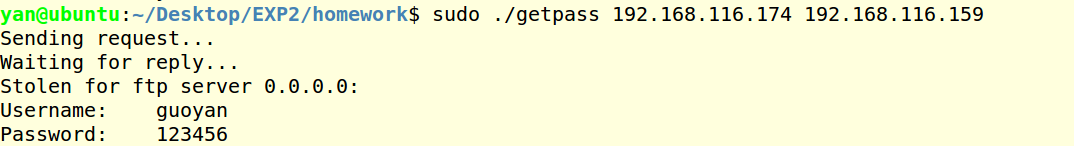
**本实验分为两部分。**

**第一部分是iptables练习**

1. **通过在iptables中添加相应的规则，使用iptables实现代理功能。**
2. **添加规则，在拒绝其它所有流量的情况下，开放ftp服务。**

**第二部分是基于netfilter实现对明文传输的HTTP用户名和密码。效果如下：**



**实验背景：**

Loadable Kernel Module是可加载内核模块，通过 Linux 内核模块（LKM）可以在运行时动态地更改 Linux，*可动态更改* 是指可以将新的功能加载到内核、从内核去除某个功能，甚至添加使用其他 LKM 的新 LKM。LKM运行在内核，因此具有极大的权限，也即所谓的LKM编程所能实现的功能“仅受想象力的限制”，对于安全而言，是一把双刃剑。（参考<http://www.ibm.com/developerworks/cn/linux/l-lkm/>）

Netfilter是从Linux 2.4开始内核的一个子系统，架构就是在整个网络流程的若干位置放置了一些检测点（HOOK），而在每个检测点上登记了一些处理函数进行处理（如包过滤，NAT等，甚至可以是 用户自定义的功能）。

　　IP层的五个HOOK点的位置如下所示

　　[1]:NF\_IP\_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）， 目的地址转换在此点进行；

　　[2]:NF\_IP\_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；

　　[3]:NF\_IP\_FORWARD：要转发的包通过此检测点，FORWORD包过滤在此点进行；

　　[4]:NF\_IP\_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；

　　[5]:NF\_IP\_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。

更多参考资料：

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》

**实验目的：**

本次编程实验的目的是为了练习LKM编程和更进一步深入了解iptables和netfilter。程序的主要功能在在受害者电脑上安装一个kernel module，此模块的功能是暗中记载受害者的mail.ustc.edu.cn（或任何一个使用明文传输用户名和密码的网站）的用户名或密码，并且在接收到攻击者发送过来的特殊数据包，本例中是一个特殊的ICMP数据包之后，将受害者该网站的用户名和密码发送给攻击者。

本实验参考代码：http://phrack.org/issues/61/13.html

**实验步骤：**

1. LKM版hello world

内核模块必须有至少两个函数，init\_module()和cleanup\_module()，分别表示起始和结束（也可以使用宏定义，module\_init或module\_exit指定函数担任起始和结束函数，并不一定是这两个函数名）。每一个内核模块都必须包括linux/module.h。

/\*

\* hello-1.c - The simplest kernel module.

\*/

#include <linux/module.h> /\* Needed by all modules \*/

#include <linux/kernel.h> /\* Needed for KERN\_INFO \*/

int init\_module(void)

{

printk(KERN\_INFO "Hello world 1.\n");

/\*

\* A non 0 return means init\_module failed; module can't be loaded.

\*/

return 0;

}

void cleanup\_module(void)

{

printk(KERN\_INFO "Goodbye world 1.\n");

}

编译内核模块的makefile如下：

obj-m += hello-1.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

内核模块编译成功之后会产生.ko文件，使用insmod命令可以将该内核模块加载到内核，使用rmmod命令可以将内核模块卸载。

对于本例，可以在insmod 和rmmod之后，使用dmesg | tail 命令查看这个hello world程序在内核中的输出。

更多LKM的例子和资料可以参考：

<http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html>

1. Netfilter编程

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。

Hook 调用的时机

NF\_INET\_PRE\_ROUTING 在完整性校验之后，选路确定之前

NF\_INET\_LOCAL\_IN 在选路确定之后，且数据包的目的是本地主机

NF\_INET\_FORWARD 目的地是其它主机地数据包

NF\_INET\_LOCAL\_OUT 来自本机进程的数据包在其离开本地主机的过程中

NF\_INET\_POST\_ROUTING 在数据包离开本地主机“上线”之前

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：

返回值 含义

NF\_DROP 丢弃该数据包

NF\_ACCEPT 保留该数据包

NF\_STOLEN 忘掉该数据包

NF\_QUEUE 将该数据包插入到用户空间

NF\_REPEAT 再次调用该hook函数

使用netfilter最重要的工作是围绕着nf\_hook\_ops数据结构，nf\_hook\_ops数据结构在linux/netfilter.h中定义，该数据结构的定义如下：

struct nf\_hook\_ops {

struct list\_head list;

/\* 此下的值由程序员填充 \*/

nf\_hookfn \*hook;

int pf;

int hooknum;

/\* Hook以升序的优先级排序 \*/

int priority;

};

也即在代码中需要指明，本次的对包进行处理的函数是哪一个nf\_hookfn \*hook（自己实现）；协议簇pf是什么（PF\_INET）；hooknum挂在哪个钩子上int hooknum（NF\_IP\_PRE\_ROUTING等五个值中的一个）；以及指定钩子函数执行顺序的优先级int priority，可用的值在linux/netfilter\_ipv4.h的nf\_ip\_hook\_priorities枚举中定义，在实验中我们使用NF\_IP\_PRI\_FIRST即可。

以上四个需要程序员自己赋值的参数中，唯一一个需要额外注意的便是nf\_hookfn \*hook，因为它的函数原型的参数比较多…

unsigned int hook\_func(unsigned int hooknum,

struct sk\_buff \*skb,

const struct net\_device \*in,

const struct net\_device \*out,

int(okfn)(struct sk\_buff \*)){

}

这五个参数其实很容易理解，如果大家在之前的实验中已经对数据包的结构有足够多的理解，skb便是存储数据包的一段内存，in便是数据的进入接口，out便是包的输出接口，第一个参数hooknum是指定钩子，最后一个参数暂时意义不明-\_-。我们在程序中需要使用的也主要是中间的三个参数。

在了解了以上内容后，我们可以写出一个很简单的防火墙内核模块，丢掉所有到达的包：

Drop\_all.c

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/netfilter.h>

#include <linux/netfilter\_ipv4.h>

static struct nf\_hook\_ops nfho;

unsigned int hook\_func(unsigned int hooknum,struct sk\_buff \*skb,const struct net\_device \*in,const struct net\_device \*out,int(okfn)(struct sk\_buff \*)){

return NF\_DROP;

}

int init\_module(){

nfho.hook = hook\_func;

nfho.hooknum = NF\_INET\_PRE\_ROUTING;

nfho.pf = PF\_INET;

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

nf\_register\_hook(&nfho);

return 0;

}

void cleanup\_module(void){

nf\_unregister\_hook(&nfho);

}

1. 接下来可以做进一步细致的处理，也即针对包的内容做判断。下面一段代码也即drop\_tcp.c，是丢弃所有到达的tcp包。为了判断到达的是否是tcp包，需要对skb中的数据进行处理。此时，首先需要对skb的内容进行类型强制转化，变成我们比较熟悉的ip包头结构。使用ip\_hdr这一函数可以达到这一目的。

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/netfilter.h>

#include <linux/netfilter\_ipv4.h>

#include <linux/ip.h>

#include <linux/tcp.h>

static struct nf\_hook\_ops nfho;

struct sk\_buff \* skbuff;

struct iphdr \* iph;

unsigned int hook\_func(unsigned int hooknum,struct sk\_buff \*skb,const struct net\_device \*in,const struct net\_device \*out,int(okfn)(struct sk\_buff \*)){

skbuff = skb;

iph = ip\_hdr(skbuff);

if(iph->protocol == 6){

printk("drop tcp!\n");

return NF\_DROP;

}

else

return NF\_ACCEPT;

}

int init\_module(){

nfho.hook = hook\_func;

nfho.hooknum = NF\_INET\_PRE\_ROUTING;

nfho.pf = PF\_INET;

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

nf\_register\_hook(&nfho);

return 0;

}

void cleanup\_module(void){

nf\_unregister\_hook(&nfho);

}

关于makefile可以参考<http://blog.csdn.net/liang13664759/article/details/1771246>；

简化版:

什么是makefile？或许很多Winodws的程序员都不知道这个东西，因为那些Windows的IDE都为你做了这个工作，但我觉得要作一个好的和professional的程序员，makefile还是要懂。这就好像现在有这么多的HTML的编辑器，但如果你想成为一个专业人士，你还是要了解HTML的标识的含义。特别在Unix下的软件编译，你就不能不自己写makefile了，会不会写makefile，从一个侧面说明了一个人是否具备完成大型工程的能力。

makefile带来的好处就是——“自动化编译”，一旦写好，只需要一个make命令，整个工程完全自动编译，极大的提高了软件开发的效率。make是一个命令工具，是一个解释makefile中指令的命令工具，一般来说，大多数的IDE都有这个命令，比如：Delphi的make，Visual C++的nmake，Linux下GNU的make。可见，makefile都成为了一种在工程方面的编译方法。

一般来说，无论是C、C++、还是pas，首先要把源文件编译成中间代码文件，在Windows下也就是 .obj 文件，UNIX下是 .o 文件，即 Object File，这个动作叫做编译（compile）。然后再把大量的Object File合成执行文件，这个动作叫作链接（link）。

编译时，编译器需要的是语法的正确，函数与变量的声明的正确。对于后者，通常是你需要告诉编译器头文件的所在位置（头文件中应该只是声明，而定义应该放在C/C++文件中），只要所有的语法正确，编译器就可以编译出中间目标文件。一般来说，每个源文件都应该对应于一个中间目标文件（O文件或是OBJ文件）。链接时，主要是链接函数和全局变量，所以，我们可以使用这些中间目标文件（O文件或是OBJ文件）来链接我们的应用程序。链接器并不管函数所在的源文件，只管函数的中间目标文件（Object File）。

总结一下，源文件首先会生成中间目标文件，再由中间目标文件生成执行文件。在编译时，编译器只检测程序语法，和函数、变量是否被声明。如果函数未被声明，编译器会给出一个警告，但可以生成Object File。而在链接程序时，链接器会在所有的Object File中找寻函数的实现，如果找不到，那到就会报链接错误码（Linker Error）。

make命令执行时，需要一个 Makefile 文件，以告诉make命令需要怎么样的去编译和链接程序。

首先，我们用一个示例来说明Makefile的书写规则。这个示例来源于GNU的make使用手册，在这个示例中，我们的工程有3个C文件，和2个头文件，我们要写一个Makefile来告诉make命令如何编译和链接这几个文件。规则是：

1）如果这个工程没有编译过，那么我们的所有C文件都要编译并被链接。

2）如果这个工程的某几个C文件被修改，那么我们只编译被修改的C文件，并链接目标程序。

3）如果这个工程的头文件被改变了，那么我们需要编译引用了这几个头文件的C文件，并链接目标程序。

只要Makefile写得够好，所有的这一切，我们只用一个make命令就可以完成，make命令会自动智能地根据当前的文件修改的情况来确定哪些文件需要重编译，从而自己编译所需要的文件和链接目标程序。

在讲述这个Makefile之前，还是让我们先来粗略地看一看Makefile的规则。  
  
target ... : prerequisites ...  
command  
...  
...  
  
target也就是一个目标文件，可以是Object File，也可以是执行文件。还可以是一个标签（Label），对于标签这种特性，在后续的“伪目标”章节中会有叙述。  
prerequisites就是，要生成那个target所需要的文件或是目标。  
command也就是make需要执行的命令。（任意的Shell命令）  
这是一个文件的依赖关系，也就是说，target这一个或多个的目标文件依赖于prerequisites中的文件，其生成规则定义在command中。说白一点就是说，prerequisites中如果有一个以上的文件比target文件要新的话，command所定义的命令就会被执行。这就是Makefile的规则。也就是Makefile中最核心的内容。