

jasenwan88的专栏

目录视图

摘要视图

RSS 订阅

个人资料



jasenwan88

访问：41908次

积分：579

等级：BLOG > 3

排名：千里之外

原创：7篇 转载：48篇

译文：1篇 评论：0条

[博客专家福利](#) [公告：CSDN论坛停站维护公告](#) [Qualcomm博客征文活动](#) [参与话题讨论，好礼等你拿](#) [公告：博客新皮肤上线啦](#)

Linux下Libpcap源码分析和包过滤机制（4）

分类：网络基础知识

2012-07-19 13:44

216人阅读

评论(0)

收藏

举报

linux

filter

socket

struct

数据结构

linux内核

过滤代码的安装

前面我们曾经提到，在内核空间过滤数据包对整个捕获机制的效率是至关重要的。早期使用 SOCK_PACKET 方式的 Linux 不支持内核过滤，因此过滤操作只能在用户空间执行（请参阅函数 pcap_read_packet() 代码），在《UNIX 网络编程(第一卷)》（参考资料 B）的第 26 章中对此有明确的描述。不过现在看起来情况已经发生改变，Linux 在 PF_PACKET 类型的 socket 上支持内核过滤。Linux 内核允许我们把一个名为 LPF(Linux Packet Filter) 的过滤器直接放到 PF_PACKET 类型 socket 的处理过程中，过滤器在网卡接收中断执行后立即执行。LSF 基于BPF机制，但两者在实现上有略微的不同。实际代码如下：

文章搜索

文章分类

[linux内核](#) (25)[编辑工具](#) (2)[移植](#) (1)[网络基础知识](#) (16)[linux基础](#) (4)[服务器类](#) (4)[unix高级环境编程](#) (2)[项目需求](#) (1)[luci](#) (1)

文章存档

[2014年11月](#) (1)[2012年07月](#) (29)[2012年06月](#) (22)[2012年04月](#) (1)[2012年03月](#) (2)[展开](#)

阅读排行

[emacs 命令小结---开关、](#) (8295)

```
/* 在包捕获设备上附加 BPF 代码 [pcap-Linux.c]*/
static int
pcap_setfilter_Linux(pcap_t *handle, struct bpf_program *filter)
{
#ifdef SO_ATTACH_FILTER
    struct sock_fprog      fcode;
    int can_filter_in_kernel;
    int err = 0;
#elseif
#endif

    /* 检查句柄和过滤器结构的正确性 */
    if (!handle)
        return -1;
    if (!filter) {
        strncpy(handle->errbuf, "setfilter: No filter specified",
            sizeof(handle->errbuf));
        return -1;
    }

    /* 具体描述如下 */
    if (install_bpf_program(handle, filter) < 0)
        return -1;

    /* 缺省情况下在用户空间运行过滤器,但如果
    在内核安装成功,则值为 1 */
```

[天玑网络安全审计系统 \(4592\)](#)
[关于IP选项 \(1729\)](#)
[Ubuntu linux下安装sqlite \(1619\)](#)
[Openssl EVP 说明三 分 \(1386\)](#)
[ifconf和ifreq \(1261\)](#)
[Openssl ASN.1 说明二 \(1113\)](#)
[POSIX semaphore: sem \(1056\)](#)
[Openssl ASN.1 说明一 \(996\)](#)
[libpcap的使用 \(954\)](#)

评论排行

[openwrt中luci界面中简单 \(0\)](#)
[Linux下Libpcap源码分析 \(0\)](#)
[Linux下Libpcap源码分析 \(0\)](#)
[Linux下Libpcap源码分析 \(0\)](#)
[以開源碼 dansguardian \(0\)](#)
[如何定制Ubuntu 12.04 C \(0\)](#)
[linux如何建立IP隧道 \(0\)](#)
[linux下的多进程服务器框 \(0\)](#)
[精通top,ps命令 \(0\)](#)
[Authentication Proxy原理 \(0\)](#)

推荐文章

```
handle->md.use_bpf = 0;

/* 尝试在内核安装过滤器 */
#ifdef SO_ATTACH_FILTER
#ifdef USHRT_MAX
if (handle->fcode.bf_len > USHRT_MAX) {
/*过滤器代码太长, 内核不支持 */
fprintf(stderr, "Warning: Filter too complex for kernel\n");
fcode.filter = NULL;
can_filter_in_kernel = 0;
} else
#endif /* USHRT_MAX */
{
/* Linux 内核设置过滤器时使用的数据结构是 sock_fprog,
而不是 BPF 的结构 bpf_program ,因此应做结构之间的转换 */
switch (fix_program(handle, &fcode)) {

/* 严重错误, 直接退出 */
case -1:
default:
return -1;

/* 通过检查, 但不能工作在内核中 */
case 0:
```

- * [struts2国际化](#)
- * [无需超级用户mpi多机执行](#)
- * [从视图索引说Notes数据库（下）](#)
- * [Java虚拟机解析篇之---垃圾回收器](#)
- * [2015互联网校招总结——一路走来](#)
- * [SSL 3.0曝出Poodle漏洞的解决方案-----开发者篇](#)

```
can_filter_in_kernel = 0;
break;

/* BPF 可以在内核中工作 */
case 1:
can_filter_in_kernel = 1;
break;
}
}

/* 如果可以在内核中过滤，则安装过滤器到内核中 */
if (can_filter_in_kernel) {
if ((err = set_kernel_filter(handle, &fcode)) == 0)
{
/* 安装成功 !!! */
handle->md.use_bpf = 1;
}
else if (err == -1)      /* 出现非致命性错误 */
{
if (errno != ENOPROTOOPT && errno != EOPNOTSUPP) {
fprintf(stderr, "Warning: Kernel filter failed:
%s\n", pcap_strerror(errno));
}
}
}
```

```
/* 如果不能在内核中使用过滤器，则去掉曾经可能在此 socket
上安装的内核过滤器。主要目的是为了避免存在的过滤器对数据包过滤的干扰 */
if (!handle->md.use_bpf)
    reset_kernel_filter(handle);[pcap-Linux.c]
#endif
}

/* 把 BPF 代码拷贝到 pcap_t 数据结构的 fcode 上 */
int install_bpf_program(pcap_t *p, struct bpf_program *fp)
{
    size_t prog_size;

    /* 首先释放可能已存在的 BPF 代码 */
    pcap_freecode(&p->fcode);

    /* 计算过滤代码的长度，分配内存空间 */
    prog_size = sizeof(*fp->bf_insns) * fp->bf_len;
    p->fcode.bf_len = fp->bf_len;
    p->fcode.bf_insns = (struct bpf_insn *)malloc(prog_size);
    if (p->fcode.bf_insns == NULL) {
        snprintf(p->errbuf, sizeof(p->errbuf),
            "malloc: %s", pcap_strerror(errno));
        return (-1);
    }
}
```

```
}

/* 把过滤代码保存在捕获句柄中 */
memcpy(p->fcode.bf_insns, fp->bf_insns, prog_size);

return (0);
}

/* 在内核中安装过滤器 */
static int set_kernel_filter(pcap_t *handle, struct sock_fprog *fcode)
{
    int total_filter_on = 0;
    int save_mode;
    int ret;
    int save_errno;

    /*在设置过滤器前，socket 的数据包接收队列中可能已存在若干数据包。当设置过滤器
    后，
    这些数据包极有可能不满足过滤条件，但它们不被过滤器丢弃。
    这意味着，传递到用户空间的头几个数据包不满足过滤条件。
    注意到在用户空间过滤这不是问题，因为用户空间的过滤器是在包进入队列后执行的。
    libpcap 解决这个问题的方法是在设置过滤器之前，
    首先读完接收队列中所有的数据包。具体步骤如下。*/

    /*为了避免无限循环的情况发生（反复的读数据包并丢弃，
```

```
但新的数据包不停的到达)，首先设置一个过滤器，阻止所有的包进入 */

setsockopt(handle->fd, SOL_SOCKET, SO_ATTACH_FILTER,
&total_fcode, sizeof(total_fcode);

/* 保存 socket 当前的属性 */
save_mode = fcntl(handle->fd, F_GETFL, 0);

/* 设置 socket 它为非阻塞模式 */
fcntl(handle->fd, F_SETFL, save_mode | O_NONBLOCK);

/* 反复读队列中的数据包，直到没有数据包可读。这意味着接收队列已被清空 */
while (recv(handle->fd, &drain, sizeof drain, MSG_TRUNC) >= 0);

/* 恢复曾保存的 socket 属性 */
fcntl(handle->fd, F_SETFL, save_mode);

/* 现在安装新的过滤器 */
setsockopt(handle->fd, SOL_SOCKET, SO_ATTACH_FILTER,
fcode, sizeof(*fcode));
}

/* 释放 socket 上可能有的内核过滤器 */
static int reset_kernel_filter(pcap_t *handle)
{
```

```
int dummy;  
return setsockopt(handle->fd, SOL_SOCKET, SO_DETACH_FILTER,  
&dummy, sizeof(dummy));  
}
```

Linux 在安装和卸载过滤器时都使用了函数 setsockopt(), 其中标志 SOL_SOCKET 代表了对 socket 进行设置, 而 SO_ATTACH_FILTER 和 SO_DETACH_FILTER 则分别对应了安装和卸载。下面是 Linux 2.4.29 版本中的相关代码:

```
[net/core/sock.c]  
#ifdef CONFIG_FILTER  
case SO_ATTACH_FILTER:  
.....  
/* 把过滤条件结构从用户空间拷贝到内核空间 */  
if (copy_from_user(&fprog, optval, sizeof(fprog)))  
break;  
/* 在 socket 上安装过滤器 */  
ret = sk_attach_filter(&fprog, sk);  
  
.....  
  
case SO_DETACH_FILTER:  
/* 使用自旋锁锁住 socket */  
spin_lock_bh(&sk->lock.slock);
```



```
filter = sk->filter;
/* 如果在 socket 上有过滤器，则简单设置为空，并释放过滤器内存 */
if (filter) {
    sk->filter = NULL;
    spin_unlock_bh(&sk->lock.slock);
    sk_filter_release(sk, filter);
    break;
}
spin_unlock_bh(&sk->lock.slock);
ret = -ENONET;
break;
#endif
```

上面出现的 `sk_attach_filter()` 定义在 `net/core/filter.c`，它把结构 `sock_fprog` 转换为结构 `sk_filter`，最后把此结构设置为 socket 的过滤器：`sk->filter = fp`。

其他代码

`libpcap` 还提供了其它若干函数，但基本上是提供辅助或扩展功能，重要性相对弱一点。我个人认为，函数 `pcap_dump_open()` 和 `pcap_open_offline()` 可能比较有用，使用它们能把在线的数据包写入文件并事后进行分析处理。

总结

1994 年libpcap 的第一个版本被发布，到现在已有 11 年的历史，如今libpcap 被广泛的应用在各种网络监控软件中。libpcap 最主要的优点在于平台无关性，用户程序几乎不需做任何改动就可移植到其它 unix 平台上；其次，libpcap也能适应各种过滤机制，特别对BPF的支持最好。分析它的源代码，可以学习开发者优秀的设计思想和实现技巧，也能了解到（Linux）操作系统的网络内核实现，对个人能力的提高有很大帮助。

[上一篇](#) [Linux下Libpcap源码分析和包过滤机制（3）](#)

[下一篇](#) [libpcap的用法入门](#)

主题推荐

[源码](#)

[linux](#)

[数据结构](#)

[网络编程](#)

[操作系统](#)

猜你在找

[应用层HTTP数据包的截获与还原技术的实现](#)

[linux sock_raw原始套接字编程（转）和Linux下Libpcap](#)

[Linux操作系统下如何编译安装源码包软件](#)

[proxy_epoll源代码分析 linux网络编程入门的源码分析资](#)

[Linux网络地址转换NAT源码分析](#)

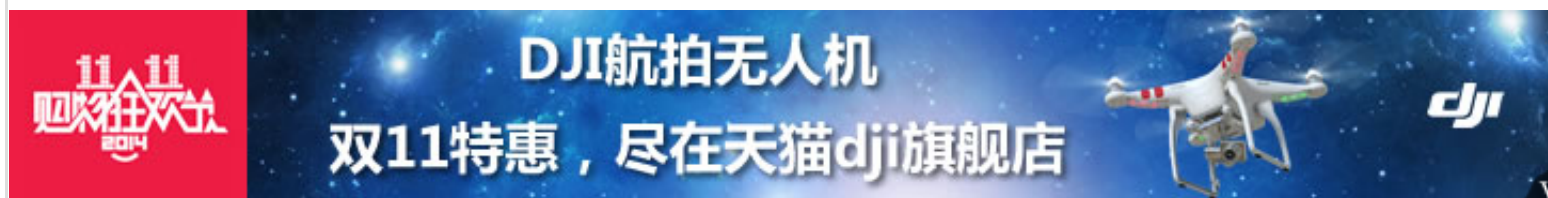
[linux sock_raw原始套接字编程（转）和Linux下Libpcap](#)

[Linux下Libpcap源码分析和包过滤机制（3）](#)

[Linux下Libpcap源码分析和包过滤机制](#)

[Linux下Libpcap源码分析和包过滤机制](#)

[Linux内核源码系列（二）：探究内核基础层数据结构，](#)



[查看评论](#)

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2014, CSDN.NET, All Rights Reserved

