基于内核 2.6 的 Linux 包过滤型防火墙的设计与实现①

Design and Implementation of Linux Packet Inspection Firewall Based on Kernel V2.6

[1]朱　冲　[2]杨　俊　[1]张向利　[1]谢志恒[1]
（1. 桂林电子科技大学信息与通信学院 广西 桂林 541004;
2. 桂林威达仪器仪表办公设备集团公司 广西 桂林 541002）

摘　要: 防火墙在网络信息安全中扮演着重要的角色，本文在深入分析 linux 内核 2.6.23 相关数据结构及函数的基础上，结合 linux 内核模块编程技术和防火墙设计的一般理论，设计并实现了一款基于内核版本 2.6.23 的 linux 包过滤型防火墙模块，实践证明，该防火墙模块具有较高的实用价值。

关键词: 信息安全 linux 包过滤型防火墙 数据结构 内核编程 模块

网络防火墙是一种用来加强网络之间访问控制的技术，它是不同网络或网络安全域之间信息的唯一出入口，利用防火墙按给定的安全政策控制（允许、拒绝、监测）出入网络的信息流，可以实现对内部网络操作环境的保护。防火墙技术根据其防范的方式及侧重点的不同可分为以下四种基本类型：包过滤型防火墙、代理型防火墙、状态监视型防火墙和复合式防火墙。

包过滤型防火墙依据网络中的分包传输技术，对两个或者多个网络之间传输的数据包按照一定的安全策略进行检查，以决定网络之间的通信是否被允许。包过滤型防火墙优点是简单实用，实现成本低，能够以较小的代价在一定程度上保证系统的安全。

代理型防火墙也叫代理服务器，它 通过对应用层进行侦测和扫描，可以有效地防治基于应用层的侵入和病毒。但此类防火墙对用户不透明且不能改进底层协议的安全性，代理速度慢，系统管理也较为复杂。

状态监视型防火墙采用了一个在网关上执行网络安全策略的软件引擎，称之为检测模块。检测模块通过抽取相关状态信息的方法对网络通信的各层实施监测和分析，并能根据分析结果有效地判断出各层中的非法侵入。状态监测型防火墙比包过滤型和代理型防火墙安全性高，但其实现成本高，配置非常复杂，且给网络速度带来了一定影响。

复合式防火墙通常是代理服务器和状态分析技术的组合，此类防火墙能够智能化地做出安全控制和流量控制的决策，对网络内外完全透明，能够提供高性能的服务和灵活的适应性，但同时也具有了代理型和状态监视型的一些缺点，如管理较为复杂，影响系统整体性能。

为了给用户提供可靠的安全保证，linux 操作系统在较高版本（如 2.4 以上）的内核中提供了功能较为全面、通用自带防火墙 netfilter。利用 iptables 可以对 netfilter 进行配置，但 netfilter、iptables 的安装、配置都较为复杂，要求用户有较多的操作系统知识及内核配置经验，且一些不必要的功能的引入也导致系统资源消耗过多，所以本文在深入分析 firewall_ops、sk_buff 等数据结构的基础上，通过对数据包协议、端口进行分类检查，设计出一款轻巧、高效的包过滤型防火墙模块供内核调用。关于包过滤型防火墙的设计，目前方法众多，文献 1 实现了一个包过滤型防火墙应用程序，文献 2 则在较低版本（如 2.4 以下）内核中实现了一个防火墙模块，然而随着内核版本的变迁，内核模块的设计方法及相关函数的应用已不尽相同，本文重点讨论如

① 基金项目：广西自然科学基金项目（桂科自 0832246）；桂林电子科技大学和广西教育厅学科软环境建设项目（编号 D200328）；广西青年科学基金项目（桂科青 0832084）

何在 linux 高版本内核(如 2.6)中实现包过滤型防火墙模块。

1 防火墙模块总体设计

本文研究的是防火墙模块的软硬件开发环境和防火墙实现的功能及开发步骤,而对 linux 内核编程环境的构建及相关工具的使用不做介绍,读者可阅读文献 3 熟悉内核模块设计的相关理论及方法。

1.1 模块开发环境

Linux 防火墙模块设计不同于传统的 linux 应用程序设计,但完全可以采用 linux 内核模块通用的设计方法,现在给出本文的模块开发环境:

硬件环境:i386

软件环境:kernel V2.6.23.1-42.fc8、GCC 4.1.10

1.2 模块功能介绍

基于内核 2.6 的 linux 包过滤型防火墙模块重点实现了以下几个功能:

1.2.1 包过滤功能

该防火墙模块在 linux 下实现了全程动态包过滤功能,通过分析数据包的地址、协议、端口,对任何网络连接的当前状态进行访问控制,从而提高系统的性能和安全性。

1.2.2 日志功能

该防火墙模块配备了日志记录系统,可将包过滤基本信息导入到日志文件,或者从屏幕实时打印输出。

1.2.3 扩展性和可操作性

该防火墙模块框架具有良好的扩展性,只需添加相关函数便能实现功能扩展,而且使用简单的命令即可实现防火墙的加载或卸载,使之容易操作和管理。

1.3 模块运行流程

为了让读者更好地理解系统内核加载防火墙模块的过程,文章给出了一个模块运行框图(图 1),其中虚线框内为内核模块加载流程图,该流程图各个阶段的功能及相关函数调用关系介绍如下。

1.3.1 加载模块

模块被加载时调用用户定义的初始化函数,初始化过程包括:防火墙模块向内核登记注册、相关变量赋初值,其中模块的登记注册通过调用防火墙模块注册函数完成。

1.3.2 包过滤

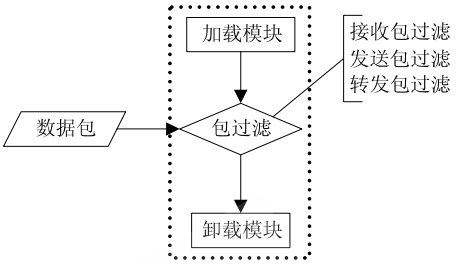

图 1 模块运行框图

模块加载完成后,若内核接收到数据包,便调用用户编制的包过滤函数进行包过滤检查。根据内核对不同数据包的处理策略不同,本文将包过滤函数分为三类:接收包过滤函数,发送包过滤函数,转发包过滤函数。

1.3.3 卸载模块

模块被卸载时调用用户定义的移除函数,移除过程包括:防火墙模块向内核反注册、相关资源释放,其中模块的反注册通过调用防火墙模块反注册函数完成。

2 防火墙模块函数设计

通过上文对防火墙模块的总体分析,并结合防火墙及内核模块设计的相关工程学知识,文章给出防火墙模块的三类框架函数:模块初始化函数与移除函数、模块注册函数与反注册函数、模块包过滤函数。下面利用内核模块编程技术,给出各框架函数的具体实现。

2.1 模块初始化函数与移除函数

linux 内核使用以下宏来实现模块的加载和卸载:

```
module_init(init_func);
module_exit(exit_func);
```

/*初始化、移除函数分别为 init_func、exit_func*/

用户只需提供初始化函数或移除函数被其调用即可,现给出本实验的初始化函数 init()伪代码如下,

```
int init(void) {
  if(register_firewall(PF_INET,&fw_ops)!=0)
  /*注册函数完成模块向内核的登记注册*/
  …… …}/*其它初始化动作及必要的打印信息*/
```

同理,给出本实验的移除函数 cleanup()伪代码

如下，

```
void cleanup(void) {
  unregister_firewall(PF_INET,&fw_ops);
  /* 反注册函数完成模块向内核的反注册 */
  … …} /* 相关资源释放及必要的打印信息 */
```

2.2 模块注册函数与反注册函数

要使内核自动调用用户编写的防火墙模块，防火墙模块必须先向内核登记注册，但内核版本的变迁导致防火墙模块注册和反注册函数在不同版本的内核中位置不同，且在内核 2.6.10 以后的版本中，作者未曾发现有内核提供这两个函数，因此，在本实验中，作者自己构造了一对防火墙注册和反注册函数，考虑到与过去内核版本函数接口的一致性，本文直接借鉴了内核 2.1.25 中防火墙注册和反注册函数的定义方式。下面将给出注册函数和反注册函数的原型，并对其参数做必要说明。

注册函数原型：extern int register_firewall(int pf, struct firewall_ops *fw)；

反注册函数原型：extern int unregister_firewall(int pf, struct firewall_ops *fw)；

返回值 0 代表成功，小于 0 代表失败。

参数说明：

协议标志 pf：取 2 代表 Ipv4 协议，取 4 代表 IPX 协议，取 10 代表 Ipv6 协议。

调用函数结构体 firewall_ops 声明如下：

```
struct firewall_ops {
struct firewall_ops *next;/* 下一个防火墙模块 */
int (*fw_forward)(struct firewall_ops *this, int pf, struct net_device *dev, void *phdr,
void *arg, struct sk_buff **pskb);/* 包过滤函数 */
int (*fw_input)(struct firewall_ops *this, int pf, struct net_device *dev, void *phdr,
void *arg, struct sk_buff **pskb);/* 包过滤函数 */
int (*fw_output)(struct firewall_ops *this, int pf,struct net_device *dev, void *phdr,
void *arg, struct sk_buff **pskb);/* 包过滤函数 */
int fw_pf;/* 协议标志 */
int fw_priority;/* 防火墙优先级 */ };
```

在内核注册函数中，要求设置模块的访问方式。本文通过调用头文件 semaphore.h 中的宏__SEMAPHORE_INITIALIZER 初始化一把互斥锁 firewall_sem，实现了对模块的互斥访问。在防火墙模块的注册函数和反注册函数中，用到了头文件 semaphore.h 中若干原子操作函数，在此不再一一列举。

2.3 模块包过滤函数

包过滤型防火墙利用包过滤函数对数据包的地址、协议、端口进行分析，从而实现不同网络之间的访问控制，考虑到所有针对数据包的操作均建立在控制结构 sk_buff 的基础之上。故本小节首先介绍了 sk_buff 结构中部分成员的含义，然后才给出包过滤函数的实现方法。

2.3.1 sk_buff 结构

网络协议栈是一个有层次的软件结构，层与层之间通过预定的接口传递网络报文。网络报文包含了在协议栈各层使用到的各种信息，且长度不一定。在 linux 的实现中，每个网络报文都有一个叫做 sk_buff 的控制结构，内核 2.6.23 的 sk_buff 结构中相关成员的含义如下：

```
struct sk_buff {
struct sk_buff *next;
struct sk_buff *prev;/* 构造双向链表 */
__be16protocol;/* 报文协议类型 */
sk_buff_data_ttransport_header;
sk_buff_data_tnetwork_header;
sk_buff_data_tmac_header;
/* 以上分别为传输头部、网络层头部、链路层头部指针 */
… …};
```

分析 sk_buff 可知，网络报文是以双向链表的方式在存储空间中组织的，网络报文依次存放的是链路层头部、网络层头部、传输层头部、传输层数据，本文通过操作 sk_buff 中的各分层头部地址指针，实现了相关协议、端口的过滤检查。

2.3.2 fw_input() 函数

在 3.2 小节的函数调用结构体 firewall_ops 中，已

经定义了三个防火墙包过滤函数：fw_input()、fw_output()、fw_forward()。其中 fw_input 完成对接收报文的检查，fw_output 完成对发送报文的检查，fw_forward 完成对转发报文的检查。

三个函数的实现机理一样，故此处仅给出 fw_input 的伪代码实现，该伪代码对地址解析协议（ARP）、反地址解析协议（RARP）、网络控制报文协议（ICMP）、传输控制协议（TCP）进行了分类检查，若允许某种协议连接，则返回 FW_ACCEPT，若禁止某种协议连接，则返回 FW_REJECT，其中 FW_ACCEPT 和 FW_REJECT 定义在头文件 ip. h 及 tcp. h 中。作者在示例的同时力图编写一个实用的 fw_input() 函数，所以允许了 ARP、RARP、ICMP 协议连接，对 TCP 协议只允许通过 8080 端口进行通信，而用户可以根据自己的需求自由设置允许协议及其允许端口，本文对接收报文的具体处理策略可见代码注释。

```
int fw_input(struct firewall_ops * this,int pf,struct device * dev,void * phdr,void * arg,struct sk_buff * * pskb) {
    #define PERMIT_PORT 8080 / * 允许访问端口定义 * /
    … … / * 其它初始化工作 * /
    if(skb - > protocol = = htons(ETH_P_ARP))
      return FW_ACCEPT; / * 允许地址解析协议报 * /
    if(skb - > protocol = = htons(ETH_P_RARP))
      return FW_ACCEPT; / * 允许反地址解析协议报 * /
    if(skb - > protocol = = htons(ETH_P_IP)) {
      iph = skb - > network_header; / * 网络层头部 * /
    if(iph - > protocol = = SOL_ICMP)
      return FW_ACCEPT; / * 允许网络控制报文 ICMP * /
    if(iph - > protocol = = SOL_TCP) {
      tcph = skb - > transport_header; / * 传输层头部 * /
    if(tcph - > dest = = PERMIT_PORT) {
          / * 允许对 TCP 端口 8080 的访问 * /
    printk(KERN_ALERT " Permit a valid access\n");
      return FW_ACCEPT;}}
    return FW_REJECT;}
```

其中，函数 htons 将 16 位无符号长整型的值由主机字节顺序转换为网络字节顺序，通过对以上各判断分支添加 printk 函数实现包过滤结果的日志文件保存，由于内核 2.6 不再包含内核 2.1.25 中的 config. h 头文件，所以要在头文件 firewall. h 中进行一定的预处理。

至此，本文已经完成了防火墙模块各框架函数的设计，如若添加其它功能函数，都可以参照内核 2.1.25 中/linux/firewall. c 的实现。

3　防火墙模块编译与加载

经过前面的软件编制过程即可得到 firewall. h 和火墙模块，还需为防火墙源代码编制相应的 makefile 文件，makefile 的编写要注意指定内核源码树目录 KERNELDIR，该目录位于/lib/modules/ $（shell uname -r）/build 下。

完成 Makefile 后，编译内核模块的过程如图 2 所示：

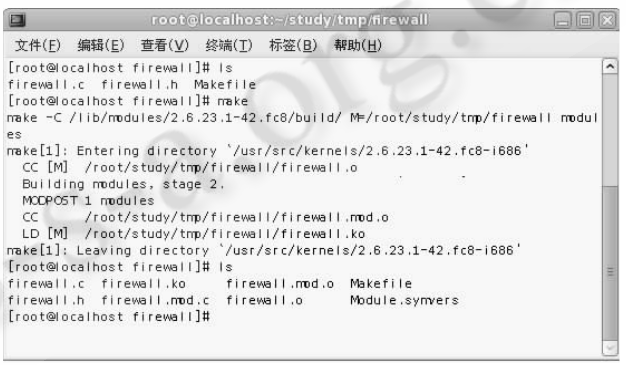

图 2　模块的编译过程

编译后，可以使用以下命令实现内核动态加载模块：

#insmod firewall. ko

防火墙模块成功加载后，内核每收到一个数据包，就会调用防火墙模块进行分类过滤检查，并将检查结果写入/var/log/messeger 日志文件中。若要将包过滤结果实时输出到屏幕，可以通过修改 printk 函数的输出级别来实现。如若卸载防火墙模块，可使用命令：

#rmmod firewall

至此，整个防火墙模块已经编制完毕。（下转第 110 页）

(上接第 117 页)

4 结束语

将防火墙以模块化的形式提供给内核动态加载，有利于提高包过滤检查时调用系统内核相关底层函数的速度，从而使防火墙模块在效率上优于构建在用户空间的防火墙应用程序[1]。本文所设计的防火墙模块功能简单，但给出的框架具有良好的扩展性，用户可根据自己的实际情况添加相关函数，便能实现一款功能上为自己量身定做的高效率包过滤型防火墙。

参考文献

1 陈晓霞，刘寿强，陈梓忠. Linux 个人防火墙的设计与实现—数据包捕获部分. 计算机安全，2005(11)：31－33.

2 张威. 网络编程教程. 北京：希望电子出版社，2002：356－360.

3 魏永明，耿岳，钟书毅译. 设备驱动程序. 第三版，北京：中国电力出版社，2006：22－23.