

使用 Netfilter 搭建一个简单的防火墙

理解一项技术的最佳方式是动手实现它。数据包过滤器只能在内核中实现，因此代码需要运行在内核中，也就意味着需要修改内核。Linux 提供了两项重要技术，使得无需重新编译整个内核就能实现包过滤器。它们分别是 Netfilter(netfilter.org, 2017) 和可加载内核模块 (loadable kernel modules)。

Netfilter 在数据包经过的路径上埋放了一些钩子 (hook)，它们位于内核中。自行编写的函数可以通过内核模块放进内核，并挂在那些钩子上。当数据包到达某个钩子上时，挂在钩子上的函数就会被调用，可以在函数中对数据包进行审查和过滤，并告诉 Netfilter 如何处理。

(一) 可加载内核模块实现

(1) 编写可加载内核模块

Linux 内核是模块化设计的，因此只有很小一部分被加载进内核。如果需要扩展内核的功能，可以将这些功能设计成内核模块，动态地载入内核。例如，为了支持一个新硬件，可以将它的设备驱动程序作为一个内核模块加载进内核。内核模块是可以根据实时需求载入或卸载的代码块。只有具有 root 权限或者 CAP_SYS_MODULE 能力的进程才能够载入或者卸载内核模块。每个内核模块都具有两个入口点，一个入口点用于载入模块，另一个用于卸载模块。宏 `module_init()` 和 `module_exit()` 是用来指定这两个入口点的函数。`module_init()` 指定的函数在模块载入时被调用，用来初始化。`module_exit()` 指定的函数在模块卸载时被调用，用来进行清理工作。下面的示例代码 (示例代码 1) 展示了如何编写一个可加载内核模块。

示例代码 1 一个简单的可加载内核模块 (kMod.c)

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

static int kmodule_init(void)
{
    printk(KERN_INFO "Initializing this module\n");
    return 0;
}

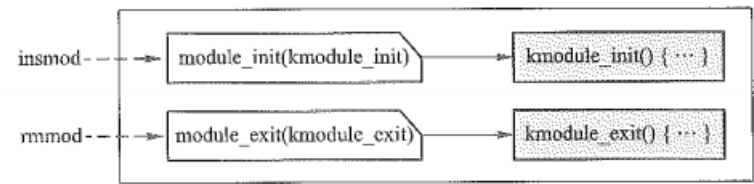
static void kmodule_exit(void)
{
    printk(KERN_INFO "module cleanup\n");
}

module_init(kmodule_init); //向内核注册模块提供新功能      (1)
module_exit(kmodule_exit); //注销由模块提供的所有功能      (2)

MODULE_LICENSE("GPL"); //告诉内核该模块具有 GNU 公共许可证
```

仔细阅读示例代码。行 (1) 和行 (2) 处定义的宏 `module_init()` 和宏 `module_exit()` 分

别指向插入、移除内核模块所需执行的函数（如下图所示）。这两个函数只是简单的打印了一些信息。在内核中打印信息不能用 `printf()` 函数把信息答应到屏幕上，而使用 `printk()` 函数打印内容至内核日志缓存。



(2) 编译内核模块

编译一个内核模块最简单最有效的方式莫过于使用 `makefile`。下面是一个编译可加载内核模块的 `makefile` 样例：

```
CONFIG_MODULE_SIG=n
obj-m := kMod.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

为了编译可加载内核模块，必须有一个和本机内核相对应的头文件及配置文件。每个 Linux 发行版本都提供访问这些文件的途径，这些文件多数保存在 `/usr/src` 目录中。

在上面的 `makefile` 中，参数 `M` 表明一个外部模块将要被编译，以及模块生成后应该放在什么位置。选项 `-C` 用于指定内核库文件的目录。当执行 `makefile` 中的 `make` 命令时，`make` 进程会切换到指定的目录并在完成时切换回来。编译结果如下：

```
-virtual-machine:~/桌面/可加载内核模块$ sudo make
make -C /lib/modules/5.3.0-64-generic/build M=/home/桌面/可加载内核模块 modules
#编译模块
make[1]: 进入目录"/usr/src/linux-headers-5.3.0-64-generic"
CC [M] /home/桌面/可加载内核模块/kMod.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/桌面/可加载内核模块/kMod.mod.o
LD [M] /home/桌面/可加载内核模块/kMod.ko
make[1]: 离开目录"/usr/src/linux-headers-5.3.0-64-generic"
```

`ls` 命令查看编译结果。

```
-virtual-machine:~/桌面/可加载内核模块$ ls
kMod.c  kMod.mod  kMod.mod.o  Makefile  Module.symvers
kMod.ko kMod.mod.c  kMod.o      modules.order
```

注：在编译时可能遇到如下图所示的问题与解决方案。

“/bin/sh: 1: flex: not found”	sudo apt-get install flex
“/bin/sh: 1: bison: not found”	sudo apt-get install bison

```

root@virtual-machine: ~/桌面/可加载内核模块$ sudo make
[sudo] 的密码:
make -C /lib/modules/5.3.0-64-generic/build SUBDIRS= modules #编译模块
make[1]: 进入目录“/usr/src/linux-headers-5.3.0-64-generic”
LEX scripts/kconfig/lexer.lex.c
/bin/sh: 1: flex: not found
make[3]: *** [scripts/Makefile.lib:196: scripts/kconfig/lexer.lex.c] 错误 127
make[2]: *** [Makefile:589: synconfig] 错误 2
make[1]: *** [Makefile:701: include/config/auto.conf.cmd] 错误 2
make[1]: 离开目录“/usr/src/linux-headers-5.3.0-64-generic”
make: *** [Makefile:3: all] 错误 2

```

```

root@virtual-machine: ~/桌面/可加载内核模块$ sudo make
make -C /lib/modules/5.3.0-64-generic/build SUBDIRS= modules #编译模块
make[1]: 进入目录“/usr/src/linux-headers-5.3.0-64-generic”
LEX scripts/kconfig/lexer.lex.c
YACC scripts/kconfig/parser.tab.h
/bin/sh: 1: bison: not found
make[3]: *** [scripts/Makefile.lib:210: scripts/kconfig/parser.tab.h] 错误 127
make[2]: *** [Makefile:589: synconfig] 错误 2
make[1]: *** [Makefile:701: include/config/auto.conf.cmd] 错误 2
make[1]: 离开目录“/usr/src/linux-headers-5.3.0-64-generic”
make: *** [Makefile:3: all] 错误 2

```

(3) 安装内核模块

内核模块生成后，可以使用 `insmod`、`rmmod` 和 `lsmod` 等命令管理内核模块。用户也可以选择使用更加复杂的 `modprobe` 命令。示例如下。

```

//把内核模块载入内核
$ sudo insmod kMod.ko

//查看 kMod 模块是否已载入内核（也可直接 lsmod）
$ lsmod | grep kMod
kMod                12453      0

//把内核模块从内核中卸载
$ sudo rmmod kMod

```

为证实模块已成功执行，可以检查模块的输出。在内核模块程序中，当模块被加载和卸载时会分别向内核日志缓冲区输出一些信息。Linux 提供了 `dmesg` 命令用于检查内核日志缓冲区。运行该命令，就能找到这些模块输出的信息。示例如下。

```

$ dmesg //也可使用 dmesg > 输出文件名，例 dmesg > 1.txt
[65368.235725] Initializing this module
[65499.594389] Module cleanup

```

使用实例如下所示。

- ① 使用 `insmod` 命令将内核模块载入内核，并使用 `lsmod` 命令查看模块是否已载入内核。

```
-virtual-machine:~/桌面/可加载内核模块$ sudo insmod kMod.ko
[sudo] 的密码:
lt@lt-virtual-machine:~/桌面/可加载内核模块$ lsmod
Module                Size  Used by
kMod                  16384  0
nls_utf8              16384  1
isofs                 49152  1
intel_rapl_msr        20480  0
intel_rapl_common     24576  1 intel_rapl_msr
snd_ens1371           28672  2
snd_ac97_codec        131072  1 snd_ens1371
crct10dif_pclmul      16384  1
crc32_pclmul          16384  0
ghash_clmulni_intel   16384  0
gameport              20480  1 snd_ens1371
ac97_bus              16384  1 snd_ac97_codec
aesni_intel           372736  0
snd_pcm               106496  2 snd_ac97_codec,snd_ens1371
vmw_balloon           24576  0
```

② 使用 `dmesg` 命令检查内核日志缓冲区，查看输出。

```
[ 162.850474] Initializing this module
[ 611.031673] module cleanup
[ 679.317122] Initializing this module
```

③ 使用 `rmmod` 命令将内核模块从内核中卸载，再用 `lsmod` 命令查看原模块已被删除。

```
lt@lt-virtual-machine:~/桌面/可加载内核模块$ sudo rmmod kMod
lt@lt-virtual-machine:~/桌面/可加载内核模块$ lsmod
Module                Size  Used by
nls_utf8              16384  1
isofs                 49152  1
intel_rapl_msr        20480  0
intel_rapl_common     24576  1 intel_rapl_msr
snd_ens1371           28672  2
snd_ac97_codec        131072  1 snd_ens1371
crct10dif_pclmul      16384  1
crc32_pclmul          16384  0
ghash_clmulni_intel   16384  0
gameport              20480  1 snd_ens1371
ac97_bus              16384  1 snd_ac97_codec
```

(二) 使用 Netfilter 搭建一个简单的防火墙

(1) 前置知识

Linux 内核通过 Netfilter 的钩子函数提供了强大的数据包处理及过滤框架。在 Linux 中，每个协议栈都在数据包经过的路径上定义了一系列钩子点。开发者使用内核模块将他们定义的函数“钩”到这些钩子点上，当每个数据包均到达钩子点时，协议栈会调用 Netfilter 框架 (Russell et al, 2002)，查找是否有内核模块挂在这个钩子点，如果有，这些模块注册的函数会被调用，这时它们得以分析或处理数据包。最后，它们将返回对数据包的处理结果。对数据包的处理结果有 5 种，说明如下。

(1) `NF_ACCEPT`。允许数据包通过。

(2) `NF_DROP`。丢弃数据包，这样数据包将不会在网络协议栈中继续传输。

(3) `NF_QUEUE`。使用 `nf_queue` 机制将数据包传递到用户空间处理。

(4) `NF_STOLEN`。告知 Netfilter 框架忽略这个数据包。这个操作主要是把数据包进一步的处理工作从 Netfilter 转交给模块。数据包在内核内表中仍然是有效的。典型用途是把分片

的数据包存储起来，以便在一个上下文中分析它们。

(5) NF_REPEAT。请求 Netfilter 框架再次调用这个模块。

Netfilter 为 IPv4 定义了 5 种钩子函数。数据包在网络协议栈中的传输情况（与防火墙相关）如下图所示。更详细的内容可参考相关资料（Rio et al, 2004）。

(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。

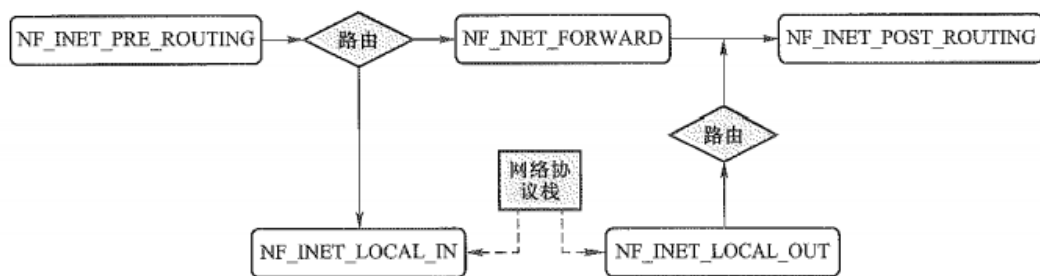
(2) NF_INET_LOCAL_IN。数据包要进行路由判决，以决定需要被转发还是发往主机。前一种情况下，数据包将前往转发路径，而后一种情况下，数据包将通过这个钩子点，之后被发送到网络协议栈，并最终被主机接受。

(3) NF_INET_FORWARD。需要被转发的数据包会达到这个钩子点。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。



(2) 下面使用 Netfilter 框架实现一个简单的数据包过滤器。目标是阻止所有发往端口号 23 的 TCP 数据包，也就是阻止用户使用 telnet 连接到其他计算机。首先编写一个回调函数，在这个函数中实现过滤功能，然后将这个函数挂到一个 Netfilter 的钩子上。示例代码（示例代码 2）如下。

示例代码 2 数据包过滤器(telnetFilter.c)

```
unsigned int telnetFilter(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 packet to %d %d %d %d\n",
            ((unsigned char *)&iph->daddr)[0],
            ((unsigned char *)&iph->daddr)[1],
            ((unsigned char *)&iph->daddr)[2],
            ((unsigned char *)&iph->daddr)[3]);
        return NF_DROP;
    }
}
```

```

    else
    {
        return NF_ACCEPT;
    }
}

```

当回调函数 `telnetFilter()` 被调用时，它会得到整个数据包（参数 `skb` 指向的就是这个数据包）。该函数检查数据包的 TCP 头部，确定目的端口号是否为 23，如果是，就丢弃该数据包，并输出一些信息到内核日志中，如果不是，数据包将被允许通行。

下面把上述回调函数挂到 Netfilter 的钩子上，可以使用 `NF_INET_LOCAL_OUT` 或 `NF_INET_POST_ROUTING`，这两个钩子点都在数据包的流出路径上。在下面的示例代码中，使用 `NF_INET_POST_ROUTING`。

示例代码 3 用 Netfilter 的内核模块屏蔽 telnet (`telnetFilter.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>

static struct nf_hook_ops telnetFilterHook;

/* The implementation of the telnet,filter function is omitted here;
   filter was shown easlier in Listing 2. */

int setUpFilter(void)
{
    printk(KERN_INFO "Registering a Telnet filter.\n");
    telnetFilterHook.hook = telnetFilter;
    telnetFilterHook.hooknum = NF_INET_POST_ROUTING;
    telnetFilterHook.pf = PF_INET;
    telnetFilterHook.priority = NF_IP_PRI_FIRST;

    // Register the hook
    nf_register_net_hook(&init_net,&telnetFilterHook);
    return 0;
}

void removeFilter(void)
{
    printk(KERN_INFO "Telnetfilter is being removed.\n");
    nf_unregister_net_hook(&init_net,&telnetFilterHook);
}

```

```
module_init(setUpFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");
```

上面的代码构建了一个 `nf_hook_ops` 结构，其中包括回调函数（`telnetFilter`）、钩子号（`NF_INET_POST_ROUTING`）、IPv4 协议族（`PF_INET`）和优先级 `priority` 等关键信息。接下来使用 `nf_register_hook()` 函数把 `telnetFilter` 挂到指定的钩子上。其余代码已在前面讨论过。

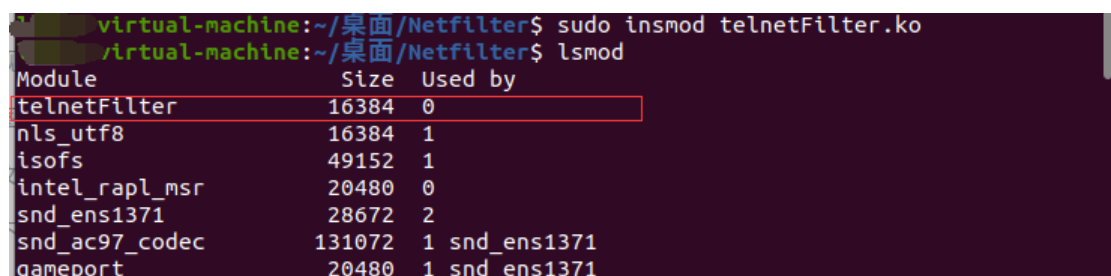
使用 `insmod` 加载模块到内核后，可以尝试用 `telnet` 命令登录其他计算机。从下面的执行结果来看，`telnet` 登录失败，使用 `dmesg` 命令能看到所有的 `telnet` 数据包均被丢弃。如果移除该内核模块，`telnet` 登录能够成功。注意：务必在试验后移除这个内核模块，否则后续实验会受到影响。代码结果示例如下。

```
//把模块加载到内核
$ sudo insmod telnetFilter.ko
$ telnet 10.0.2.5
Tring 10.0.2.5...
telnet: Unable to connet to remote host: Connection timed out //登录失败
$ dmesg

[1166456.149046] Registering a Telnet filter.
[1166535.962316] Dropping telnet packet to 10.0.2.5
[1166536.958065] Dropping telnet packet to 10.0.2.5
//把模块从内核中卸载
$ sudo rmmod telnetFilter
$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 12.04.2 LTS
ubuntu login: //登录成功
```

结果示例：

① 先将源程序“`telnetFilter.c`”通过 `makefile` 编译，再使用 `insmod` 命令将模块加载入内核，用 `lsmod` 命令查看结果。



```
virtual-machine:~/桌面/Netfilter$ sudo insmod telnetFilter.ko
virtual-machine:~/桌面/Netfilter$ lsmod
Module              Size  Used by
telnetFilter         16384  0
nls_utf8             16384  1
isofs                49152  1
intel_rapl_msr       20480  0
snd_ens1371          28672  2
snd_ac97_codec       131072  1 snd_ens1371
gameport             20480  1 snd_ens1371
```

② 选择另一台主机查看 `ip` 地址（这里的示例选择的是 `ip` 地址为 `192.168.153.131` 的 `kali` 虚拟机），然后使用 `telnet` 命令登录，发现登录失败（`telnet: Unable to connet to remote host: Connection timed out`）。

```

root@kali:~# sudo ifconfig
[sudo] 的密码:
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.153.131 netmask 255.255.255.0 broadcast 192.168.153.255
    inet6 fe80::20c:29ff:fe03:40fd prefixlen 64 scopeid 0<20<link>
    ether 00:0c:29:03:40:fd txqueuelen 1000 (Ethernet)
    RX packets 152 bytes 10130 (9.8 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 31 bytes 3101 (3.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0<10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 24 bytes 1564 (1.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 24 bytes 1564 (1.5 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

```

root@kali:~# telnet 192.168.153.131
Trying 192.168.153.131...
telnet: Unable to connect to remote host: Connection timed out

```

③ 使用 `dmesg` 命令检查内核日志缓冲区，查看输出。

```

[ 2265.347787] Registering a Telnet filter.
[ 2308.451237] Dropping telnet packet to 192 168 153 131
[ 2309.473549] Dropping telnet packet to 192 168 153 131
[ 2311.489668] Dropping telnet packet to 192 168 153 131
[ 2315.585906] Dropping telnet packet to 192 168 153 131
[ 2323.777554] Dropping telnet packet to 192 168 153 131
[ 2339.905218] Dropping telnet packet to 192 168 153 131
[ 2372.673443] Dropping telnet packet to 192 168 153 131

```

④ 使用 `rmmod` 命令将内核模块从内核中卸载，再用 `lsmod` 命令查看原模块已被删除。

```

root@kali:~# sudo rmmod telnetFilter
[sudo] 的密码:
root@kali:~# lsmod
Module                  Size  Used by
nls_utf8                16384  1
isofs                   49152  1
intel_rapl_msr          20480  0
snd_ens1371             28672  2
snd_ac97_codec          131072  1 snd_ens1371
gameport                20480  1 snd_ens1371

```

⑤ 模块卸载后，再使用 `telnet` 命令登录 kali。登录成功。

```

root@kali:~# telnet 192.168.153.131
Trying 192.168.153.131...
Connected to 192.168.153.131.
Escape character is '^]'.
Kali GNU/Linux Rolling
kali login:
Password:
Linux kali 5.5.0-kali2-amd64 #1 SMP Debian 5.5.17-1kali1 (2020-04-21) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@kali:~#

```