C语言实现 Linux 网络嗅探器（BirdSniffer）

by: bird

1.项目介绍

网络嗅探器是拦截通过网络接口流入和流出的数据的程序。比如你正在浏览的互联网，嗅探器以数据包的形式抓到它并且显示给拦截者，这是常用的黑客工具，比较著名的有 wireshark，Burpsuit。

在本本次项目中，我将用 C 语言实现了一个网络嗅探器：BirdSniffer

2.基础知识

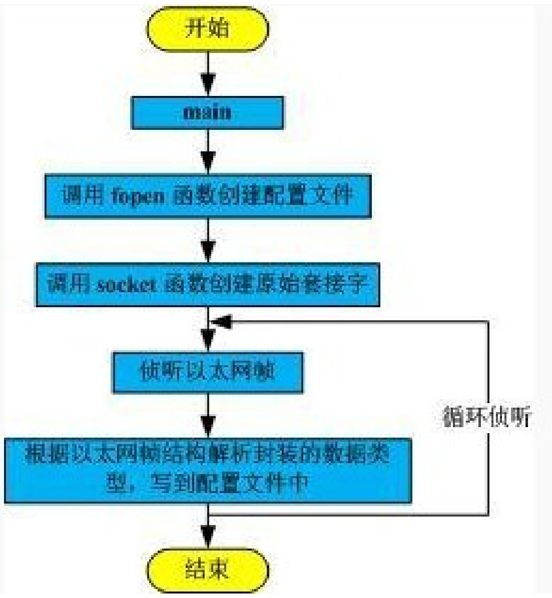
原始套接字

以太网帧结构

IP 数据报结构

3.开发步骤

本项目的主框架如下：



1）原始套接字

原始套接字的创建

只有超级用户才能创建原始套接字：

```
int sockfd;
sockfd = socket(PF_PACKET, SOCK_RAW, protocol);
```

利用原始套接字访问数据链路层

通过下面语句获得负载为 IP 数据报的以太网帧：

```
sd = socket(PF_PACKET, SOCK_RAW,htons(ETH_P_IP));
```

2)main函数

1 创建日志文件

以可写的方式在当前文件夹中创建日志文件：

```
sniffer.logfile = fopen("log.txt", "w");
    fprintf(sniffer.logfile,"***LOGFILE(%s - %s)***\n", __DATE__,
__TIME__);
    if (sniffer.logfile == NULL)
    {
        perror("fopen(): ");
        return (EXIT_FAILURE);
    }
```

2 创建原始套接字监听所有的数据链路层帧

创建原始套接字，ETH_P_ALL 表示侦听负载为 IP 数据报的以太网帧：

```
sd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP));
if (sd < 0)
    {
        perror("socket(): ");
```

```
        return (EXIT_FAILURE);
    }
```

3 循环侦听以太网帧，并调用 ProcessPacket 函数解析

首先设置 select 监听的描述符集：

```
FD_ZERO(&fd_read);
FD_SET(0, &fd_read);
FD_SET(sd, &fd_read);
```

多路复用检测可读的套接字和标准输入：

```
res = select(sd + 1, &fd_read, NULL, NULL, NULL);
```

如果是套接字可读，则读取以太网数据帧的内容：

```
多路复用检测可读的套接字和标准输入：
res = select(sd + 1, &fd_read, NULL, NULL, NULL);
如果是套接字可读，则读取以太网数据帧的内容：
saddr_size = sizeof(saddr);
data_size = recvfrom(sd, buffer, 65536, 0, &saddr,
(socklen_t*)&saddr_size); /* 读取以太网数据帧的内容 */
if (data_size <= 0)
    {
        close(sd);
        perror("recvfrom(): ");
        return (EXIT_FAILURE);
    }
调用 ProcessPacket 函数解析出数据包的类型：
ProcessPacket(buffer, data_size, &sniffer);
```

调用 ProcessPacket 函数解析出数据包的类型：

```
ProcessPacket(buffer, data_size, &sniffer);
```

```
```

这部分的完整代码如下：

```
/* 主函数入口 */
int     main()
{
    /* 声明部分 */
    int     sd;
    int     res;
    int     saddr_size;
    int     data_size;
    struct sockaddr saddr;
    unsigned char *buffer; /* 保存数据包的数据 */
    t_sniffer sniffer; /* 保存数据包的类型和日志文件等信息 */
    fd_set fd_read;

    buffer = malloc(sizeof(unsigned char *) * 65536);

    /* 以可写的方式在当前文件夹中创建日志文件 */
    sniffer.logfile = fopen("log.txt", "w");
    fprintf(sniffer.logfile,"***LOGFILE(%s - %s)***\n", __DATE__,
__TIME__);
    if (sniffer.logfile == NULL)
    {
        perror("fopen(): ");
        return (EXIT_FAILURE);
    }

    sniffer.prot = malloc(sizeof(t_protocol *));

    /* 创建原始套接字，ETH_P_ALL 表示侦听负载为 IP 数据报的以太网帧 */
    sd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP));
    if (sd < 0)
    {
        perror("socket(): ");
        return (EXIT_FAILURE);
    }
    getting_started();
    signal(SIGINT, &signal_white_now);
    signal(SIGQUIT, &signal_white_now);
```

```
    /* 循环侦听以太网帧，并调用 ProcessPacket 函数解析 */
    while (1)
    {
        FD_ZERO(&fd_read);
        FD_SET(0, &fd_read);
        FD_SET(sd, &fd_read);

        /* 多路复用检测可读的套接字和标准输入 */
        res = select(sd + 1, &fd_read, NULL, NULL, NULL);
        if (res < 0)
            {
                close(sd);
                if (errno != EINTR)
                perror("select() ");
                return (EXIT_FAILURE);
            }
        else
            {
                /* 如果是标准输入可读，进入命令行处理程序 command_interpreter，暂时只支持 'quit' 命令 */
                if (FD_ISSET(0, &fd_read))
                {
                    if (command_interpreter(sd) == 1)
                    break;
                }

                /* 如果是套接字可读，则读取以太网数据帧的内容，并调用 ProcessPacket 函数解析出数据包的类型 */
                else if (FD_ISSET(sd, &fd_read))
                    {
                        /* 读取以太网数据帧的内容 */
                        saddr_size = sizeof(saddr);
                        data_size = recvfrom(sd, buffer, 65536, 0, &saddr,(socklen_t*)&saddr_size); /* 读取以太网数据帧的内容 */
                        if (data_size <= 0)
                            {
                                close(sd);
                                perror("recvfrom(): ");
                                return (EXIT_FAILURE);
                            }

```

```
                        ProcessPacket(buffer, data_size, &sniffer); /* 调用 ProcessPacket 函数解析出数据包的类型 */
                    }
            }
    }

    close(sd);
    return (EXIT_SUCCESS);
}
```

3)ProcessPocket函数解析以太网数据帧

1 分析以太网帧结构，分离出 IP 数据报

以太网帧结构如下：

MAC 目标地址	MAC 源地址	802.1Q 标签 (可选)	以太长度	负载	冗余校验	帧间距
6 octets	6 octets	(4 octets)	2 octets	46－1500 octets	4 octets	12 octets

根据太网帧结构，前 6B 是目的 MAC 地址，接下来的是源 MAC 地址，接下来 2B 是帧长度，其余的是负载（上层的 IP 数据报），所以将指针 buffer 加上 6 + 6 + 2 便指向 IP 数据报的首地址：

```
buffer = buffer + 6 + 6 + 2;
```

2 获取 IP 数据报头

此时 buffer 指向 IP 数据报的头部，所以强制类型转换为指向 iphdr结构的指针：

```
struct iphdr *iph = (struct iphdr*)buffer;
```

3 判断 IP 负载的类型

根据 TCP/IP 协议规定的 IP 数据报头部的 protocol 字段的值，可以判断 IP 数据报负载的数据类型，其中，IP 协议规定，1 表示 ICMP 协议；2 表示 IGMP 协议；6 表示 TCP 协议；17 表示 UDP 协议：

```c
switch (iph->protocol)
        {
            /* 1 表示 icmp 协议 */
            case 1:
                ++sniffer->prot->icmp;
                print_icmp_packet(buffer, size, sniffer);
                break;

            /* 2 表示 igmp 协议 */
            case 2:
                ++sniffer->prot->igmp;
                break;

            /* 6 表示 tcp 协议 */
            case 6:
                ++sniffer->prot->tcp;
                print_tcp_packet(buffer , size, sniffer);
                break;

            /* 17 表示 udp 协议 */
            case 17:
                ++sniffer->prot->udp;
                print_udp_packet(buffer , size, sniffer);
                break;

            default:
                ++sniffer->prot->others;
                break;
        }
```

这部分的完整代码：

```c
void ProcessPacket(unsigned char* buffer, int size, t_sniffer *sniffer)
{
    buffer = buffer + 6 + 6 + 2; /* 根据以太网帧结构，前 6B 是目的 MAC 地址，接下来的是源 MAC 地址，接下来 2B 是帧长度，其余的是负载（上层的 IP 数据报） */
    struct iphdr *iph = (struct iphdr*)buffer;
```

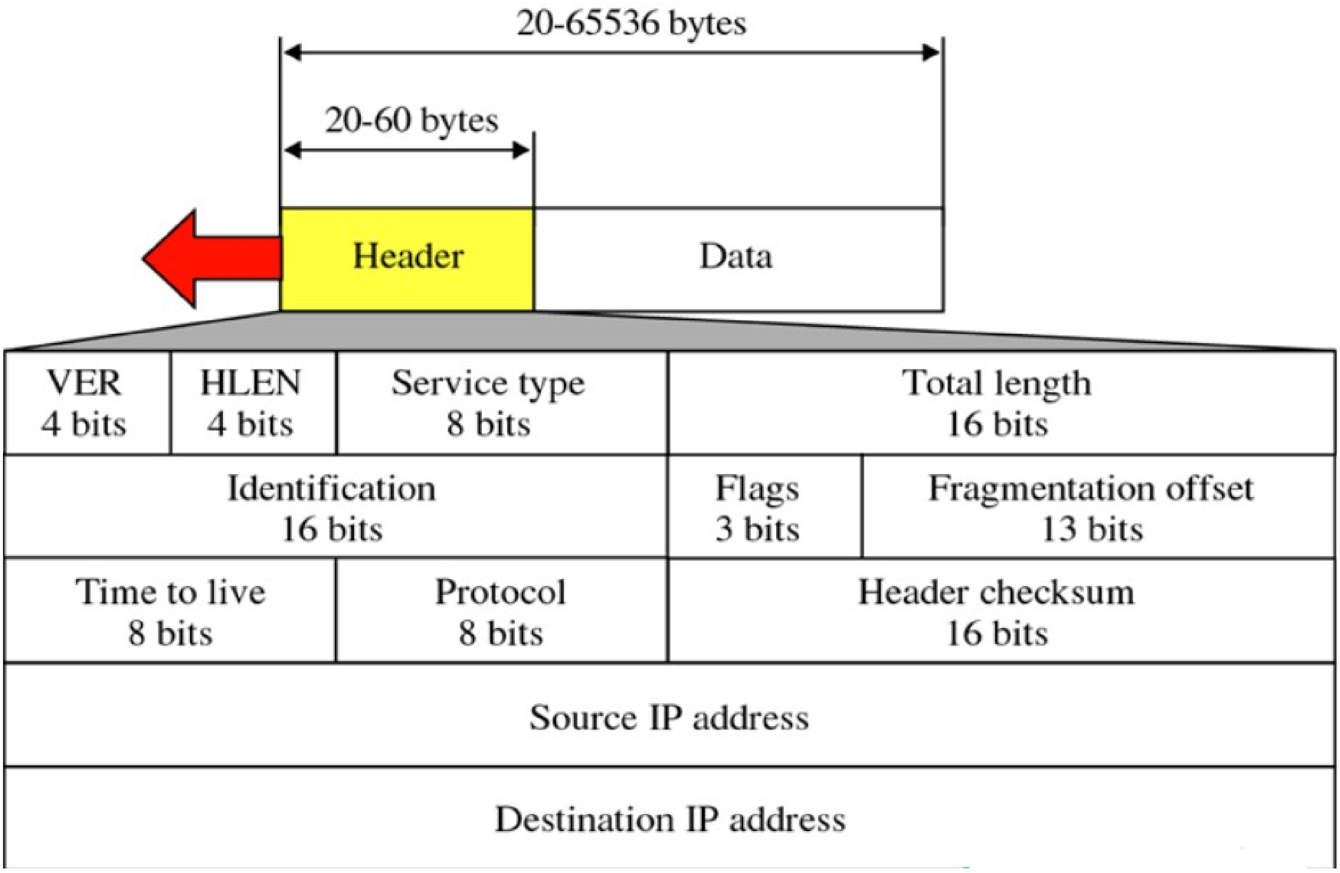
```c
    ++sniffer->prot->total; /* 数据包总数加 1 */

    /* 根据 TCP/IP 协议规定的 IP 数据报头部的 protocol 字段的值，判断上层的数据包类型 */
    switch (iph->protocol)
        {
            /* 1 表示 icmp 协议 */
            case 1:
                ++sniffer->prot->icmp;
                print_icmp_packet(buffer, size, sniffer);
                break;

            /* 2 表示 igmp 协议 */
            case 2:
                ++sniffer->prot->igmp;
                break;

            /* 6 表示 tcp 协议 */
            case 6:
                ++sniffer->prot->tcp;
                print_tcp_packet(buffer , size, sniffer);
                break;

            /* 17 表示 udp 协议 */
            case 17:
                ++sniffer->prot->udp;
                print_udp_packet(buffer , size, sniffer);
                break;

            default:
                ++sniffer->prot->others;
                break;
        }

    display_time_and_date(); /* 显示时间 */

    /* 打印 sniffer 中的信息 */
    printf("TCP : %d   UDP : %d   ICMP : %d   IGMP : %d   Others : %d   Total : %d\n",
     sniffer->prot->tcp, sniffer->prot->udp,
     sniffer->prot->icmp, sniffer->prot->igmp,
     sniffer->prot->others, sniffer->prot->total);
```

```
}
```

4) 写入日志文件

1 写 IP 头部到日志文件

IP 数据包头格式如下：



首先应该根据 IP 数据报的获取 IP头部：

```
iph = (struct iphdr *)buf;
```

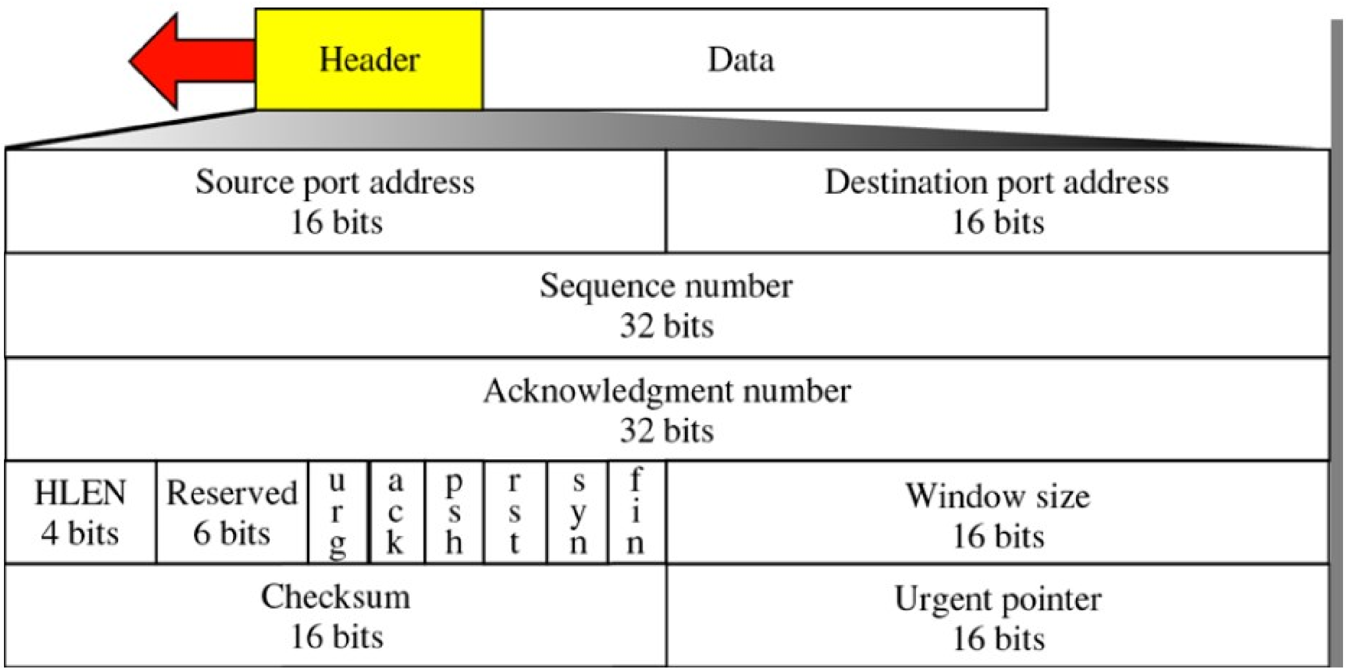
然后将头部的信息分别输入到配置文件中：

```
fprintf(sniffer->logfile,"\n");
    fprintf(sniffer->logfile,"IP Header\n");
    fprintf(sniffer->logfile,"    |-IP Version        : %d\n",(unsigned int)iph->version);
    fprintf(sniffer->logfile,"    |-IP Header Length  : %d DWORDS or %d Bytes\n",(unsigned int)iph->ihl,((unsigned int)(iph->ihl))*4);
    fprintf(sniffer->logfile,"    |-Type Of Service   : %d\n",(unsigned
```

```
int)iph->tos);
    fprintf(sniffer->logfile,"   |-IP Total Length   : %d  Bytes(size of Packet)\n",ntohs(iph->tot_len));
    fprintf(sniffer->logfile,"   |-Identification    : %d\n",ntohs(iph->id));
    fprintf(sniffer->logfile,"   |-TTL      : %d\n",(unsigned int)iph->ttl);
    fprintf(sniffer->logfile,"   |-Protocol : %d\n",(unsigned int)iph->protocol);
    fprintf(sniffer->logfile,"   |-Checksum : %d\n",ntohs(iph->check));
    fprintf(sniffer->logfile,"   |-Source IP        : %s\n",inet_ntoa(source.sin_addr));
    fprintf(sniffer->logfile,"   |-Destination IP   : %s\n",inet_ntoa(dest.sin_addr));
```

2 写 TCP 数据包到日志文件

TCP 数据包头格式如下：



首先应该根据 IP 数据报的获取 TCP 头部：

```
iph = (struct iphdr *)buf;
    iphdrlen = iph->ihl * 4;
    tcph = (struct tcphdr*)(buf + iphdrlen);
```

然后将头部的信息分别输入到配置文件中：

```
fprintf(sniffer->logfile,"\n");
fprintf(sniffer->logfile,"TCP Header\n");
fprintf(sniffer->logfile,"   |-Source Port      : %u\n",ntohs(tcph->source));
fprintf(sniffer->logfile,"   |-Destination Port : %u\n",ntohs(tcph->dest));
fprintf(sniffer->logfile,"   |-Sequence Number    : %u\n",ntohl(tcph->seq));
fprintf(sniffer->logfile,"   |-Acknowledge Number : %u\n",ntohl(tcph->ack_seq));
fprintf(sniffer->logfile,"   |-Header Length      : %d DWORDS or %d BYTES\n" ,(unsigned int)tcph->doff,(unsigned int)tcph->doff*4);
fprintf(sniffer->logfile,"   |-Urgent Flag          : %d\n",(unsigned int)tcph->urg);
fprintf(sniffer->logfile,"   |-Acknowledgement Flag : %d\n",(unsigned int)tcph->ack);
fprintf(sniffer->logfile,"   |-Push Flag            : %d\n",(unsigned int)tcph->psh);
fprintf(sniffer->logfile,"   |-Reset Flag           : %d\n",(unsigned int)tcph->rst);
fprintf(sniffer->logfile,"   |-Synchronise Flag     : %d\n",(unsigned int)tcph->syn);
fprintf(sniffer->logfile,"   |-Finish Flag          : %d\n",(unsigned int)tcph->fin);
fprintf(sniffer->logfile,"   |-Window         : %d\n",ntohs(tcph->window));
fprintf(sniffer->logfile,"   |-Checksum       : %d\n",ntohs(tcph->check));
fprintf(sniffer->logfile,"   |-Urgent Pointer : %d\n",tcph->urg_ptr);
fprintf(sniffer->logfile,"\n");
fprintf(sniffer->logfile,"                        DATA Dump                        ");
fprintf(sniffer->logfile,"\n");

fprintf(sniffer->logfile,"IP Header\n");
PrintData(buf, iphdrlen, sniffer);

fprintf(sniffer->logfile,"TCP Header\n");
```

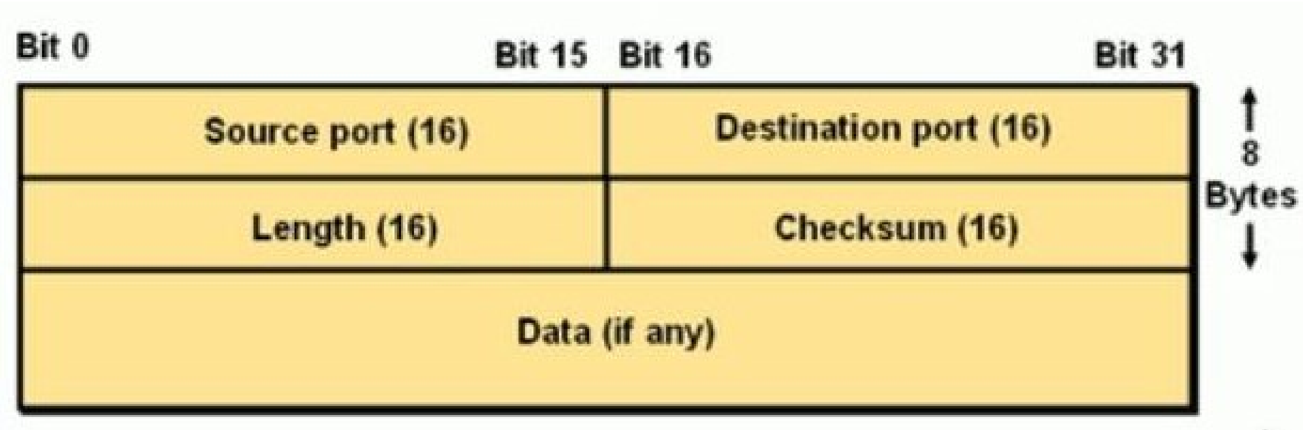
```
PrintData(buf+iphdrlen, tcph->doff*4, sniffer);

fprintf(sniffer->logfile,"Data Payload\n");
```

最后将用户数据写入日志文件中：

```
PrintData(buf + iphdrlen + tcph->doff*4,
        (size - tcph->doff*4-iph->ihl*4),
        sniffer );
```

3 写 UDP 数据包到日志文件

UDP 数据包头格式如下：



首先应该根据 IP 数据报的获取 UDP 头部：

```
iph = (struct iphdr *)buf;
    iphdrlen = iph->ihl*4;
    udph = (struct udphdr*)(buf + iphdrlen);
```

然后将头部的信息分别输入到配置文件中：

```
fprintf(sniffer->logfile,"\nUDP Header\n");
fprintf(sniffer->logfile,"   |-Source Port      : %d\n" , ntohs(udph->source));
fprintf(sniffer->logfile,"   |-Destination Port : %d\n" , ntohs(udph-
```

```
>dest));
fprintf(sniffer->logfile,"   |-UDP Length       : %d\n" , ntohs(udph->len));
fprintf(sniffer->logfile,"   |-UDP Checksum     : %d\n" , ntohs(udph->check));

fprintf(sniffer->logfile,"\n");
fprintf(sniffer->logfile,"IP Header\n");
PrintData(buf , iphdrlen, sniffer);

fprintf(sniffer->logfile,"UDP Header\n");
PrintData(buf+iphdrlen, sizeof(udph), sniffer);

fprintf(sniffer->logfile,"Data Payload\n");
```

最后将用户数据写入日志文件中:

```
PrintData(buf + iphdrlen + sizeof udph,
        (size - sizeof udph - iph->ihl * 4),
        sniffer);
```

4 写 ICMP 数据包到日志文件

ICMP 数据包头格式如下:

首先应该根据 IP 数据报的获取 ICMP 头部:

```
iph = (struct iphdr *)buf;
    iphdrlen = iph->ihl * 4;
    icmph = (struct icmphdr *)(buf + iphdrlen);
```

把 ICMP 头信息写入日志文件中:

```c
fprintf(sniffer->logfile,"\n\n***********************ICMP Packet*************************\n");
print_ip_header(buf , size, sniffer);
fprintf(sniffer->logfile,"\n");
fprintf(sniffer->logfile,"ICMP Header\n");
fprintf(sniffer->logfile,"   |-Type : %d",(unsigned int)(icmph->type));
if((unsigned int)(icmph->type) == 11)
fprintf(sniffer->logfile,"  (TTL Expired)\n");
else if((unsigned int)(icmph->type) == ICMP_ECHOREPLY)
fprintf(sniffer->logfile,"  (ICMP Echo Reply)\n");
fprintf(sniffer->logfile,"   |-Code : %d\n",(unsigned int)(icmph->code));
fprintf(sniffer->logfile,"   |-Checksum : %d\n",ntohs(icmph->checksum));
fprintf(sniffer->logfile,"\n");
fprintf(sniffer->logfile,"IP Header\n");
PrintData(buf, iphdrlen, sniffer);
fprintf(sniffer->logfile,"UDP Header\n");
PrintData(buf + iphdrlen , sizeof(icmph), sniffer);

fprintf(sniffer->logfile,"Data Payload\n");
```

最后将用户数据写入日志文件中：

```c
PrintData(buf + iphdrlen + sizeof(icmph),
        (size - sizeof(icmph) - iph->ihl * 4),
        sniffer);
```

4. 项目测试

```
┌─root@kali ~/C_Network_Sniffer_LINUX-master
└─➤ gcc -c show_data.c
┌─root@kali ~/C_Network_Sniffer_LINUX-master
└─➤ gcc -c tools.c
┌─root@kali ~/C_Network_Sniffer_LINUX-master
└─➤ gcc -c main.c
┌─root@kali ~/C_Network_Sniffer_LINUX-master
└─➤ ls
launcher.sh  main.c  main.o  Makefile  README.md  show_data.c  show_data.o  sniffer.h  tools.c  tools.h  tools.o
┌─root@kali ~/C_Network_Sniffer_LINUX-master
└─➤ gcc main.o show_data.o tools.o -o BirdSniffer
┌─root@kali ~/C_Network_Sniffer_LINUX-master
└─➤ ls
BirdSniffer  launcher.sh  main.c  main.o  Makefile  README.md  show_data.c  show_data.o  sniffer.h  tools.c  tools.h  tools.o
┌─root@kali ~/C_Network_Sniffer_LINUX-master
└─➤ ./BirdSniffer

[Nov 22 2018][23:37:32]  Getting started of Network sniffer

[Nov 22 2018][23:37:32]  TCP : 0   UDP : 1   ICMP : 0   IGMP : 0   Others : 0 Total : 1
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 2   ICMP : 0   IGMP : 0   Others : 0 Total : 2
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 3   ICMP : 0   IGMP : 0   Others : 0 Total : 3
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 4   ICMP : 0   IGMP : 0   Others : 0 Total : 4
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 5   ICMP : 0   IGMP : 0   Others : 0 Total : 5
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 6   ICMP : 0   IGMP : 0   Others : 0 Total : 6
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 7   ICMP : 0   IGMP : 0   Others : 0 Total : 7
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 8   ICMP : 0   IGMP : 0   Others : 0 Total : 8
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 9   ICMP : 0   IGMP : 0   Others : 0 Total : 9
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 10   ICMP : 0   IGMP : 0   Others : 0 Total : 10
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 11   ICMP : 0   IGMP : 0   Others : 0 Total : 11
[Nov 22 2018][23:37:32]  TCP : 0   UDP : 12   ICMP : 0   IGMP : 0   Others : 0 Total : 12
```