.168.1.147:56815 ESTABLISHED 27884/dropbear

同一个端口既是在LISTEN也是在ESTABLISHED

ipv6支持：

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 2556/uhttpd

tcp 0 0 :::80 :::\* LISTEN 2556/uhttpd

udp 0 0 :::546 :::\* 6167/odhcp6c

udp 0 0 :::546 :::\* 6168/odhcp6c

udp 0 0 :::546 :::\* 6153/odhcp6c

udp 0 0 :::546 :::\* 6159/odhcp6c

udp 0 0 :::546 :::\* 6157/odhcp6c....

我的OpenCV防护：

443和444端口虽然说是ss的连接端口，一直处于LISTEN，如果我有用到翻墙的话，就会开启一些新的socket处于ESTABLISHED，同时原连接端口既是ESTABLISHED也是LISTEN。

netstat -pantu会出现一堆使用udp端口的py脚本

route [add|del] [-net|-host] target [netmask Nm] [gw Gw] [[dev] If]

其中：

add : 添加一条路由规则

del : 删除一条路由规则

-net : 目的地址是一个网络

-host : 目的地址是一个主机

target : 目的网络或主机

netmask : 目的地址的网络掩码

gw : 路由数据包通过的网关

dev : 为路由指定的网络接口

删除的时候指定的规则越详细删除的范围就会越小，比如route delete IP就会删除所有dest为这个IP的规则。

<https://blog.csdn.net/vevenlcf/article/details/48026965> Route命令详细介绍与包转发

包转发临时有效命令：echo 1 > /proc/sys/net/ipv4/ip\_forward

------------------------------------------------------------------------------------------------------------------------

bypass windows 4firewall ：

进程访问网络的时候，windows会向用户询问是否允许公网与私有网络，这就是在防火墙上进行设置。哦对，也就是说还按照进程名字来进行不同的策略。

bypass方法：runtime process infection（运行时过程感染，最常见的手段就是dll注入）

runtime process infection：

将代码注入到IE或者其他已经允许上网的进程中，CreateRemoteThread执行我们的代码，CreateProcess并带有SW\_HIDE来开启被注入进程，记得调用WaitForInputIdle()。