doi: 10.7690/bgzdh.2013.07.026

基于 Zentyal 的数据过滤程序

吴昌昊
（中国兵器工业第五八研究所特种电子技术部，四川 绵阳 621000）

摘要：为了满足获取特殊数据包的需求，开发基于网关操作系统 Zentyal 的数据过滤程序。在 Zentyal 平台上，采用 Libpcap 网络数据捕获函数包，运用了函数包提供的应用程序开发接口，根据以太网的网络数据结构使用 C 语言开发了一个数据过滤程序，并给出了详细流程和部分代码。结果表明：该程序实现了捕获数据包，并根据需要丢弃无用数据包的功能。

关键词：Libpcap；函数包；Zentyal；网关；数据捕获；数据过滤
中图分类号：TJ02　　**文献标志码**：A

Data Filter Program Based on Zentyal

Wu Changhao
(*Department of Special Electronics Technology*, *No.* 58 *Research Institute of China Ordnance Industries*, *Mianyang* 621000, *China*)

Abstract: In order to capture specific data packets, developed data filter program under gateway operating system Zentyal. Based on Zentyal platform, use Libpcap network data to capture function package and adopt application program based on function package to develop interface. According to Ethernet network data structure to develop data filter program based on C language. Then put forwards detailed flow and part of code. The result shows that this program has realized these function of capture data packets and reject useless data package according to requirements.

Key words: Libpcap; function package; Zentyal; gateway; data capture; data filter

0　引言

Zentyal 的前身是 eBox Platform，是非常流行的 Linux 安全网关发行版，现成为 eBox 的社区版品牌。Zentyal 能提供网关、统一安全管理、办公服务、基础设施管理、统一通信服务等功能。该系统的源代码是基于 GPL 发布的，可进行二次开发。

由于 Zentyal 不能完成特殊的数据包获取需求，而完全从底层开始编写关于 OSI 网络模型中网络层和传输层程序，不仅难度很大，开发周期很长，还不一定能保证成功实现跨平台。在这种情况下，如果基于一个成熟的、受到广泛认可的包捕获机制和函数包进行开发，将会大大提高开发效率，并且可以提高程序的健壮性。

所谓包捕获机制，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。它并不影响操作系统对数据包的网络栈处理，使针对特定操作系统的捕获机制对用户透明，从而获得了较好的可移植性。Libpcap 是 Unix/Linux 平台下的网络数据包捕获函数包，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，大多数网络监控软件都以它为基础。基于此，笔者采用 Libpcap1.3.0 的网络数据捕获函数包，开发基于网关操作系统 Zentyal 的数据过滤程序。

1　开发环境搭建

1.1　Zentyal 系统的准备工作

为减小自身体积，Zentyal 没有集成任何与开发相关的软件；因此，首先需要自行下载安装软件包。

Linux 下比较常见的几种软件安装方式有：直接运行软件可执行安装文件、使用 apt、rpm、pacman 等包管理器进行安装、使用软件源码进行安装[1]。

Zentyal 3.0 是基于著名发行版 Ubuntu 12.04 制作，而 Ubuntu 则衍生于 Debian 发行版，所以可直接使用 apt 包管理器离线安装.deb 格式的软件包，并可根据 Ubuntu 官网提供的软件包依赖关系分析软件包从属，并按顺序安装，基本安装指令如下：

sudo dpkg –i [package name].deb

通过上述方法，可成功安装对开发环境比较重要的几个软件：make、gcc、gdb、glibc，libstdc++、m4、flex、bison 等，然后就可以编译安装 Lipcap。

1.2　Libpcap 的安装

Libpcap 是完全开源的，其源码压缩包可在官网 www.tcpdump.org 下载到。整个安装过程皆在命令行终端中完成，具体如下：

tar –zxvf libpcap-1.3.0.tar.gz

收稿日期：2013-01-27；修回日期：2013-02-03
作者简介：吴昌昊(1990—)，男，四川人，本科，助理工程师，从事 Linux 软件开发和高性能并行运算研究。

```
cd ./libpcap-1.3.0
./configure
make
make install
```

经过以上几个步骤就完成了 libpcap 的安装。

1.3 基于 Libpcap 程序的编写与编译

本次开发无需使用任何 IDE(集成开发环境)，仅仅通过 Vim 文档编辑器和 gcc 编译器的搭配，在命令行终端下即可完成整个程序的编写和编译，这样就节省下大量的环境搭建时间，而且不会使系统变得臃肿[2]。

Libpcap 本身是由 C 语言完成的，提供的函数封装形式和其他普通 C 函数并无本质上的区别，但在预处理阶段需要通过#include 代码引入 pcap.h 这个头文件。

编译阶段需要人工引入 pcap 的库，否则不能成功通过编译，会在链接阶段失败。指令如下所示：

```
gcc –o example example.c –O2 –lpcap
```

2 数据过滤主函数基本工作流程

主函数的内容相对比较单一和固定，即使程序目的有所改变，主程序的模式也不会变，大致流程如图 1 所示。

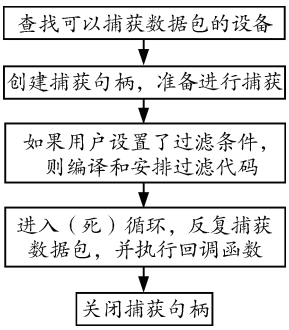

图 1 过滤程序主函数流程[3]

部分代码如下：

```
//声明部分
char *dev=NULL;//设备指针
char errbuf[PCAP_ERRBUF_SIZE];//保存错误信息的数组
pcap_t *handle;//句柄指针，指向打开的设备
char filter_exp[]="ip"; //过滤函数表达式，此时将过滤获取所有 IP 数据包
struct bpf_program fp;//编译过滤程序所需结构体
bpf_u_int32 mask;//用于存放掩码地址
bpf_u_int32 net;//存放网络地址
int num_packets =-1; //需捕获的数据包数量，-1 表示无限
//查找可以捕获数据包的设备
if(pcap_lookupnet(dev,&net,&mask,errbuf)==-1)
{
 fprintf(
stderr,"cant get netmask for device:%s\n",dev);
 net=0;
 mask=0;
}
//创建捕获句柄，准备进行捕获
handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
if(handle==NULL){
 fprintf(
stderr,"couldnt open device %s:%s\n",dev,errbuf);
 exit(EXIT_FAILURE);
}
//编译和安装过滤代码
if(
pcap_compile(handle,&fp,filter_exp,0,net)==-1)
{
fprintf(stderr,"couldnt parse filter %s:%s\n",
filter_exp,pcap_geterr(handle));
exit(EXIT_FAILURE);
}
if(pcap_setfilter(handle,&fp)==-1){
fprintf(stderr,"couldnt install filter %s:%s\n",
filter_exp,pcap_geterr(handle));
exit(EXIT_FAILURE);
}
//进入(死)循环，反复捕获数据包
pcap_loop(handle, num_packets, got_packet, NULL);
//关闭捕获句柄
pcap_freecode(&fp);
pcap_close(handle);
```

3 捕获函数的工作流程

因为主函数并没有完成数据处理的功能，仅仅有主函数是不完整的。捕获函数才是整个程序的工作重点，它要完成捕获、过滤和处理数据的功能。

主函数中执行捕获和处理的代码只有短短一行，由 pcap_loop()完成。pcap_loop()函数的第 3 个参数就是回调(Call-back)函数，其实也就是捕获函数，在主函数执行循环的时候，回调函数已经开始无间断地运行。回调函数的主要工作是：对捕获的数据进行类型转换，转化成以太数据包类型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。需要注意的是，虽然调用回调函数时没有带参数，但实际上是需要实参的，如下：

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
```

过滤流程如图 2 所示。

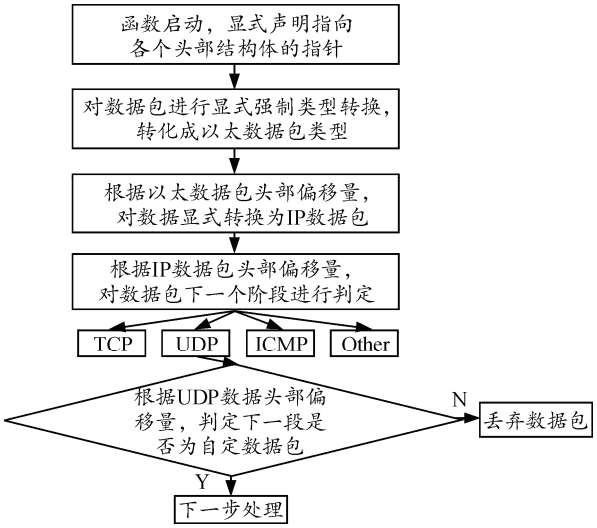

图 2　捕获函数流程

本次捕获函数中假设数据包是通过 UDP 协议发送过来的自制头部的数据包，所以 IP 头部的下一个分析任务就是主要针对 UDP 的封装，部分捕获代码如下：

```
//声明部分
const struct sniff_ethernet*ethernet;//以太网头部
const struct sniff_ip *ip;//IP 头部
const struct sniff_udp *udp;//udp 头部
const struct sniff_custom *custom; //自定义数据包结构体
//强制类型转换，获取以太数据包和 IP 数据包的地址
ethernet=(struct sniff_ethernet*)(packet);
ip=(struct sniff_ip*)(packet+SIZE_ETHERNET);
//协议判断，捕获 udp 数据包
switch(ip->ip_p){
    case IPPROTO_TCP:
      printf(" protocol:TCP\n");
      return;
    case IPPROTO_UDP:
      printf(" protocol:UDP\n");
      break;
    case IPPROTO_ICMP:
      printf(" protocol:ICMP\n");
      return;
    case IPPROTO_IP:
      printf(" protocol:IP\n");
      return;
    default:
      printf(" protocol:unknown\n");
      return;
}
//获取 UDP 数据包地址
udp=(struct sniff_udp*)(packet+SIZE_ETHERNET+size_ip);
//获取自定义数据包地址
custom=(struct sniff_custom*)(packet+SIZE_ETHERNET+size_ip+size_udp);
```

这时已经获得了指向自制数据包头部位置的指针，接下来便要判断头部之后的部分是否是我们自制的结构体内容。笔者假设自定数据包结构体的第 1 个无符号整形变量的值为 0xAAAAAAAA，这就意味着 UDP 结构体结束之后的连续 4 个字节是作为一个无符号整形出现，而且是固定值，否则就不是笔者定义的数据包。伪代码如下所示：

```
#define HEAD (custom->cus_hd)
if((u_int)HEAD!=0xAAAAAAAA){
    printf("Custom Packet Captured\n");
    （处理过程）
    return;
}
else{
    printf("Wrong Packet\n");
    return
}
```

至此，数据包过滤的工作已经完成。接下来便可以根据获得的数据进行处理。此处具体代码会因实现方式和目的的不同而不同，最简单的方法是：当数据结构满足要求时，输出一个信息提示即可。

4　数据包头部结构体

从上小节的声明部分代码可以看到，数据包头部实质为结构体[4]，结构体的声明和定义也是在本程序中完成，代码如下：

```
//以太网头部
#define ETHER_ADDR_LEN 6
struct sniff_ethernet{
u_char  ether_dhost[ETHER_ADDR_LEN];  //目标地址
u_char  ether_shost[ETHER_ADDR_LEN];  //源地址
u_short ether_type; //数据类型
};
//ip 头部
struct sniff_ip{
u_char ip_vhl;
u_char ip_tos;
u_short ip_len;
u_short ip_id;
u_short ip_off;
#define IP_RF 0x8000
#define IP_DF 0x4000
#define IP_MF 0x2000
```

```
    #define IP_OFFMASK 0x1fff
    u_char ip_ttl;
    u_char ip_p;
    u_short ip_sum;
    struct in_addr ip_src,ip_dst;
};
    //udp 头部
    struct sniff_udp{
    u_short uh_sport;
    u_short uh_dport;
    u_short uh_ulen;
    u_short uh_sum;
};
```

以上列出的以太网头部结构体长度为 14 字节，IP 包头部长度为 20 字节，UDP 包头部长度为 8 字节，计算数据包内容开始处相对于数据包起始的偏移量需要这些数据。

若是自行构造数据包头，按照上面列出的结构体进行构造新的、满足需求的结构体即可，结构体大小可用 sizeof()函数得到。如果遇到了“字节对齐”的问题[5]，即数据结构体实际长度大于计划长度，则需要在结构体定义部分加入预处理声明，方式为：

```
    #pragma pack(push)
    #pragma pack(1)
    struct sniff_custom{
        //结构体内容示意
        u_int cus_hd;
        u_char cus_flag;
        u_int cus_data;
        u_int cus_end;
    }
    #pragma pack(pop)
```

5 结束语

通过数据捕获函数包 libpcap 和构建的特定数据包头部结构体，实现了具备数据包的捕获、对无关数据包进行丢弃功能的程序。

在此基础上，如果再针对已获取的数据包按需求进行处理，就能达到扩充 Zentyal 网关能力的目的。比如：将自行封装的数据包头部分加入一个判断位或者字节，可将被封装的数据内容区分为控制数据和其他数据 2 种类型，网关截获到控制数据后根据其内容完成定制的功能[6]。

同时，鉴于 libpcap 良好的兼容性，整个程序理论上可以无需修改就使用在 Linux 的其他发行版上，具有比较好的通用性。

参考文献：

[1] Richard Blum, Christine Bresnahan. Linux 命令行与 shell 脚本编程大全[M]. 2 版. 北京: 人民邮电出版社, 2012.

[2] 宋敬彬, 孙海滨. Linux 网络编程[M]. 北京: 清华大学出版社, 2010.

[3] 唐军, 陈勇, 周虎, 等. 基于 VC 的伺服速度环特性分析系统[J]. 兵工自动化, 2012, 31(10): 93-96.

[4] Christian Benvenuti. 深入理解 LINUX 网络技术内幕[M]. 北京: 中国电力出版社, 2009.

[5] 陈伟, 陈先玉. 一种实时系统消息类数据存储方法[J]. 兵工自动化, 2012, 31(11): 44-45.

[6] 卢冬亮, 廖兴禾, 宋令邦, 等. 数据融合系统分析[J]. 四川兵工学报, 2010, 31(10): 93.

****

（上接第 85 页）

3 改造前后性能对比

经试验，改造后火炮随动系统的集成电路放大器与原电子管放大器性能对比结果如表 1 所示。

表 1 集成电路放大器与原电子管放大器性能对比

项目	电子管放大器	晶体管放大器
响应带宽	较宽	宽
电磁兼容性	较好	好
稳定精度	较好	较好
抗振性	一般	较强
平均无故障时间	数百小时	2 000 h 以上
故障排除平均时间	2 h	0.5 h
线性误差	较差	较高
保障性	差(缺配件)	好
元器件寿命	低	提高 200 倍以上
战斗准备时间	预热时间不少于 15 min	不需要预热

4 结束语

目前，该放大器已成功使用某高炮随动系统中，其控制精度高，稳定性和可靠性高，不需要预热，反应速度快，提高了战术指标，解决了我国现役某高炮电子管放大器维修难、稳定性差的技术难题。目前，使用该方法设计的放大器已在部队中得到了推广应用，并已获得专利授权(专利号 ZL 2007 10080475.6“一种用于火炮随动系统的放大装置”)。

参考文献：

[1] 邱益朋, 韩杨. 提高炮兵指挥信息系统使用效能[J]. 四川兵工学报, 2010, 31(6): 127.

[2] 唐恭富. 一种用于火炮随动系统的放大装置: 中国, ZL 2007 10080475.6[P]. 2007-01-15.

[3] 陈有卿. 新颖集成电路应用手册[S]. 北京: 人民出版社, 1997.

[4] 孔有林. 集成运算放大器及其应用[M]. 北京: 人民邮电出版社, 1988.

[5] 张伯文. 由模拟开关组成的相敏检波器[J]. 电测与仪表, 1992(9).

[6] 陈先玉, 苏艳蕊, 袁强, 等. 基于 ADG726 的开关电路系统[J]. 兵工自动化, 2012, 31(3): 73-74.