原文：https://blog.csdn.net/tennysonsky/article/details/44811899

**概述**

libpcap 是一个网络数据包捕获函数库，功能非常强大，Linux 下著名的 tcpdump 就是以它为基础的。

**libpcap主要的作用**

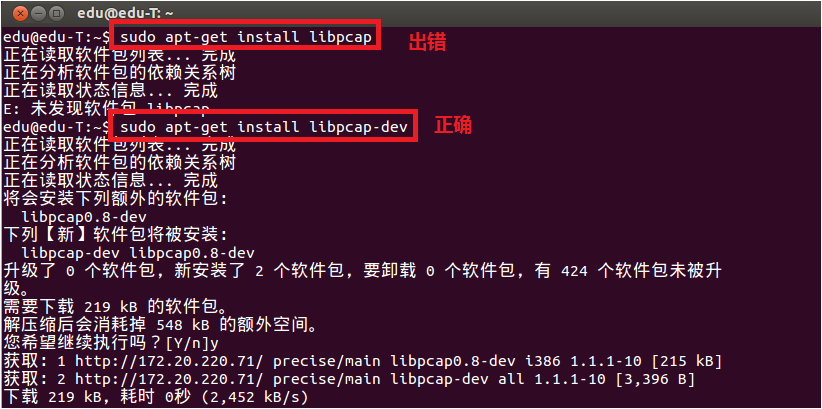
1）捕获各种数据包，列如：网络流量统计。

2）过滤网络数据包，列如：过滤掉本地上的一些数据，类似防火墙。

3）分析网络数据包，列如：分析网络协议，数据的采集。

4）存储网络数据包，列如：保存捕获的数据以为将来进行分析。

**libpcap 的安装**



**libpcap 的抓包框架**

pcap\_lookupdev()：函数用于查找网络设备，返回可被 pcap\_open\_live() 函数调用的网络设备名指针。

pcap\_lookupnet()：函数获得指定网络设备的网络号和掩码。  
pcap\_open\_live()： 函数用于打开网络设备，并且返回用于捕获网络数据包的数据包捕获描述字。对于此网络设备的操作都要基于此网络设备描述字。  
pcap\_compile()： 函数用于将用户制定的过滤策略编译到过滤程序中。  
pcap\_setfilter()：函数用于设置过滤器。  
pcap\_loop()：函数 pcap\_dispatch() 函数用于捕获数据包，捕获后还可以进行处理，此外 pcap\_next() 和 pcap\_next\_ex() 两个函数也可以用来捕获数据包。  
pcap\_close()：函数用于关闭网络设备，释放资源。

**利用 libpcap 函数库开发应用程序的基本步骤：**

1、打开网络设备

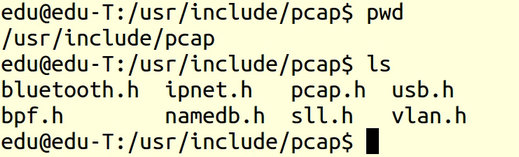
2、设置过滤规则

3、捕获数据

4、关闭网络设备

**抓包详细步骤**

首先要使用 libpcap，我们必须包含 pcap.h 头文件，可以在 /usr/local/include/pcap/pcap.h 找到，其中包含了每个类型定义的详细说明。



1、获取网络接口设备名

char \*pcap\_lookupdev(char \*errbuf);

功能：

得到可用的网络设备名指针

参数：

errbuf：存放出错信息字符串，里面有个宏定义：PCAP\_ERRBUF\_SIZE，为错误缓冲区大小

返回值：

成功返回设备名指针（第一个合适的网络接口的字符串指针）；

失败返回 NULL，同时，errbuf 存放出错信息字符串。

实例如下：

char error\_content[PCAP\_ERRBUF\_SIZE] = {0}; // 出错信息

char \*dev = pcap\_lookupdev(error\_content);

if(NULL == dev)

{

    printf(error\_content);

    exit(-1);

}

2、获取网络号（ip 地址）和掩码

int pcap\_lookupnet( char \*device,

bpf\_u\_int32 \*netp,

bpf\_u\_int32 \*maskp,

char \*errbuf  );

功能：

获取指定网卡的 ip 地址，子网掩码

参数：

device：网络设备名，为第一步获取的网络接口字符串（pcap\_lookupdev() 的返回值 ），也可人为指定，如“eth0”。

netp：存放 ip 地址的指针，bpf\_u\_int32 为 32 位无符号整型

maskp：存放子网掩码的指针，bpf\_u\_int32 为 32 位无符号整型

errbuf：存放出错信息

返回值：

成功返回 0，失败返回 -1

实例如下：

char error\_content[PCAP\_ERRBUF\_SIZE] = {0}; // 出错信息

char \*dev = pcap\_lookupdev(error\_content);

if(NULL == dev)

{

    printf(error\_content);

    exit(-1);

}

bpf\_u\_int32 netp = 0, maskp = 0;

pcap\_t \* pcap\_handle = NULL;

int ret = 0;

//获得网络号和掩码

ret = pcap\_lookupnet(dev, &netp, &maskp, error\_content);

if(ret == -1)

{

    printf(error\_content);

    exit(-1);

}

3、打开网络接口

pcap\_t \*pcap\_open\_live( const char \*device,

int snaplen,

int promisc,

int to\_ms,

char \*ebuf );

功能：

打开一个用于捕获数据的网络接口

参数：

device：网络接口的名字，为第一步获取的网络接口字符串（pcap\_lookupdev() 的返回值 ），也可人为指定，如“eth0”。

snaplen：捕获数据包的长度，长度不能大于 65535 个字节。

promise：“1” 代表混杂模式，其它非混杂模式。什么为混杂模式

to\_ms：指定需要等待的毫秒数，超过这个数值后，获取数据包的函数就会立即返回（这个函数不会阻塞，后面的抓包函数才会阻塞）。0 表示一直等待直到有数据包到来。

ebuf：存储错误信息。

返回值：

返回 pcap\_t 类型指针，后面的所有操作都要使用这个指针。

实例如下：

char error\_content[PCAP\_ERRBUF\_SIZE] = {0}; // 出错信息

char \*dev = pcap\_lookupdev(error\_content);  // 获取网络接口

if(NULL == dev)

{

    printf(error\_content);

    exit(-1);

}

// 打开网络接口

pcap\_t \* pcap\_handle = pcap\_open\_live(dev, 1024, 1, 0, error\_content);

if(NULL == pcap\_handle)

{

    printf(error\_content);

    exit(-1);

}

4、获取数据包

a）

const u\_char \*pcap\_next(pcap\_t \*p, struct pcap\_pkthdr \*h);

功能：

捕获一个网络数据包，收到一个数据包立即返回

参数：

p：pcap\_open\_live()返回的 pcap\_t 类型的指针  
 h：数据包头

pcap\_pkthdr 类型的定义如下：

struct pcap\_pkthdr

{

struct timeval ts; // 抓到包的时间

bpf\_u\_int32 caplen; // 表示抓到的数据长度

bpf\_u\_int32 len; // 表示数据包的实际长度

}

len 和 caplen的区别：

因为在某些情况下你不能保证捕获的包是完整的，例如一个包长 1480，但是你捕获到 1000 的时候，可能因为某些原因就中止捕获了，所以 caplen 是记录实际捕获的包长，也就是 1000，而 len 就是 1480。

返回值：

成功返回捕获数据包的地址，失败返回 NULL

实例如下：

const unsigned char \*p\_packet\_content = NULL; // 保存接收到的数据包的起始地址

pcap\_t \*pcap\_handle = NULL;

struct pcap\_pkthdr protocol\_header;

pcap\_handle = pcap\_open\_live("eth0", 1024, 1, 0,NULL);

p\_packet\_content = pcap\_next(pcap\_handle, &protocol\_header);

//p\_packet\_content  所捕获数据包的地址

printf("Capture Time is :%s",ctime((const time\_t \*)&protocol\_header.ts.tv\_sec)); // 时间

printf("Packet Lenght is :%d\n",protocol\_header.len);   // 数据包的实际长度

// 分析以太网中的 源mac、目的mac

struct ether\_header \*ethernet\_protocol = NULL;

unsigned char \*p\_mac\_string = NULL;         // 保存mac的地址，临时变量

ethernet\_protocol = (struct ether\_header \*)p\_packet\_content;  //struct ether\_header 以太网帧头部

p\_mac\_string = (unsigned char \*)ethernet\_protocol->ether\_shost;//获取源mac

printf("Mac Source Address is %02x:%02x:%02x:%02x:%02x:%02x\n",\*(p\_mac\_string+0),\*(p\_mac\_string+1),\*(p\_mac\_string+2),\*(p\_mac\_string+3),\*(p\_mac\_string+4),\*(p\_mac\_string+5));

p\_mac\_string = (unsigned char \*)ethernet\_protocol->ether\_dhost;//获取目的mac

printf("Mac Destination Address is %02x:%02x:%02x:%02x:%02x:%02x\n",\*(p\_mac\_string+0),\*(p\_mac\_string+1),\*(p\_mac\_string+2),\*(p\_mac\_string+3),\*(p\_mac\_string+4),\*(p\_mac\_string+5));

b）

int pcap\_loop( pcap\_t \*p,

int cnt,

pcap\_handler callback,

u\_char \*user );

功能：

循环捕获网络数据包，直到遇到错误或者满足退出条件。每次捕获一个数据包就会调用 callback 指定的回调函数，所以，可以在回调函数中进行数据包的处理操作。

参数：

p：pcap\_open\_live()返回的 pcap\_t 类型的指针。

cnt：指定捕获数据包的个数，一旦抓到了 cnt 个数据包，pcap\_loop 立即返回。如果是 -1，就会永无休止的捕获，直到出现错误。

callback：回调函数，名字任意，根据需要自行起名。

user：向回调函数中传递的参数。

callback 回调函数的定义：

void callback(  u\_char \*userarg,

const struct pcap\_pkthdr \* pkthdr,

const u\_char \* packet )

userarg：pcap\_loop() 的最后一个参数，当收到足够数量的包后 pcap\_loop 会调用callback 回调函数，同时将pcap\_loop()的user参数传递给它  
 pkthdr：是收到数据包的 pcap\_pkthdr 类型的指针，和 pcap\_next() 第二个参数是一样的。  
 packet ：收到的数据包数据

返回值：

成功返回0，失败返回负数

实例如下：

if( pcap\_loop(pcap\_handle, -1, ethernet\_protocol\_callback, NULL) < 0 )

{

    perror("pcap\_loop");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*回调函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void ethernet\_protocol\_callback(unsigned char \*argument,const struct pcap\_pkthdr \*packet\_heaher,const unsigned char \*packet\_content)

{

    unsigned char \*mac\_string;              //

    struct ether\_header \*ethernet\_protocol;

    unsigned short ethernet\_type;           //以太网类型

    printf("----------------------------------------------------\n");

    printf("%s\n", ctime((time\_t \*)&(packet\_heaher->ts.tv\_sec))); //转换时间

    ethernet\_protocol = (struct ether\_header \*)packet\_content;

    mac\_string = (unsigned char \*)ethernet\_protocol->ether\_shost;//获取源mac地址

    printf("Mac Source Address is %02x:%02x:%02x:%02x:%02x:%02x\n",\*(mac\_string+0),\*(mac\_string+1),\*(mac\_string+2),\*(mac\_string+3),\*(mac\_string+4),\*(mac\_string+5));

    mac\_string = (unsigned char \*)ethernet\_protocol->ether\_dhost;//获取目的mac

    printf("Mac Destination Address is %02x:%02x:%02x:%02x:%02x:%02x\n",\*(mac\_string+0),\*(mac\_string+1),\*(mac\_string+2),\*(mac\_string+3),\*(mac\_string+4),\*(mac\_string+5));

    ethernet\_type = ntohs(ethernet\_protocol->ether\_type);//获得以太网的类型

    printf("Ethernet type is :%04x\n",ethernet\_type);

    switch(ethernet\_type)

    {

        case 0x0800:printf("The network layer is IP protocol\n");break;//ip

        case 0x0806:printf("The network layer is ARP protocol\n");break;//arp

        case 0x0835:printf("The network layer is RARP protocol\n");break;//rarp

        default:break;

    }

    usleep(800\*1000);

}

c）

int pcap\_dispatch(pcap\_t \* p, int cnt, pcap\_handler callback, u\_char \* user);

这个函数和 pcap\_loop() 非常类似，只是在超过 to\_ms 毫秒后就会返回( to\_ms 是pcap\_open\_live() 的第4个参数 )

5、释放网络接口

void pcap\_close(pcap\_t \*p);

功能：

关闭 pcap\_open\_live() 打开的网络接口（即其返回值，pcap\_t 类型指针），并释放相关资源。注意，操作完网络接口，应该释放其资源。

参数：

p：需要关闭的网络接口，pcap\_open\_live() 的返回值（pcap\_t 类型指针）

返回值：

无

实例如下：

// 打开网络接口

pcap\_t \* pcap\_handle = pcap\_open\_live("eth0", 1024, 1, 0, error\_content);

if(NULL == pcap\_handle)

{

    printf(error\_content);

    exit(-1);

}

//// ……

//// ……

pcap\_close(pcap\_handle); //释放网络接口

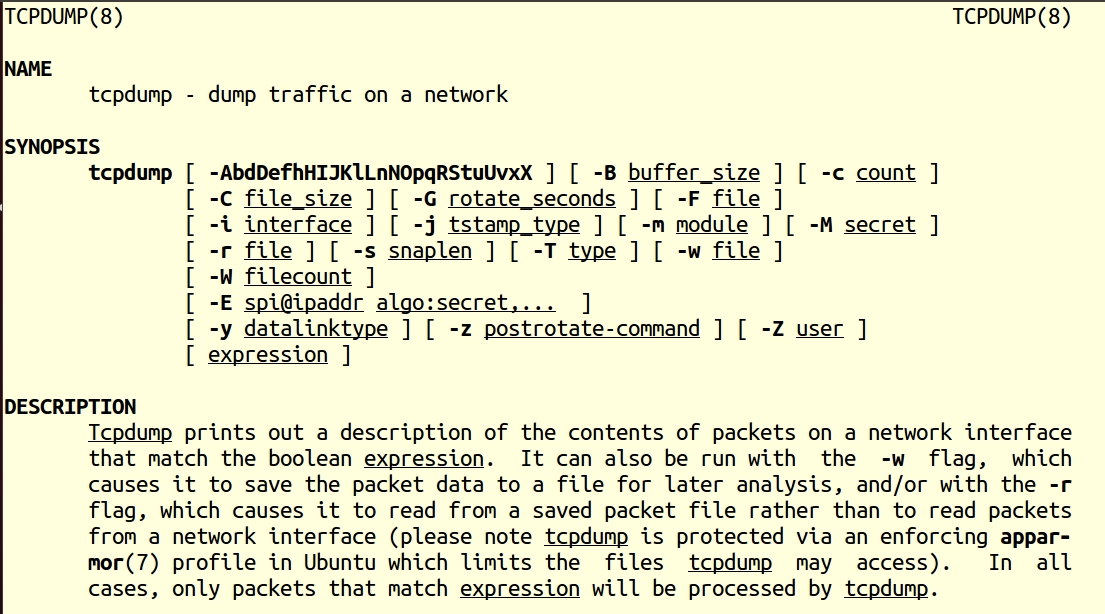
注意：gcc 编译时需要加上 -lpcap，运行时需要使用超级权限

**过滤数据包**

我们抓到的数据包往往很多，如何过滤掉我们不感兴趣的数据包呢?

libpcap 利用 BPF 来过滤数据包。

1）设置过滤条件  
BPF 使用一种类似于汇编语言的语法书写过滤表达式，不过 libpcap 和 tcpdump 都把它封装成更高级且更容易的语法了，具体可以通过 man tcpdump查看：

IMG_260  
  
  
以下是一些例子：

src host 192.168.1.177

只接收源 ip 地址是 192.168.1.177 的数据包

dst port 80

只接收 tcp/udp 的目的端口是 80 的数据包

not tcp

只接收不使用 tcp 协议的数据包

tcp[13] == 0x02 and (dst port 22 or dst port 23)

只接收 SYN 标志位置位且目标端口是 22 或 23 的数据包（ tcp 首部开始的第 13 个字节）

icmp[icmptype] == icmp-echoreply or icmp[icmptype] == icmp-echo

只接收 icmp 的 ping 请求和 ping 响应的数据包

ehter dst 00:e0:09:c1:0e:82

只接收以太网 mac 地址是 00:e0:09:c1:0e:82 的数据包

ip[8] == 5

只接收 ip 的 ttl=5 的数据包（ip首部开始的第8个字节）

2）编译 BPF 过滤规则

int pcap\_compile( pcap\_t \*p,

struct bpf\_program \*fp,

char \*buf,

int optimize,

bpf\_u\_int32 mask );

功能：

编译 BPF 过滤规则

参数：

p：pcap\_open\_live() 返回的 pcap\_t 类型的指针

fp：存放编译后的 bpf，应用过滤规则时需要用到这个指针

buf：过滤条件

optimize：是否需要优化过滤表达式

mask：指定本地网络的网络掩码，不需要时可写 0

返回值：

成功返回 0，失败返回 -1

3）应用 BPF 过滤规则

int pcap\_setfilter( pcap\_t \* p,  struct bpf\_program \* fp );

功能：

应用 BPF 过滤规则，简单理解为让过滤生效

参数：

p：pcap\_open\_live() 返回的 pcap\_t 类型的指针

fp：pcap\_compile() 的第二个参数

返回值：

成功返回 0，失败返回 -1

这个编译应用过程，有点类似于，我们写一个 C 程序，先编译，后运行的过程。

应用完过滤表达式之后我们便可以使用 pcap\_loop() 或 pcap\_next() 等抓包函数来抓包了。

下面的程序演示了如何过滤数据包，我们只接收目的端口是 80 的数据包：

#include <pcap.h>

#include <time.h>

#include <stdlib.h>

#include <stdio.h>

void getPacket(u\_char \* arg, const struct pcap\_pkthdr \* pkthdr, const u\_char \* packet)

{

  int \* id = (int \*)arg;

  printf("id: %d\n", ++(\*id));

  printf("Packet length: %d\n", pkthdr->len);

  printf("Number of bytes: %d\n", pkthdr->caplen);

  printf("Recieved time: %s", ctime((const time\_t \*)&pkthdr->ts.tv\_sec));

  int i;

  for(i=0; i<pkthdr->len; ++i)

  {

    printf(" %02x", packet[i]);

    if( (i + 1) % 16 == 0 )

    {

      printf("\n");

    }

  }

  printf("\n\n");

}

int main()

{

  char errBuf[PCAP\_ERRBUF\_SIZE], \* devStr;

  /\* get a device \*/

  devStr = pcap\_lookupdev(errBuf);

  if(devStr)

  {

    printf("success: device: %s\n", devStr);

  }

  else

  {

    printf("error: %s\n", errBuf);

    exit(1);

  }

  /\* open a device, wait until a packet arrives \*/

  pcap\_t \* device = pcap\_open\_live(devStr, 65535, 1, 0, errBuf);

  if(!device)

  {

    printf("error: pcap\_open\_live(): %s\n", errBuf);

    exit(1);

  }

  /\* construct a filter \*/

  struct bpf\_program filter;

  pcap\_compile(device, &filter, "dst port 80", 1, 0);

  pcap\_setfilter(device, &filter);

  /\* wait loop forever \*/

  int id = 0;

  pcap\_loop(device, -1, getPacket, (u\_char\*)&id);

  pcap\_close(device);

  return 0;

}