Netfilter 的实现分析与网络数据包的捕获

The analysis of netfilter's implementation and packet interception

吴 结 高随祥 （中国科学院研究生院 北京 100049）

摘要：netfilter 是出现在 linux 2.4 版本中通用的网络功能框架，该框架具有功能完善，扩充方便的优点，因而被广泛地使用。通过对于源码分析，探讨了 netfilter 在内核中的实现，之后对利用 netfilter 框架进行网络数据包捕获进行了论述，并与其他的网络数据包捕获机制进行了对比。

关键词：netfilter 钩子函数 可加载模块(LKM) 数据包捕获

1 Netfilter 的特点与结构

Netfilter[1,2]是 linux2.4 内核实现数据包过滤/数据包处理/NAT 等的功能框架，它提供了一个抽象、通用化的框架，框架特点如下：

（1）内核定义完善，netfilter 为每种网络协议（IPv4、IPv6 等）定义一套监测点和钩子函数（hook function）（IPv4 定义了 5 个监测点），当数据包流过协议栈的监视点对这些钩子函数被调用，协议栈将数据包及钩子函数标号作为参数调用 netfilter 框架。

（2）扩展性强，netfilter 支持多个钩子函数登记在同一监测点上，并采用优先级机制，顺序执行钩子函数，netfilter 通过检测钩子函数函数列表来判断是否有模块对该协议下的监测点进行了钩子函数注册。若注册了，则调用该模块的注册时使用的钩子函数。

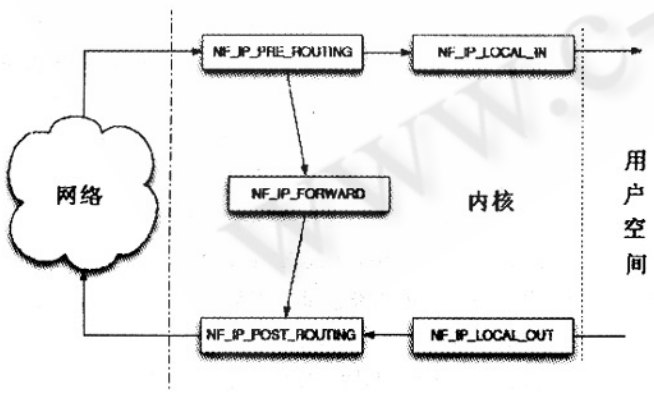

图 1 netfilter 监测点位置图

（3）支持异步操作，并且提供了数据包传递到用户空间的能力，这样允许用户进程检查数据包，修改数据包，并且重新将该数据包通过注入到内核中。

（4）应用方便，采用可以加载模块（Loadable Kernel Module）的方式，实现功能的扩充，不需要对内核进行额外的调整。

对于 IPv4 协议 netfilter 提供了五个监测点[2]，它们的位置图一中下所示，具体的定义如下：

（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），源地址转换在此点进行；

（2）NF_IP_LOCAL_IN：经路由查找后，送往本机的数据包通过此监测点；

（3）NF_IP_FORWARD：要转发的数据包通过此检测点；

（4）NF_IP_POST_ROUTING：所有马上便要通过网络设备出去的数据包通过此检测点，内置的目的地址转换功能（包括地址伪装）在此点进行；

（5）NF_IP_LOCAL_OUT：本机进程发出的数据包通过此检测点。

这些点是已经在内核中定义好的，内核模块可以注册在这些监测点的钩子函数，完成特定的功能。下面我们从源码分析着手，探讨 netfilter 这一通用框架的实现。

2 实现分析

在 netfilter 中引入了全局变量 nf_hooks[4,5]，它是 netfilter 的核心，为 netfilter 良好扩充性提供了可

能。其定义如下，struct list_head nf_hooks[NPROTO][NF_MAX_HOOKS]；是二维数组的存储结构，一维是网络簇，代表不同的网络协议，可以是 PF_INET，PF_INET6，PF_DECnet 等等；二维是监视点，代表网络协议下的监视点，例如：对于 PF_INET 协议簇，可以是 NF_IP_PRE_ROUTING、NF_IP_LOCAL_IN 等五个不同的监视点；数组的成员是函数指针链表，每条链表存储着登记在这一的监视点下的所有钩子函数的函数指针，如 nf_hooks[PF_INET][NF_IP_FORWORD] 就是指在 IPv4 协议中登记在 NF_IP_FORWARD 监视点的所有钩子函数的函数指针链表。内核模块可以通过调用 nf_register_hook() 为特定的监视点登记钩子函数，也可以通过调用 nf_unregister_hook() 取消登记。

下面我们将分析在 IP 协议栈中，如何实现 netfilter 的可扩充性。在 IP 层代码中，有一些带有 NF_HOOK 宏的语句，以 IP 的转发函数 ip_forward() 为例，如下：

NF_HOOK 宏的参数[5,6]分别为：

表 1　NF_HOOK 宏的参数

pf	协议族名，netfilter 架构同样可以用于 IP 层之外，因此这个变量还可以有诸如 PF_INET6，PF_DECnet 等名字
hooknum	监测点的编号
* * pskb	指向 TCP/IP 协议栈中数据包容器的指针；例如 sk_buff。
struct net_device * indev	数据包流入设备的设备结构的指针。
struct net_device * outdev	数据包流出设备的设备结构的指针。
(* okfn)(struct sk_buff *)	函数指针，当所有的该 HOOK 点的所有登记函数调用完后，转而走此流程。

NF_HOOK 宏的定义[4]如下：

```
#ifdef CONFIG_NETFILTER
  #define NF_HOOK (pf, hook, skb, indev, outdev, okfn)
    (list_empty(&nf_hooks[pf][hook]) ? (okfn)(skb)
    :nf_hook_slow(pf,hook,skb,indev,outdev,okfn))
#else
  #define NF_HOOK(pf, hook, skb, indev, outdev, okfn) (okfn)(skb)
#endif
```

根据定义我们不难理解：如果在编译内核时没有配置 netfilter 时，NF_HOOK 就相当于调用最后一个参数，此例中即执行 ip_forward_finish() 函数；否则，首先检查 nf_hook[PF_INET][NF_IP_FORWORD] 队列是否为空，为空则表明该监测点没有登记钩子函数，执行 ip_forward_finish() 函数；不空的话，则进入 nf_hook_slow() 函数，该函数将遍历整个列表，根据登记的优先级，顺序执行登记的钩子函数。由此可见，通过在源码采用 NF_HOOK 宏以及全局变量 nf_hooks，在网络内核中引入了 netfilter 的钩子函数机制。

3　数据包的捕获

数据包是 TCP/IP 协议中信息流的载体。数据包捕获是指获取一个数据包，对数据包进行某种处理后，再将其释放，使该数据包能够按照常规路径进行传输的一种机制。在很多的网络设备和软件中，如网络协议转换设备、VPN、网络分析程序、路由程序、入侵监测系统（Intrusion Detection Systems，IDS）和嗅探器，等等，都包括了捕获数据包、提取或操作数据包的功能。下面我们首先探讨通过 netfilter 实现数据的捕获，并对于几种不同的捕获方式。

3.1　netfilter 进行数据包捕获

在上面的讨论我们知道，以 netfilter 为基础我们可以在几个监测点上登记钩子函数，在钩子函数中队数据包进行处理，根据要求，以不同的返回值决定数据包的流向，从而实现捕获的功能。具体的实现步骤如下：

3.1.1　钩子函数的实现

钩子函数的原型如下所示：

```
static unsigned int hook_func(unsigned int hook,
    struct sk_buff * * pskb,
    const struct net_device * indev,
    const struct net_device * outdev,
    int(* okfn)(struct sk_buff * ))
```

其参数与表一中的参数完全相同，并且也将由 NF_HOOK 宏进行参数传递。在钩子函数中，用户可以根据自己的需要对数据包进行处理，并且其返回值一定是如下的其中之一：

NF_ACCEPT 继续正常传输数据报

NF_DROP 丢弃该数据报，不再传输

NF_STOLEN 模块接管该数据报，不要继续传输该数据报

NF_QUEUE 对该数据报进行排队（通常用于将数据报给用户空间的进程进行处理）

NF_REPEAT 再次调用该钩子函数

钩子函数的返回值直接决定了 nefilter 框架对数据包的处理，因而选择返回值是相当重要也是应当慎重的。在下面的例子中，我们需要拦截所有的 TCP 数据包对其他的数据包不作处理，那么就对 tcp 数据包返回 NF_DROP，其他数据包返回 NF_ACCEPT。

```
static unsigned int interceptor_tcp
    (unsigned int hooknum,struct sk_buff ** skb,
    const struct net_device *in,
    const struct net_device *out,
    int (*okfn)(struct sk_buff *))
{
    struct iphdr *iph;
    iph=(*skb)->nh.iph;
    if(iph->protocol==6) // TCP
      return NF_DROP;
    else
      return NF_ACCEPT;
}
```

3.1.2 数据结构的填充

要进行钩子函数登记需要用到 struct nf_hooks_ps 结构，该结构定义于 /usr/src/include/linux/netfilter.h，类似如下：

```
struct nf_hook_ops
{   struct list_head list;
    nf_hookfn *hook;
    int pf;
    int hooknum;
    /* 钩子函数以升序进行排列 */
    int priority;
};
```

参数如表二的定义：

表 2　nf_hook_ops 结构说明

list	Netfilter 本身是一个钩子链；它指向 netfilter 钩子的头部，通常设置为 { NULL, NULL }。
hook	钩子函数函数指针，与前面描述的函数相同
pf	协议簇；例如，适用于 IPv4 的 PF_INET。
hooknum	监测点标示编号
priority	优先级，一个监测点可能挂多个钩子函数，由优先级决定调用次序

根据钩子函数，监测点以及优先级对结构进行填充。如下面的示例

```
static struct nf_hook_ops nf_tcp_interceptor
    ={ {NULL,NULL} , interceptor_tcp, PF_INET, NF_
IP_PRE_ROUTING, 0};
```

3.1.3 钩子函数的登记与取消

由于 netfilter 处于内核空间，需要以可加载内核模块（LKM）的形式实现钩子函数。可加载内核模块是内核的扩展，可以在需要时附加到内核中，不需要时从内核中删除。由于只加载所需的模块，因此使用可加载内核模块可使内核变得轻便、模块化、灵活和可伸缩。

在加载模块的时候，会调用模块中 init_module() 函数中，所以可以在此函数中，加入登记钩子函数的代码：

```
int nf_register_hook(struct nf_hook_ops *req);
```

卸载模块时，netfilter 结构需要从内核中取消注册。这一操作在 cleanup_module() 函数中完成：

```
void nf_unregister_hook(struct nf_hook_ops *
req);
```

3.1.4 模块的加载/卸载

简单通过 insmod 命令和 rmmod 命令实现对模块的加载和卸载。至此，我们已经实现通过 netfilter 对数据包的简单捕获功能，相应得功能扩充，都可以在已有的钩子函数基础上完成。

```
int init_module(void)
{
    return nf_register_hook(&nf_tcp_interceptor);
}
void cleanup_module(void)
```

```
}
    nf_unregister_hook(&nf_tcp_interceptor);
}
```

3.2 其他的数据包捕获方式

3.2.1 ipchains 钩子函数

防火墙钩子(Firewall hook)是 2.2.x 内核运行的数据包截获方法,防火墙钩子在 TCP/IP 协议栈的 IP 层截获数据包,防火墙钩子同样是可加载内核模块(LKM)实现的,可以根据需要从内核中加载或卸载。它在三个方向进行数据包的捕获,分别是:流入的数据包,流出的数据包和转发的数据包。与 netfilter 相类似,可以使用该数据包截获机制来开发路由程序、VPN、数据包嗅探器或位于网络边缘且要求实时捕获数据包的任何其他网络应用程序。

3.2.2 转向套接字

转向套接字(divert socket)是一种特殊类型的原始套接字,与任何其他套接字一样,通过该套接字可以接收和发送数据包。转向套接字使用 Linux 内核的 ipchains 功能,工作原理是:根据 ipchains 设置的策略对流入、流出和转发数据包机性处理,将其重定向到指定的端口,转向套接字将在该端口进行监听,结收重定向的数据包。使用转向套接字的主要缺点是将数据包复制到用户空间的开销,这会占用时间和资源,从而降低了网络性能。任何没有对数据包实时处理施加硬性限制的联网应用程序(如嗅探器),都可以使用转向套接字。

3.2.3 内核改动

操作内核源代码以扩展内核本地数据包截获能力。内核源文件位于 /usr/src/linux,要修改的两个主要文件是 ip_input.c 和 ip_output.c。这两个文件位于 /usr/src/linux/net/ipv4/ 目录中。

源文件修改:ip_input.c 包含这两个函数,可以为所有流入数据包调用它们

表 3　ip_input.c 中函数

ip_local_deliver	当数据包流入时被调用而不考虑目的地
ip_rcv	当目的地是本地机器的数据包流入时被调用

源文件修改:ip_output.c 它处理流出的数据包。它的四个函数是:

表 4　ip_outinput.c 中函数

ip_build_and_sent_pkt	将 IP 报头添加到 sk_buff 并将其发送出去
ip_queue_xmit	对要发送的数据包排队
ip_build_xmit_slow	构建和传送数据包(处理所有流出的 IP 数据包)
ip_build_xmit	"快速"构建和传送数据包,该选项只用于无需碎片整理的选项。(处理所有流出的 IP 数据包)

可以使用 Makefiles 动态修改内核源代码;Makefile 脚本确保对源代码做必需的修改。由于该机制要求修改 Linux 内核源代码,所以快速调试可能非常困难。而且,由于该方法要求重新编译内核,所以这个过程是冗长、复杂且不灵活的。

3.3 几种方式的对比

我们对以上几种方式进行对比,可以得到如下的结论:

(1) 防火墙钩子机制与 netfilter 钩子机制有很多相同之处,但它们有如下的不同:支持的内核不同,netfilter 在版本在 2.4 之后的内核中,ipchains 仅在 2.2 的版本中;监测点不同,netfilter 有 5 处监测点,ipchains 中只有 3 处监测点;另外,受到 ipchains 本身的影响,ipchains 钩子存在一定的缺陷。

(2) netfilter 钩子机制和转向套接字,它们都可以提供用户空间的数据包捕获,所不同的是,由于转向套接字必须在 ipchains 的基础之上,所以受到了 ipchains 本身使用上的限制。

(3) 内核改动具有在性能上的优势,但实现起来非常复杂,并且难以扩展,因而只适用功能较为单一,需求固定的应用中。以下的表格为几种方式对比表:

表 5　数据包捕获机制对比

机制	实现	用户开发方式	性能影响	空间
Netfilter 钩子	钩子函数在模块内实现,模块可插入/可删除	可以通过更改模块的源代码并重新编译来增强功能	影响不 大	内核空间 + 用户空间
Ipchains 钩子	同上	同上	影响不 大	内核空间
转向套接字	标准的 socket 机制	采用标准的 socket 机制	影响较 大	用户空间
内核改动	直接集成到内核源代码中	修改内核文件,编译整个内核	基本没有影响	内核空间

(下转第 90 页)

(上接第 87 页)

综合以上各点,可以看到,采用 netfilter 钩子机制进行数据包捕获要优于其他的几种方式。

4 结论

Netfilter 是在 2.4 版本后的 Linux 内核中采用的网络框架模型,在本文中讨论了采用这一模型的优点,以源码为基础分析了 netfilter 的实现,讨论了基于 netfilter 框架进行数据包捕获的方法,并与其他方法进行了对比。采用基于 netfilter 的数据包捕获机制具有良好的性能,很好的可扩性是目前数据包捕获机制的首选。

参考文献

1 The netfilter/iptables project. http://www.netfilter.org/

2 P. RUSSEL, H. WELTE. Linux netfilter Hacking HOWTO[J/OL]. 2002/07/02.

3 H. WELTE, The journey of a packet through the linux 2.4 network stack[J/OL]. 2000,10.

4 Redhat Co. ,redhat 7.3 source code. http://www.redhat.com

5 李善平、刘文峰、李程远等,Linux 内核 2.4 版源代码分析大全[M],北京 机械工业出版社,2002 年 1 月。

6 毛德操、胡希明,Linux 内核源代码情景分析[M],浙江,浙江大学出版社,2001 年 9 月。