jasenwan88的专栏

Linux下Libpcap源码分析和包过滤机制（1）

分类： 网络基础知识 2012-07-19 13:40 667人阅读 评论(0) 收藏 举报

linux 网络 网络协议 数据结构 工作 struct

当设备找到后，下一步工作就是打开设备以准备捕获数据包。Libpcap的包捕获是建立在具体的操作系统所提供的捕获机制上，而Linux系统随着版本的不同，所支持的捕获机制也有所不同。

libpcap是unix/Linux平台下的网络数据包捕获函数包，大多数网络监控软件都以它为基础。libpcap可以在绝大多数类unix平台下工作，本文分析了libpcap在Linux 下的源代码实现，其中重点是Linux的底层包捕获机制和过滤器设置方式,同时也简要的讨论了 libpcap使用的包过滤机制 BPF。

网络监控

绝大多数的现代操作系统都提供了对底层网络数据包捕获的机制，在捕获机制之上可以建立网络监控（Network Monitoring）应用软件。网络监控也常简称为sniffer,其最初的目的在于对网络通信情况进行监控，以对网络的一些异常情况进行调试处理。但随着互连网的快速普及和网络攻击行为的频繁出现，保护网络的运行安全也成为监控软件的另一个重要目的。例如，网络监控在路由器，防火墙、入侵检查等方面使用也很广泛。除此而外，它也是一种比较有效的黑客手段，例如，美国政府安全部门的"肉食动物"计划。

包捕获机制

从广义的角度上看，一个包捕获机制包含三个主要部分：最底层是针对特定操作系统的包捕获机制，最高层是针对用户程序的接口，第三部分是包过滤机制。

不同的操作系统实现的底层包捕获机制可能是不一样的，但从形式上看大同小异。数据包常规的传输路径依次为网卡、设备驱动层、数据链路层、IP层、传输层、最后到达应用程序。而包捕获机制是在数据链路层增加一个旁路处理，对发送和接收到的数据包做过滤/缓冲等相关处理，最后直接传递到应用程序。值得注意的是，包捕获机制并不影响操作系统对数据包的网络栈处理。对用户程序而言，包捕获机制提供了一个统一的接口，使用户程序只需要简单的调用若干函数就能获得所期望的数据包。这样一来，针对特定操作系统的捕获机制对用户透明，使用户程序有比较好的可移植性。包过滤机制是对所捕获到的数据包根据用户的要求进行筛选，最终只把满足过滤条件的数据包传递给用户程序。

libpcap应用程序框架

libpcap提供了系统独立的用户级别网络数据包捕获接口，并充分考虑到应用程序的可移植性。libpcap可以在绝大多数类unix平台下工作，参考资料 A 中是对基于 libpcap 的网络应用程序的一个详细列表。在windows平台下，一个与libpcap 很类似的函数包 winpcap 提供捕获功能，其官方网站是http://winpcap.polito.it/。

libpcap 软件包可从 http://www.tcpdump.org/ 下载，然后依此执行下列三条命令即可安装，但如果希

评论排行

推荐文章

望libpcap能在Linux上正常工作，则必须使内核支持"packet"协议，也即在编译内核时打开配置选项CONFIG_PACKET(选项缺省为打开)。

```
./configure
./make
./make install
```

libpcap源代码由20多个C文件构成，但在Linux系统下并不是所有文件都用到。可以通过查看命令make的输出了解实际所用的文件。本文所针对的libpcap版本号为0.8.3，网络类型为常规以太网。libpcap应用程序从形式上看很简单，下面是一个简单的程序框架：

```
char * device; /* 用来捕获数据包的网络接口的名称 */
pcap_t * p; /* 捕获数据包句柄，最重要的数据结构 */
struct bpf_program fcode; /* BPF 过滤代码结构 */

/* 第一步：查找可以捕获数据包的设备 */
device = pcap_lookupdev(errbuf);

/* 第二步：创建捕获句柄，准备进行捕获 */
p = pcap_open_live(device, 8000, 1, 500, errbuf);

/* 第三步：如果用户设置了过滤条件，则编译和安装过滤代码 */
pcap_compile(p, &fcode, filter_string, 0, netmask);
```

* 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

```
pcap_setfilter(p, &fcode);

/* 第四步：进入（死）循环，反复捕获数据包 */
for( ; ; )
{
while((ptr = (char *)(pcap_next(p, &hdr))) == NULL);

/* 第五步：对捕获的数据进行类型转换，转化成以太数据包类型 */
eth = (struct libnet_ethernet_hdr *)ptr;

/* 第六步：对以太头部进行分析，判断所包含的数据包类型，做进一步的处理 */
if(eth->ether_type == ntohs(ETHERTYPE_IP))
……
if(eth->ether_type == ntohs(ETHERTYPE_ARP))
……
}

/* 最后一步：关闭捕获句柄,一个简单技巧是在程序初始化时增加信号处理函数,
以便在程序退出前执行本条代码 */
pcap_close(p);
```

检查网络设备

libpcap 程序的第一步通常是在系统中找到合适的网络接口设备。网络接口在Linux网络体系中是一个很重要的概念，它是对具体网络硬件设备的一个抽象，在它的下面是具体的网卡驱动程序，而其上则是网络协议

层。Linux中最常见的接口设备名eth0和lo。Lo 称为回路设备，是一种逻辑意义上的设备,其主要目的是为了调试网络程序之间的通讯功能。eth0对应了实际的物理网卡，在真实网络环境下，数据包的发送和接收都要通过 eht0。如果计算机有多个网卡，则还可以有更多的网络接口，如eth1,eth2 等等。调用命令ifconfig可以列出当前所有活跃的接口及相关信息，注意对eth0的描述中既有物理网卡的MAC地址，也有网络协议的IP地址。查看文件/proc/net/dev也可获得接口信息。

libpcap调用pcap_lookupdev()函数获得可用网络接口的设备名。首先利用函数 getifaddrs() 获得所有网络接口的地址，以及对应的网络掩码、广播地址、目标地址等相关信息，再利用 add_addr_to_iflist()、add_or_find_if()、get_instance() 把网络接口的信息增加到结构链表 pcap_if 中，最后从链表中提取第一个接口作为捕获设备。其中 get_instanced()的功能是从设备名开始,找第一个是数字的字符,做为接口的实例号。网络接口的设备号越小，则排在链表的越前面，因此，通常函数最后返回的设备名为 eth0。虽然 libpcap 可以工作在回路接口上，但显然libpcap 开发者认为捕获本机进程之间的数据包没有多大意义。在检查网络设备操作中，主要用到的数据结构和代码如下：

```
       /* libpcap 自定义的接口信息链表 [pcap.h] */
struct pcap_if
{
struct pcap_if *next;
char *name; /* 接口设备名 */
char *description; /* 接口描述 */

/*接口的 IP 地址, 地址掩码, 广播地址,目的地址 */
struct pcap_addr addresses;
bpf_u_int32 flags;      /* 接口的参数 */
```

```
};

char * pcap_lookupdev(register char * errbuf)
{
        pcap_if_t *alldevs;
        ……
                pcap_findalldevs(&alldevs, errbuf);
                ……
                strlcpy(device, alldevs->name, sizeof(device));
        }
```

上一篇 以開源碼 dansguardian+tinyproxy 實作色情守門員

下一篇 Linux下Libpcap源码分析和包过滤机制（2）

主题推荐 源码 linux 应用程序 网络协议 数据结构

猜你在找

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