目录

第一章　引言

第一节　编写目的

通过编写网络嗅探器，可以提高编程能力，加深对网络协议的理解，培养团队协作能力，完成课程项目

第二节　编写背景

1.2.1 系统名称及版本号

MySniff V1.5.0

第三节　文档概述

1.3.1 文档结构说明

文章结构分为编写背景，以及功能概述与功能详述

1.3.2 电子文档编写工具

WPS Office

1.3.3 参考资料

【说明】格式：作者，[版本号]，资料来源，日期，[起止页号]。其中，《需求规格说明书》与《概要设计说明书》是必选的参考资料。

第二章　功能概述

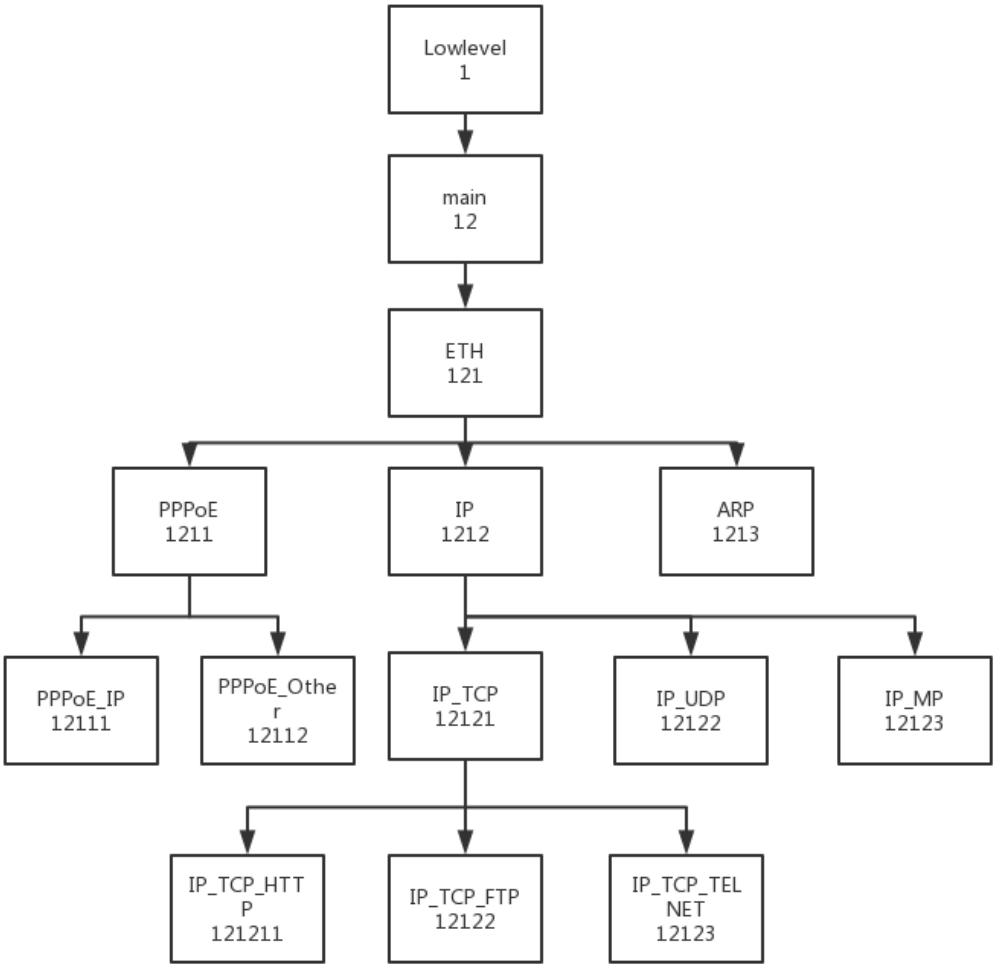
第一节　功能模块命名原则

使用英文进行命名

第二节　功能层次图

【底层模块】

LowLevel	1
main	12
eth	121
pppoe	1211
ip	1212
arp	1213
pppoe_ip	12111
pppoe_other	12112
ip_tcp	12121
ip_udp	12122
ip_mp	12123
ip_tcp_http	121211
ip_tcp_ftp	121212
ip_tcp_telnet	121213


Lowlevel
1
main
12
ETH
121
PPPoE
1211
IP
1212
ARP
1213
PPPoE_IP
12111
PPPoE_Othe
r
12112
IP_TCP
12121
IP_UDP
12122
IP_MP
12123
IP_TCP_HTT
P
121211
IP_TCP_FTP
12122
IP_TCP_TEL
NET
12123

第三章 功能模块详述

第一节 底层模块

3.1.1 模块编号与中文注释

【说明】应与第二章第二节的表述一致。
模块编号 1
中文注释 底层模块

3.1.2 功能描述与性能描述

【说明】功能描述：本叶模块的主要功能。
性能描述：精度指标、响应速度指标、数据吞吐量指标……
功能描述：本模块主要实现对流量数据包的抓取与分析，实现对各层协议的具体分析，能够捕获使用 HTTP、FTP、TELNET 协议传输的用户名和密码。
性能描述：精度能够捕获每一个数据包；响应速度目前能够计时分析每一个数据包 在 100 条 printf 打印/S 的流量下能够顺利完成；数据吞吐量能够在 10MB/S 环境顺利完成分析任务

3.1.3 与本模块相关的代码表和表

【说明】

名称	中文注释	类型		作用
		代码表	表	
main	主文件	✔		input
eth	以太网协议	✔		input
ip	ip 协议	✔		input
arp	arp 协议	✔		input
tcp	tcp 协议	✔		input/output
mp	icmp 和 igmp 协议	✔		input
udp	udp 协议	✔		output
protocol_based_tcp	基于 tcp 的协议	✔		output
protocol_based_udp	基于 udp 的协议	✔		output

作用：input、 output 、update 等。

3.1.4 输入信息

【说明】参数（含参数名、中文注释、缺省值、格式）、数据文件的格式与权限、输入频度。使用特殊输入设备情况。输入时使用代码表与基本表的情况。
基于 pacap_next_ex()函数的内存数据

3.1.5 输出信息

【说明】参数（含参数名、中文注释、缺省值、格式）、数据文件的格式、输出频度、报表格式样张。使用特殊输出设备情况。输出时使用代码表与基本表的情况。
协议的内容、输出在标准输出上。

3.1.6 算法

【说明】包括计算公式与说明、某些设定的或必然的逻辑关系。对于函数，要着重说明。

a) main.c

i. 代码思路

首先，对输入的参数，进行解析，提取用户的目的。

```
int ret = _parse(argc, argv);
if (ret < 0)
    return -1;
```

然后，选定网卡。这个过程中分为两部，根据输入的参数选定网卡，首先是获得所有网卡信息，然后遍历网卡信息，若与输入的网卡信息匹配，就将它选定。

```
pcap_if_t *dev = _get_device();
if (!dev)
    goto EXIT;

memset(&g_addrs, 0, sizeof(g_addrs));
_get_address(dev);
```

然后，确定过滤信息。根据用户的输入，传给相应函数的参数。

```
ret = _filter(dev, adhandle);
if (ret < 0)
    goto EXIT;

ret = link_create();
if (ret < 0)
    goto EXIT;
```

然后，开始抓取数据包。

```
_poll_device(adhandle);
```

主要涉及的函数：

_get_all_devices ()[return the name of dev]->pcap_open_live()[open dev-file to be ready read file]-> pcap_compile()[compile the rule]->pcap_setfilter()[configurate the rule of filtering]->pcap_loop()[loop to capture the datagram]->pacap_close()[close the lib]

b) eth.c

对 eth 头部信息进行处理，然后根据它上层的协议类型将它的 payload 部分交给不同的函数，传递时，将 payload 的首地址交给协议函数进行处理。在 eth.h 中定义了以太头结构体和 PPPoe 头结构体，方便对数据包进行分析，还声明了相关的函数。

```
typedef struct eth_head_t {
    uint8_t dst[6];
    uint8_t src[6];
    uint16_t type;
} eth_head_t;
```

```
typedef struct pppoe_head_t {
    uint8_t vTy;
#define PPPoE_V(pppoe) (((pppoe)->vhl & 0xf0) >> 4)
#define PPPoE_Ty(pppoe) ((pppoe)->vhl & 0x0f)
    uint8_t code;
    uint16_t session_id;
    uint16_t length;
    uint16_t protocol;
} pppoe_head_t;
```

```
#define ETH_IP 0x0800
#define ETH_ARP 0x0806
#define ETH_RARP 0x8035
#define ETH_PPPoE_Session 0x8864
#define ETH_PPPoE_Discovery 0x8863
```

```
#define PPP_IP 0x0021
#define PPP_LCP 0xC021
#define PPP_NCP 0x8021
#define PPP_PAP 0xC023
#define PPP_LQR 0xC025
#define PPP_CHAP 0xC223
```

```
int eth_parse(const unsigned char* data);
int pppoe_parse_S(const pppoe_head_t* pppoe);
int pppoe_parse_D(const pppoe_head_t* pppoe);
#endif
```

```c
int eth_parse(const unsigned char *data)
{
    const eth_head_t *eth = (eth_head_t *)data;
    uint16_t type = ntohs(eth->type);
    switch (type)
    {
    case ETH_IP:
        //printf("IP\n");
        return ip_parse((const ip_head_t *)(eth + 1));
    case ETH_ARP:
        return arp_parse((const arp_head_t *)(eth + 1));
    case ETH_RARP:
        return GAZE_ETH_NOT_SUPPORT;
    case ETH_PPPoE_Session:
        return pppoe_parse_S((const pppoe_head_t *)(eth + 1));
    case ETH_PPPoE_Discovery:
        return pppoe_parse_D((const pppoe_head_t *)(eth + 1));
    default:
        return GAZE_ETH_NOT_SUPPORT;
    }

    return GAZE_ETH_FAIL;
}
```

c) arp.c

主要是对 arp 协议的分析处理，首先在 arp.h 进行 arp 协议头部的结构体定义，然后定义 arp_prase 函数。在 arp_parse 函数中主要是获得源 MAC 的目的 MAC，源 IP 和目的 IP，然后打印他的 payload。没有对 arp 协议的 opcode 字段进行分析。

```c
typedef struct mac_addr_t {
    uint8_t data[6];
} mac_addr_t;

typedef struct arp_head_t {
    uint16_t hrd_type;
    uint16_t pro_type;
    uint8_t  hrd_addr_len;
    uint8_t  pro_addr_len;
    uint16_t opcode;
} arp_head_t;

int arp_parse(const arp_head_t* head);
```

```c
int arp_parse(const arp_head_t* head)
{
    int send=1;
    int receve=0;
    const char* buffer=(const char*)(head);
    mac_addr_t *sender_mac = ( mac_addr_t *)(head + 1);
    ip_addr_t  *sender_ip = (ip_addr_t *)(sender_mac+1);

    mac_addr_t *receiver_mac = ( mac_addr_t *)(sender_ip + 1);
    ip_addr_t  *receiver_ip = (ip_addr_t *)(receiver_mac + 1);

    printf("***************protacol:ARP*****************\n");
```

```
printf("***************protacol:ARP******************\n");

print_mac(send,sender_mac->data);
print_ip(send,sender_ip->data);

print_mac(receve,receiver_mac->data);
print_ip(receve,receiver_ip->data);
printf("\n\n");

_output_bytes(buffer,28);

return GAZE_OK;
```

d) ip.c

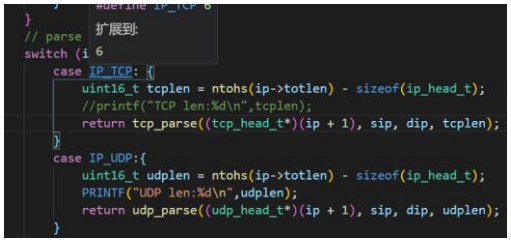
网络层就只处理 IP 协议。Ip_parse 函数首先检查是否是 IPV4，然后检查 IP 是否分片，如果分片了就不处理返回 NOT SOUPPORT。然后检查是否是到监听的网卡的包，然后在判断它的上层协议，最后在分别交给不同的协议函数处理，传参的时候还传递了长度等参数，这时要将网络字节顺序转换为主机字节顺序。在 ip.h 中定义了 ip 头部结构体，声明了 ip_parse 函数。

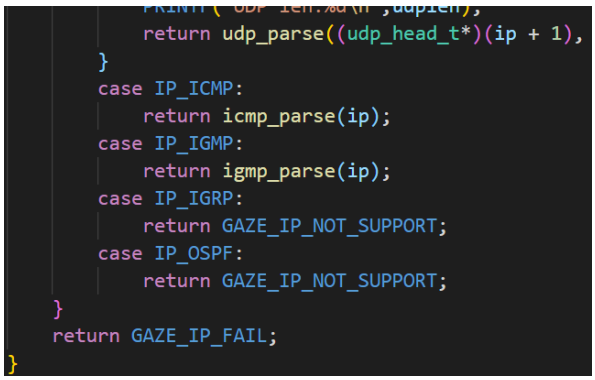
```
typedef struct ip_addr_t {
    uint8_t data[4];
} ip_addr_t;

typedef struct ip_head_t {
    uint8_t vhl;
#define IP_V(ip) (((ip)->vhl & 0xf0) >> 4)
#define IP_HL(ip) ((ip)->vhl & 0x0f)
    uint8_t tos;
    uint16_t totlen;
    uint16_t ident;
    uint16_t offset;
#define IP_DF 0x4000
```

```
#define IP_DF 0x4000
#define IP_MF 0x2000
#define IP_OFFMASK 0x1FFF
    uint8_t ttl;
    uint8_t proto;
    uint16_t checksum;
    ip_addr_t src;
    ip_addr_t dst;
    // ignore option
} ip_head_t;
```

```
#define IP_ICMP 1
#define IP_IGMP 2
#define IP_TCP 6
#define IP_UDP 17
#define IP_IGRP 88
#define IP_OSPF 89

#define IP_MAX_LEN 65536

int ip_parse(const ip_head_t* head);

#endif
```

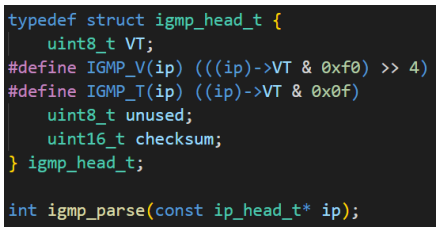
```
ip_parse(const ip_head_t* ip) {
    // only IP V4 support
    if (IP_V(ip) != 4) {
        return GAZE_IP_NOT_V4;
    }
    // ignore IP option
    if (IP_HL(ip) != sizeof(ip_head_t) / 4) {
        return GAZE_IP_WITH_OPTION;
    }
    // ignore MF (more fragment)
    if (ip->offset & IP_MF) {
        return GAZE_IP_MF_NOT_SUPPORT;
    }
    // source & dst ip address
```

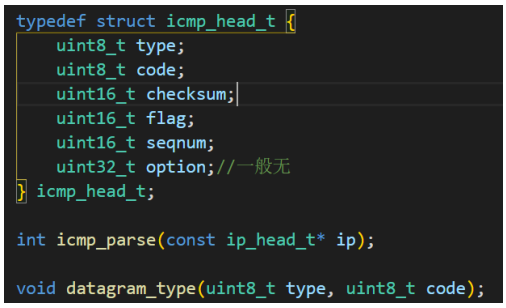




e) mp.c

这个文件主要是对 IGMP 和 ICMP 协议进行分析。对 IGMP 协议，本程序主要分析了 IGMP 头部 type 字段，以及源和目的 IP。对于 ICMP 协议，本程序主要分析了 icmp 报文类型，并打印出来。在 mp.h 头文件定义了相关的结构体，以及声明了相关的函数。





```c
void datagram_type(uint8_t type, uint8_t code)
{
    uint8_t type_ = (type & 0x0f)<<4;
    uint8_t code_ = (code & 0x0f);
    uint8_t type_code = (type_ | code_);
    switch (type_code)
    {
    case 0x00:
        printf("echo reply(ping)\n");
        break;
    case 0x30:
        printf("dest. network unreachable\n");
        break;
    case 0x31:
```

```c
int icmp_parse(const ip_head_t* ip)
{
    int send=1;
    int receve=0;
    //ip_head_t* ip_=ip;
    icmp_head_t* icmp=(icmp_head_t*)(ip + 1);
    printf("**************protacol:ICMP*****************\n");
    datagram_type(icmp->type,icmp->code);
    printf("seqence number: %d  \n",ntohs(icmp->seqnum));
    print_ip(send,ip->src.data);
    print_ip(receve,ip->dst.data);
    printf("\n");
    return GAZE_OK;
```

f) udp.c

对 UDP 协议进行处理。在 udp.h 定义 udp 协议的头部结构体，然后声明了 udp_parse 函数和 protocol_based_udp 函数对基于 udp 协议的协议进行进一步处理，这个判断的依据主要是根据端口。在本程序中我们只是对 DNS 和 DHCP 协议进行了分析。如果是其他的协议，本程序只是打印其数据，没有做具体分析。

```c
typedef struct udp_head_t {
    uint16_t sport;
    uint16_t dport;
    uint16_t datalen;
    uint16_t checksum;
} udp_head_t;

int udp_parse(const udp_head_t* head, uint32_t sip, uint32_t dip, uint16_t datalen);

#endif

void protocol_based_udp(uint16_t sport, uint16_t dport);
```

```c
void protocol_based_udp(uint16_t sport, uint16_t dport)
{
    if (sport == 67 || dport == 67)
    {
        printf("**************protacol:DHCP*****************\n");
    }
    else if (sport == 53 || dport == 53)
    {
        printf("**************protacol:DNS*****************\n");
    }
    else
    {
        printf("**************protacol:UDP*****************\n");
    }
}
```

```c
int udp_parse(const udp_head_t *udp, uint32_t sip, uint32_t dip, uint16_t datalen)
{

    int send = 1;
    int receive = 0;
    const char *buffer = (const char *)(udp);
    uint16_t  uint16_t dport p->sport);
    uint16_t dport = ntohs(udp->dport);
    protocol_based_udp(sport, dport);
    printf("datalen:%d\n",ntohs(udp->datalen));
    printf("sender port:%d    receiver port:%d\n", sport, dport );
    print_ip(send,(const unsigned char*) &sip);
    print_ip(receive,(const unsigned char*) &dip);
    _output_bytes(buffer, ntohs(udp->datalen));
    printf("\n\n");
    return GAZE_OK;
}
```

g) tcp.c

这个 TCP 协议是本程序实现的重点和难点，涉及到 tcp 流数据的重组。

应用层向 TCP 层发送用于网间传输的数据流，TCP 则把数据流分割成适当长度的报文段，最大报文段大小(MSS)通常受以太网 MTU 限制；对于 TCP 来说应尽量避免 IP 分片，TCP 协议在实现的时候往往用 MTU 值减去 IP 数据包头部和 TCP 数据段头部长度代替 MSS，一般为 1460 字节；所以通常 TCP 分段不会再出现 IP 分片的情况。MSS 是 TCP 协议定义的一个选项，MSS 选项用于在 TCP 连接建立时，收发双方协商通信时每一个报文段所能承载的最大数据长度。

因为 TCP 使用 IP 来传递它的报文段，IP 不提供重复消除和保证次序正确的功能，所以 **TCP 重组主要处理包失序和包重复**等问题。TCP 是一个字节流协议，TCP 绝不

会以杂乱的次序给接收应用程序发送数据。因此，TCP 接收端可能会被迫先保持大序列号的数据不交给应用程序，直到缺失的小序列号的报文段被填满，最终按序将数据提交给应用程序，这也是 TCP 重组的目的。

TCP 分段为什么要避免 IP 分片？

因为如果在 IP 层进行分片的话，如果其中的某片数据丢失了，对于保证可靠性的 TCP 协议来说，会增大重传数据包的机率，而且只能重传整个 TCP 分组(进行 IP 分片前的数据包)，因为 TCP 层是不知道 IP 层进行分片的细节的，也不关心。

TCP 出现包部分重复的原因？

当 TCP 超时重传，它并不需要完全重传相同的报文段。TCP 允许执行重新组包，发送一个更大的报文段来提高性能，因此可能导致部分数据重复！

在 tcp.h 头文件中对 tcp 头部进行了一个结构体的定义，然后定义了 tcp 头部相关的标志位，然后声明了 tcp_parse 函数。在 tcp_parse 函数中，首先进行了 tcp 校验，这里是 check.c 文件处理，然后判断选项字段是否存在值，然后接着就是判断 tcp 包是否是重复或者乱序到达的，这里是转到了 link.c 文件处理。

然后就是对 tcp 标志位进行获得，按需打印。

```c
typedef struct tcp_head_t {
    uint16_t sport;
    uint16_t dport;
    uint32_t seq;
    uint32_t ack;
    uint8_t offx2; // offset & reserved
    uint8_t flags;
    uint16_t window;
    uint16_t checksum;
    uint16_t ptr;
```

```c
#define TCP_OPTION_EOF 0x0
#define TCP_OPTION_NOP 0x01
#define TCP_OPTION_MSS 0x02
#define TCP_OPTION_WND_SCALE 0x03
#define TCP_OPTION_SACK 0x04
#define TCP_OPTION_TS  0x08

#define TCP_FLAG_FIN 0x01
#define TCP_FLAG_SYN 0x02
#define TCP_FLAG_RST 0x04
#define TCP_FLAG_PSH 0x08
#define TCP_FLAG_ACK 0x10
#define TCP_FLAG_URG 0x20
```

```c
} tcp_head_t;

int tcp_parse(const tcp_head_t* head, uint32_t sip, uint32_t dip, uint16_t len);

#endif
```

```c
int tcp_parse(const tcp_head_t *tcp, uint32_t sip, uint32_t dip, uint16_t tcpbytes)
{
    int ret;

    if (is_local_address(sip))
    {
        ret = _tcp_checksum(tcp, sip, dip, tcpbytes);
        if (ret < 0)
        {
            return ret;
        }
    }
```

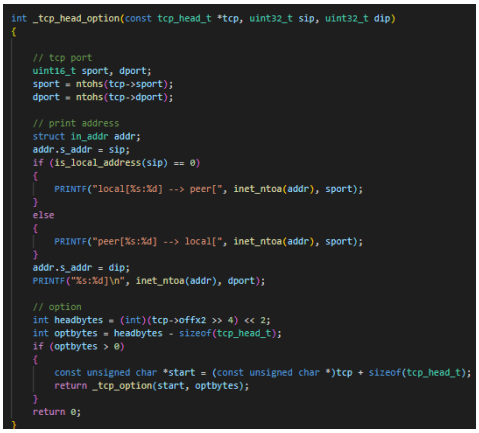
```c
// tcp head option
ret = _tcp_head_option(tcp, sip, dip);
if (ret < 0)
{
    return ret;
}
```

```
// tcpp link
link_key_t key;
link_key_init(&key, sip, dip, ntohs(tcp->sport), ntohs(tcp->dport));
struct link_value_t *val = link_find_insert(&key, is_local_address(sip));
if (!val)
    return GAZE_TCP_LINK_FAIL;

// tcp flag
```

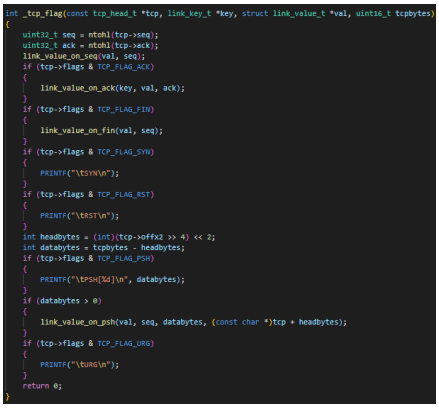
```
// tcp finish
ret = link_value_is_finish(val);
if (ret == 0)
{
    link_erase(&key);
}

PRINTF("\n\n");
return GAZE_OK;
```

```
int _tcp_checksum(const tcp_head_t *tcp, uint32_t sip, uint32_t dip, uint16_t tcpbytes)
{
    // pseudo header
    struct phead
    {
        uint32_t sip;
        uint32_t dip;
        uint8_t mbz;
        uint8_t proto;
        uint16_t len;
    } ph;
    ph.mbz = 0;
    ph.proto = IP_TCP;
    ph.len = htons(tcpbytes);
    ph.sip = sip;
    ph.dip = dip;
    struct cksum_vec vec[2];
    vec[0].ptr = (const uint8_t *)(&ph);
    vec[0].len = sizeof(ph);
    vec[1].ptr = (const uint8_t *)(tcp);
    vec[1].len = tcpbytes;
    uint16_t sum = checksum(vec, 2);
    if (sum != 0)
    {
        uint16_t tcpsum = ntohs(tcp->checksum);
        PRINTF("tcp checksum: %x, tcpbytes=%d, head->cksum=%d\n", sum, tcpbytes, tcpsum);
        return GAZE_TCP_CHECKSUM_ERROR;
    }
    return 0;
}
```

```
int _tcp_option(const unsigned char *start, int bytes)
{
    PRINTF("\t");
    for (int i = 0; i < bytes;)
    {
        if (start[i] == TCP_OPTION_NOP)
        {
            PRINTF("<nop> ");
            ++i;
            continue;
        }

        if (i + 1 >= bytes)
            return GAZE_TCP_OPTION_FAIL;
        int len = (int)start[i + 1];
        if (i + len > bytes)
            return GAZE_TCP_OPTION_FAIL;

        if (start[i] == TCP_OPTION_MSS)
        {
            assert(len == 4);
            PRINTF("<mss %u> ", ntohs(*(uint16_t *)&start[i + 2]));
        }
        else if (start[i] == TCP_OPTION_TS)
        {
            assert(len == 10);
            PRINTF("<ts %u %u> ", ntohl(*(uint32_t *)&start[i + 2]), ntohl(*(uint32_t *)&start[i + 6]));
        }
        else if (start[i] == TCP_OPTION_WND_SCALE)
        {
            assert(len == 3);
            PRINTF("<window scale %d> ", (int)*(uint8_t *)&start[i + 2]);
        }
        else if (start[i] == TCP_OPTION_SACK)
        {
            assert(len == 2);
            PRINTF("<SACK> ");
        }
        else if (start[i] == TCP_OPTION_EOF)
        {
            break;
        }
        else
        {
            PRINTF("<option[%d]> ", start[i]);
        }
        i += len;
    }
    PRINTF("\n");
    return 0;
}
```



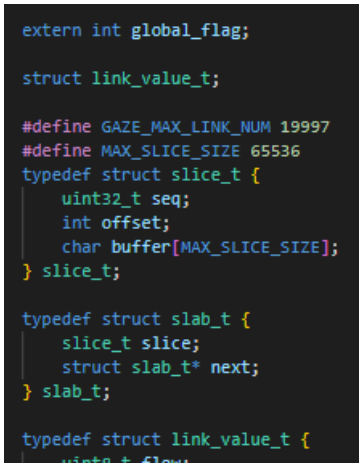
```
int _tcp_head_option(const tcp_head_t *tcp, uint32_t sip, uint32_t dip)
{

    // tcp port
    uint16_t sport, dport;
    sport = ntohs(tcp->sport);
    dport = ntohs(tcp->dport);

    // print address
    struct in_addr addr;
    addr.s_addr = sip;
    if (is_local_address(sip) == 0)
    {
        PRINTF("local[%s:%d] --> peer[", inet_ntoa(addr), sport);
    }
    else
    {
        PRINTF("peer[%s:%d] --> local[", inet_ntoa(addr), sport);
    }
    addr.s_addr = dip;
    PRINTF("%s:%d]\n", inet_ntoa(addr), dport);

    // option
    int headbytes = (int)(tcp->offx2 >> 4) << 2;
    int optbytes = headbytes - sizeof(tcp_head_t);
    if (optbytes > 0)
    {
        const unsigned char *start = (const unsigned char *)tcp + sizeof(tcp_head_t);
        return _tcp_option(start, optbytes);
    }
    return 0;
}
```



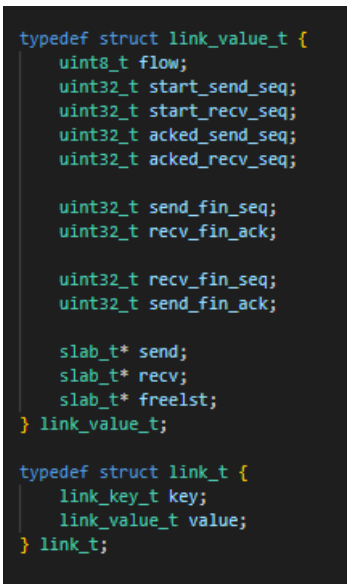
```
int _tcp_flag(const tcp_head_t *tcp, link_key_t *key, struct link_value_t *val, uint16_t tcpbytes)
{
    uint32_t seq = ntohl(tcp->seq);
    uint32_t ack = ntohl(tcp->ack);
    link_value_on_seq(val, seq);
    if (tcp->flags & TCP_FLAG_ACK)
    {
        link_value_on_ack(key, val, ack);
    }
    if (tcp->flags & TCP_FLAG_FIN)
    {
        link_value_on_fin(val, seq);
    }
    if (tcp->flags & TCP_FLAG_SYN)
    {
        PRINTF("\tSYN\n");
    }
    if (tcp->flags & TCP_FLAG_RST)
    {
        PRINTF("\tRST\n");
    }
    int headbytes = (int)(tcp->offx2 >> 4) << 2;
    int databytes = tcpbytes - headbytes;
    if (tcp->flags & TCP_FLAG_PSH)
    {
        PRINTF("\tPSH[%d]\n", databytes);
    }
    if (databytes > 0)
    {
        link_value_on_psh(val, seq, databytes, (const char *)tcp + headbytes);
    }
    if (tcp->flags & TCP_FLAG_URG)
    {
        PRINTF("\tURG\n");
    }
    return 0;
}
```

h) link.c

这一部分主要是对 tcp 流重组实现。由于 pcap_next_ex()函数抓取的数据包是原始的数据包，所以这里必须对它进行分析。上面也提到，由于网络的原因，tcp 超时等原因，tcp 会进行重传等操作，以及会发生重复的数据的现象。这里用了 hash 来判断它是否重复，然后使用链表的方式对 tcp 头部的 seq 和 ack 字段进行分析，先暂存不按顺序到达的 tcp 数据包，然后和下一个 tcp 数据包进行分析，最后当这个超时处理完后，就释放这次链表操作。在这里处理完一个完整的数据包后，同时又对 tcp 的 payload 部分进行上层协议分析。



```
extern int global_flag;

struct link_value_t;

#define GAZE_MAX_LINK_NUM 19997
#define MAX_SLICE_SIZE 65536
typedef struct slice_t {
    uint32_t seq;
    int offset;
    char buffer[MAX_SLICE_SIZE];
} slice_t;

typedef struct slab_t {
    slice_t slice;
    struct slab_t* next;
} slab_t;

typedef struct link_value_t {
```



```
typedef struct link_value_t {
    uint8_t flow;
    uint32_t start_send_seq;
    uint32_t start_recv_seq;
    uint32_t acked_send_seq;
    uint32_t acked_recv_seq;

    uint32_t send_fin_seq;
    uint32_t recv_fin_ack;

    uint32_t recv_fin_seq;
    uint32_t send_fin_ack;

    slab_t* send;
    slab_t* recv;
    slab_t* freelst;
} link_value_t;

typedef struct link_t {
    link_key_t key;
    link_value_t value;
} link_t;
```

```
int link_create();
struct link_value_t* link_find(link_key_t*);
struct link_value_t* link_insert(link_key_t*);
struct link_value_t* link_find_insert(link_key_t*, int is_send);
void link_erase(link_key_t*);
void link_release();

void link_key_init(link_key_t* key, uint32_t sip, uint32_t dip, uint16_t sport, uint16_t dport);

void link_value_on_seq(struct link_value_t*, uint32_t seq);
void link_value_on_ack(link_key_t*, struct link_value_t*, uint32_t ack);
void link_value_on_psh(struct link_value_t*, uint32_t seq, int bytes, const char*);
void link_value_on_fin(struct link_value_t*, uint32_t seq);

int link_value_is_finish(struct link_value_t*);

#endif
```

```
*/
static struct hash_t *g_links = NULL;

uint32_t
_link_hash(const void *data)
{
    const link_t *link = (const link_t *)data;
    return hash_jhash((const void *)&link->key, sizeof(link_key_t));
}

int32_t
_link_cmp(const void *data1, const void *data2)
{
    const link_t *link1 = (const link_t *)data1;
    const link_t *link2 = (const link_t *)data2;
    return memcmp(&link1->key, &link2->key, sizeof(link_key_t));
}
```

```
void _link_value_slab_gc(link_value_t *val, slab_t *slab)
{
    if (val && slab)
    {
        if (val->freelst)
        {
            slab->next = val->freelst;
            val->freelst = slab;
        }
        else
        {
            slab->next = NULL;
            val->freelst = slab;
        }
    }
}
```

```
slab_t *
_link_value_slab_alloc(link_value_t *val)
{
    if (val && val->freelst)
    {
        slab_t *get = val->freelst;
        val->freelst = val->freelst->next;
        memset(get, 0, sizeof(slab_t));
        return get;
    }
    return (slab_t *)calloc(sizeof(slab_t), 1);
}
```

```
int link_create()
{
    g_links = hash_create(_link_hash, _link_cmp, GAZE_MAX_LINK_NUM);
    return g_links ? 0 : -1;
}

link_value_t *
link_find(link_key_t *key)
{
    link_t link;
    link.key = *key;
    void *dst = hash_find(g_links, &link);
    if (dst)
    {
        return &((link_t *)dst)->value;
    }
    return NULL;
}
```

```
link_value_t *
link_insert(link_key_t *key)
{
    link_t *link = (link_t *)calloc(sizeof(link_t), 1);
    link->key = *key;
    if (hash_insert(g_links, link))
    {
        free(link);
        return NULL;
    }
    return &link->value;
}

#define PKG_SEND 0
#define PKG_RECV 1
```

```
#define PKG_SEND 0
#define PKG_RECV 1

void link_key_init(link_key_t *key, uint32_t sip, uint32_t dip, uint16_t sport, uint16_t dport)
{
    if (is_local_address(sip) == 0)
    {
        key->local_ip = sip;
        key->peer_ip = dip;
        key->local_port = sport;
        key->peer_port = dport;
    }
    else
    {
        key->peer_ip = sip;
        key->local_ip = dip;
        key->peer_port = sport;
        key->local_port = dport;
    }
}
```

```c
link_value_t *
link_find_insert(link_key_t *key, int is_send)
{
    int flow = (is_send == 0 ? PKG_SEND : PKG_RECV);
    link_value_t *val = link_find(key);
    if (!val)
    {
        val = link_insert(key);
        if (!val)
        {
            return NULL;
        }

        g_build_hook(key);

        memset(val, 0, sizeof(link_value_t));
    }
    val->flow = flow;
    return val;
}
```

```c
void _link_release(void *data, void *args)
{
    link_t *dst = (link_t *)data;
    if (dst)
    {
        slab_t *tmp;
        while (dst->value.send)
        {
            tmp = dst->value.send;
            dst->value.send = dst->value.send->next;
            _link_value_slab_gc(&dst->value, tmp);
        }
        while (dst->value.recv)
        {
            tmp = dst->value.recv;
            dst->value.recv = dst->value.recv->next;
            _link_value_slab_gc(&dst->value, tmp);
        }
        while (dst->value.freelst)
        {
            tmp = dst->value.freelst;
            dst->value.freelst = dst->value.freelst->next;
            free(tmp);
        }
        free(dst);
    }
}
```

```c
void link_erase(link_key_t *key)
{
    link_t link;
    link.key = *key;
    link_t *dst = (link_t *)hash_find(g_links, &link);
    if (dst)
    {

        g_finish_hook(key);

        hash_remove(g_links, &link);
        _link_release(dst, NULL);
    }
}
```

```c
void link_value_on_seq(link_value_t *val, uint32_t seq)
{
    if (val && val->flow == PKG_SEND && val->start_send_seq == 0)
    {
        val->start_send_seq = seq;
        PRINTF("\tS = %u\n", seq);
    }
    if (val && val->flow == PKG_RECV && val->start_recv_seq == 0)
    {
        val->start_recv_seq = seq;
        PRINTF("\tR = %u\n", seq);
    }

    if (val->flow == PKG_SEND)
    {
        PRINTF("\tSEQ[S + %u] \n", seq - val->start_send_seq);
    }
    else
    {
        PRINTF("\tSEQ[R + %u] \n", seq - val->start_recv_seq);
    }
}
```

```c
void _link_value_on_ack_notify(link_key_t *key, link_value_t *val, uint32_t ack)
{

    int protocol_based_tcp_flag = 0;

    slab_t *slab = (val->flow == PKG_SEND ? val->recv : val->send);
    while (slab)
    {
        if (val->flow == PKG_SEND)
        {
            PRINTF("\tacked recv slice[%d]\n", slab->slice.offset);
        }
        else
        {
            PRINTF("\tacked send slice[%d]\n", slab->slice.offset);
        }
        if (ack == slab->slice.seq + slab->slice.offset)
        {
            break;
        }
        slab = slab->next;
    }
```

```c
if (!slab)
    return;

slab_t *from;
if (val->flow == PKG_SEND)
{
    from = val->recv;
    val->recv = slab->next;
}
else
{
    from = val->send;
    val->send = slab->next;
}
slab->next = 0;

while (from)
{
    if (val->flow == PKG_SEND)
    {
```

```c
while (from)
{
    if (val->flow == PKG_SEND)
    {

        if (global_flag == 0 || global_flag == 1 || global_flag == 2 || global_flag == 3)

        {
            protocol_based_tcp_flag = protocol_based_tcp(key, (const uint8_t *)from->slice.buffer);
            protocol_process(key, protocol_based_tcp_flag, (const uint8_t *)from->slice.buffer, from->slice.offset);
            //printf("%d  send\n",from->slice.offset);
        }else
        {
            g_recv_hook(key, from->slice.buffer, from->slice.offset);
            //printf("%d send\n",from->slice.offset);
        }

    }
    else
    {
```

```
else
{

    if (global_flag == 0 || global_flag == 1 || global_flag == 2 || global_flag == 3)

    {
        protocol_based_tcp_flag = protocol_based_tcp(key, (const uint8_t *)from->slice.buffer);
        protocol_process(key, protocol_based_tcp_flag, (const uint8_t *)from->slice.buffer, from->slice.offset);
        //printf("%d  receive\n",from->slice.offset);
    }else
    {
        g_recv_hook(key, from->slice.buffer, from->slice.offset);
        //printf("%d  receive\n",from->slice.offset);
    }

}
```

```
void link_value_on_ack(link_key_t *key, link_value_t *val, uint32_t ack)
{
    if (val->flow == PKG_SEND)
    {
        val->acked_recv_seq = ack;
        if (val->start_recv_seq == 0)
        {
            PRINTF("\tACK[%u]\n", ack);
        }
        else
        {
            PRINTF("\tACK[R + %u]\n", ack - val->start_recv_seq);
        }

        if (val->recv_fin_seq > 0 && ack >= val->recv_fin_seq)
        {
            val->send_fin_ack = ack;
        }
    }
    else
    {

        val->acked_send_seq = ack;
        if (val->start_send_seq == 0)
        {
            PRINTF("\tACK[%u]\n", ack);
        }
        else
        {
            PRINTF("\tACK[S + %u]\n", ack - val->start_send_seq);
        }

        if (val->send_fin_seq > 0 && ack >= val->send_fin_seq)
        {
            val->recv_fin_ack = ack;
        }
    }
    _link_value_on_ack_notify(key, val, ack);
}
```

```
void link_value_on_fin(link_value_t *val, uint32_t seq)
{
    PRINTF("\tFIN\n");
    if (val->flow == PKG_SEND)
    {
        val->send_fin_seq = seq;
    }
    else
    {
        val->recv_fin_seq = seq;
    }
}
```

```
void link_value_on_psh(link_value_t *val, uint32_t seq, int bytes, const char *data)
{
    if (!val || !data || bytes <= 0)
        return;
    slab_t *slab;
    slab = (val->flow == PKG_SEND ? val->send : val->recv);
    if (!slab)
    {
        slab = _link_value_slab_alloc(val);
        if (val->flow == PKG_SEND)
        {
            val->send = slab;
        }
        else
        {
            val->recv = slab;
        }
    }
    else
    {
```

```
        // insert by ascending
        while (slab->next && slab->next->slice.seq < seq)
        {
            slab = slab->next;
        }
        slab_t *newslab = _link_value_slab_alloc(val);
        if (slab->next)
        {
            newslab->next = slab->next;
        }
        slab->next = newslab;
        slab = slab->next;
    }
    slab->slice.seq = seq;
    slab->slice.offset = bytes;
    memcpy(slab->slice.buffer, data, bytes);
}
```

```
int link_value_is_finish(struct link_value_t *val)
{
    if (val && val->send_fin_seq > 0 && val->recv_fin_seq > 0 && val->recv_fin_ack >= val->send_fin_seq && val->send_fin_ack >= val->recv_fin_seq)
    {
        return 0;
    }
    return -1;
}
```

i) hash.c 和 check.c

这里就不对他们进行展示，因为这是开源的代码。

j) protocol_based_tcp.c

这也是本程序主要设计的功能之一。这个文件中实现了对 HTPP、FTP、TELNET 三个协议的分析并能抓取通过这三个协议传输的用户名和密码。在这个文件中，首先对 tcp 的端口进行判定，如果是 21 21 端口就说明这是 ftp 传输协议、如果是 23 端口就是 telnet 协议，http 协议是通过 payload 的字段数据来判断的。

假如判断这是 http 的数据包之后，，就会判断它是何种类型的，例如 GET、POST 或者其他的，然后假如是 POST，就会抓取 HTTP 协议的 payload 部分，将它全部写到 http.txt 中去。因为对于使用 http 协议登录的网站，有可能是明文传输，也有可能是 MD5 加密后的。

假如是 FTP 协议，通过对 FTP 协议的分析，发现它传输用户名和密码时会进行关键字提示，所以就根据关键字来定位用户名和密码，然后写到 ftp.txt 文件中去。

假如是 TELNET 协议，这个协议比较复杂，在网络良好的情况下，它会一个字符一个字符的传输用户名密码，假如 Ack 确认包丢失或者超时后，它会重发该字符并再加一个字符，所以这种情况就是两个字符。同时，telnet 服务器在收到用户名的字符后还会回传这个字符表示收到。但是对于密码的字符它就不会回传，只会最终确认。在对这个协议进行用户名和密码抓取时本程序使用了链表，因为考虑到多个用户同时登陆时两者的数据包会交叉接收，所以同股票 PORT 和 IP 来进行判断，同时利用\r\n 和其他关键字进行判断。

```
extern int global_flag;
extern telnet_capture_t *head;

int protocol_based_tcp(link_key_t *tcp, const uint8_t *start);
void protocol_process(link_key_t *tcp, int flag, const uint8_t *start, int lengt);



int http_justify(const uint8_t *start);
int http_methods(const uint8_t *start);
int three_str_cmp(const char *src, const char *str1, const char *str2, int len);

int http_print(const uint8_t *start, int length);
void http_txt_write(const uint8_t *data);

void ftp_txt_write(const uint8_t *data, uint16_t sport, uint16_t dport, int totlen);
void ftp_print(const uint8_t *start, int length);

void telnet_packet_handle(const uint8_t *data, uint16_t sport, uint16_t dport, int totlen);
void telnet_txt_write(telnet_capture_t *me);
void telnet_print(const uint8_t *start, int length);
```

```c
int protocol_based_tcp(link_key_t *tcp, const uint8_t *start)
{
    if (http_justify(start) == 1)
    {
        return 1; //http
    }

    else if ((tcp->local_port) == 21 || (tcp->peer_port == 21))
    {

        //printf("ftp*******\n");
        return 3; //ftp
    }
    else if ((tcp->local_port == 22223) || (tcp->peer_port == 22223))
    {
        return 2; //telnet
    }
    else
    {
        return 0;
    }
}
```

```c
void protocol_process(link_key_t *tcp, int flag, const uint8_t *start, int length)
{
    // 1 http 2 telnet 3 ftp 0 三个都要
    //int global_flag = 0;

    int offset = 0;
    // tcp port
    uint16_t sport, dport;
    sport = /**/ (tcp->local_port);
    dport = /**/ (tcp->peer_port);

    //printf("%d global_flag\n",global_flag);

    switch (global_flag)
    {
    case 1:
        /* code http*/
        if (flag == 1)
        {
            /* do*/
            /* code http*/
            offset = http_print(start, length);
```

```c
        //printf("%c\n", start[offset]);
        if (offset == -1)
        {
            printf("HTTP resolve error!\n");
            break;
        }
        else
        {
            if (http_methods(start) == 2)
            {
                http_txt_write((start + offset));
            }
        }
    }

    break;
case 2:
    /* code telnet*/
    if (flag == 2)
    {
        /* do */
        telnet_print(start, length);
        telnet_packet_handle(start, sport, dport, length);
    }
```

```c
        break;
    case 3:
        /* code ftp*/
        if (flag == 3)
        {
            /* do*/;
            /* code ftp*/
            ftp_print(start, length);
            ftp_txt_write(start, sport, dport, length);
        }

        break;
    case 0:
        /*http*/
        /*telnet*/
        /*ftp*/
        switch (flag)
        {
        case 1:
            /* code http*/
            offset = http_print(start, length);
            printf("offset %d length %d\n", offset, length);
            //printf("%c\n", start[offset]);
            if (offset == -1)
            {
```

```c
    case 1:
        /* code http*/
        offset = http_print(start, length);
        printf("offset %d length %d\n", offset, length);
        //printf("%c\n", start[offset]);
        if (offset == -1)
        {
            printf("HTTP resolve error!\n");
            break;
        }
        else
        {
            if (http_methods(start) == 2)
            {
                http_txt_write((start + offset));
            }
        }

        break;
    case 2:
        /* code telnet*/
        telnet_print(start, length);
        telnet_packet_handle(start, sport, dport, length);
```

```c
            /* code telnet*/
            telnet_print(start, length);
            telnet_packet_handle(start, sport, dport, length);
            break;
        case 3:
            /* code ftp*/
            ftp_print(start, length);
            ftp_txt_write(start, sport, dport, length);
            break;

        default:
            printf("unknown protocol based on tcp\n");
            break;
        }
        break;
    }
}
```

```c
void ftp_txt_write(const uint8_t *data, uint16_t sport, uint16_t dport, int totlen)
{
    FILE *fp;
    uint16_t port;
    int len = totlen - 2;
    const char text[len];
    //printf("FTP ***********\n");
    memcpy((void *)text, (void *)data, len);
    if (memcmp((void *)text, "PASS", 4) == 0 || memcmp((void *)text, "USER", 4) == 0)
    {
        fp = fopen("ftp.txt", "a+");
        port = ((sport) != 21) ? (sport) : (dport);
        /*
        a+ 以附加方式打开可读写的文件，若文件不存在，则会建立该文件，
        如果文件存在，写入的数据会被加到文件尾后，即文件原先的内容会被保留。
        原来的EOF符不保留）
        */
        //printf("+++++++++++++++%s %d++++++++++++++++\n", text, len);
        fprintf(fp, "%s %d", text, port);
        fputc(32, fp);
```

```c
        如果文件存在，写入的数据会被加到文件尾后，即文件原先的内容会被保留。
        原来的EOF符不保留）
        */
        //printf("+++++++++++++++%s %d++++++++++++++++\n", text, len);
        fprintf(fp, "%s %d", text, port);
        fputc(32, fp);
        if (memcmp((void *)text, "PASS", 4) == 0)
        {
            fputc(10, fp);
        }

        fclose(fp);
    }
}
```

```c
void ftp_print(const uint8_t *start, int length)
{
    int offset = 0;
    for (offset = 0; offset < length; offset++)
    {
        printf("%c", start[offset]);
    }
    printf("\n");
}
```

```c
void telnet_print(const uint8_t *start, int length)
{
    int offset = 0;
    for (offset = 0; offset < length; offset++)
    {
        printf("%c", start[offset]);
    }
    printf("\n");
}
```

```
}
void telnet_txt_write(telnet_capture_t *me)
{
    FILE *fp;
    int length_username;
    int i= 0;
    char name[128];
    length_username = strlen(me->username);
    for ( i = 0; i < length_username; i += 2)
    {
        name[i/2]=me->username[i];
    }

    fp = fopen("telnet.txt", "a+");

    printf("U:%s  P:%s\n", name, me->password);

    fprintf(fp, "%s %s ", name, me->password);
    fputc(10, fp);
    fclose(fp);
}
```

```
void telnet_packet_handle(const uint8_t *start, uint16_t sport, uint16_t dport, int length)
{
    telnet_capture_t *telnet = NULL;
    telnet_capture_t *temp = NULL;
    telnet_capture_t *delete_temp = NULL;
    int find_flag = 0;
    int initial = 0;
    for (temp = head; temp->next != NULL; temp = temp->next)
    {
        if (temp->next->peer_port == dport)
        {
            if (length == 1 || (length ==2 && start[0] != '\x0d'))
            //if (temp->next->write_flag == 1  && start[0] != '\x0d' )
            {
                if (temp->next->write_flag == 1)
                {

                    if (temp->next->username_finish_flag == 0)
                    {
                        strncat(temp->next->username, (char *)start, length);
                        printf("username %s++++++++\n", temp->next->username);
                        //printf("%c ************\n", start[0]);
                    }
                    else if (temp->next->password_finish_flag == 0)
                    {
                        strncat(temp->next->password, (char *)start, length);
                        printf("%c password************\n", start[0]);
                    }
                    else
                    {
```

```
                    find_flag = 1;
                    break;
                }
                else if (length == 2)
                {
                    if (temp->next->username_finish_flag == 0)
                    {
                        if (start[0] == '\x0d')
                        {
                            temp->next->username_finish_flag = 1;
                        }
                    }
                    else if (temp->next->password_finish_flag == 0)
                    {
                        if (start[0] == '\x0d')
                        {
                            temp->next->username_finish_flag = 1;
                            printf("begin to write\n");
                            telnet_txt_write(temp->next);
                            delete_temp = temp->next->next;
                            temp->next = delete_temp;
                        }
                    }
                    else
                    {
                        temp->next->write_flag = 0;
                    }
```

```
                    }
                    else
                    {
                        temp->next->write_flag = 0;
                    }

                    find_flag = 1;
                    break;
                }
                else if (length > 7)
                {
                    //printf("start %s\n-----------", start);
                    if (strstr((char *)(start + length - 8), "login") != NULL)
                    {
                        temp->next->write_flag = 1;
                        temp->next->username_finish_flag = 0;
                        printf("检测到login\n");

                        find_flag = 1;
                        break;
                    }

                    else if (memcmp((char *)start, "Password:\x20", 10) == 0)
                    {
                        temp->next->write_flag = 1;
                        temp->next->password_finish_flag = 0;
                        printf(" password checked\n");
```

```
    }
    if (find_flag == 0)
    {
        telnet = (telnet_capture_t *)malloc(sizeof(telnet_capture_t));
        printf(" 检测到telnet登录\n" );
        telnet->peer_port = dport;
        telnet->local_port = sport;
        telnet->write_flag = 0;
        telnet->username_finish_flag = 1;
        telnet->password_finish_flag = 1;
        for (; initial < 128; initial++)
        {
            telnet->username[initial] = '\x00';
            telnet->password[initial] = '\x00';
        }
        telnet->next = NULL;

        for (temp = head; temp != NULL; temp = temp->next)
        {
            if (temp->next == NULL)
            {
                temp->next = telnet;
                break;
            }
```

```
                {
                    telnet->username[initial] = '\x00';
                    telnet->password[initial] = '\x00';
                }
                telnet->next = NULL;

                for (temp = head; temp != NULL; temp = temp->next)
                {
                    if (temp->next == NULL)
                    {
                        temp->next = telnet;
                        break;
                    }
                }
            }
            void http_txt_write(const uint8_t *data)
```

```
void http_txt_write(const uint8_t *data)
{
    FILE *fp;
    fp = fopen("http.txt", "a+");
    /*
    a+ 以附加方式打开可读写的文件。若文件不存在，则会建立该文件，
    如果文件存在，写入的数据会被加到文件尾后，即文件原先的内容会被保留。
        （原来的EOF符不保留）
    */
    //printf("%s\n", data);
    fprintf(fp, "%s ", data);
    fputc(10, fp);
    //fputs(data,fp);
    //printf("555");
    fclose(fp);
}
```

```
int http_print(const uint8_t *start, int length)
{
    int offset = 0;
    for (offset = 0; offset < length; offset++)
    {
        if (memcmp((void *)(start + offset + 1), "\x0d\x0a", 2) == 0)
        {
            if (memcmp((void *)(start + offset + 1), "\x0d\x0a\x0d\x0a", 4) == 0)
            {
                if (isprint(start[offset]))
                {
                    printf("%c\n", start[offset]);
                    return offset + 5;
                }
                else
                {
                    printf(".\n");
                    return offset + 5;
                }
            }
            else
```

```
                    //printf(".\n");
                }
            }
        }
        else
        {
            if (isprint(start[offset]))
            {
                printf("%c", start[offset]);
            }
            else
            {
                //printf(".");
            }
        }
    }
    return -1;
}
int three_str_cmp(const char *src, const char *str1, const char *str2, int len)
{
    if ((memcmp((void *)src, (void *)str1, len) * memcmp((void *)src, (void *)str2, len)) == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

```
int http_methods(const uint8_t *start)
{
    char get[3];
    char post[4];
    char put[3];
    char delete[6];
    char head[4];
    char trace[5];
    char options[7];
    memcpy((void *)get, (void *)start, 3);
    memcpy((void *)post, (void *)start, 4);
    memcpy((void *)put, (void *)start, 3);
    memcpy((void *)delete, (void *)start, 6);
    memcpy((void *)head, (void *)start, 4);
    memcpy((void *)trace, (void *)start, 5);
    memcpy((void *)options, (void *)start, 7);
    if (three_str_cmp(get, "GET", "get", 3))
    {
        return 1;
    }
    if (three_str_cmp(post, "POST", "post", 4))
    {
        return 2;
    }
    if (three_str_cmp(put, "PUT", "put", 3))
    {
        return 3;
    }
    if (three_str_cmp(delete, "DELETE", "delete", 6))
    {
        return 4;
    }
    if (three_str_cmp(head, "HEAD", "head", 4))
    {
        return 5;
    }
    if (three_str_cmp(trace, "TRACE", "trace", 5))
    {
        return 6;
    }
    if (three_str_cmp(options, "OPTIONS", "options", 7))
    {
        return 7;
    }
    return 0;
```

```
int http_justify(const uint8_t *start)
{

    if (strstr((char *)start, "\x48\x54\x54\x50\x2f\x31\x2e") != NULL || strstr((char *)start, "\x48\x54\x54\x50\x2f\x31\x2e") != NULL)
    {
        //printf("http**********\n");
        return 1;
    }
    return 0;
}
```

k) gaze.h 这是本程序自定义的返回类型

```
#if defined(__LINUX__) || defined(__linux__)
    #define COLOR_RED   { printf("\033[31;1m"); }
    #define COLOR_GREEN { printf("\033[32;1m"); }
    #define COLOR_RESET { printf("\033[;0m"); }

#elif defined(WIN32)
    #include <windows.h>
    #define COLOR_RED \
        { \
            HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE); \
            SetConsoleTextAttribute(h, FOREGROUND_RED | FOREGROUND_INTENSITY); \
        }
    #define COLOR_GREEN \
        { \
            HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE); \
            SetConsoleTextAttribute(h, FOREGROUND_GREEN | FOREGROUND_INTENSITY); \
        }
    #define COLOR_RESET \
        { \
            HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE); \
            SetConsoleTextAttribute(h, FOREGROUND_RED | FOREGROUND_BLUE \
                | FOREGROUND_GREEN | FOREGROUND_INTENSITY); \
        }
```

```
#else
    #define COLOR_RED
    #define COLOR_GREEN
    #define COLOR_RESET
#endif

// ignore same port+ip pair
typedef struct link_key_t {
    int local_ip;
    int peer_ip;
    uint16_t local_port;
    uint16_t peer_port;
} link_key_t;
```

```c
// ignore same port+ip pair
typedef struct link_key_t {
    int local_ip;
    int peer_ip;
    uint16_t local_port;
    uint16_t peer_port;
} link_key_t;

#define DYLIB_SEND_SYMBOL "OnSend"
#define DYLIB_RECV_SYMBOL "OnRecv"
#define DYLIB_BUILD_SYMBOL "OnBuild"
#define DYLIB_FINISH_SYMBOL "OnFinish"

typedef void (*send_hook)(link_key_t* key, const char* buffer, int len);
typedef void (*recv_hook)(link_key_t* key, const char* buffer, int len);
typedef void (*build_hook)(link_key_t* key);
typedef void (*finish_hook)(link_key_t* key);

#define GAZE_OK 0
```

```c
#define GAZE_IP_FAIL -200
#define GAZE_IP_NOT_SUPPORT -201
#define GAZE_IP_NOT_V4 -202
#define GAZE_IP_WITH_OPTION -203
#define GAZE_IP_CHECKSUM_ERROR -204
#define GAZE_IP_MF_NOT_SUPPORT -205

#define GAZE_TCP_FAIL -300
#define GAZE_TCP_CHECKSUM_ERROR -301
#define GAZE_TCP_OPTION_FAIL -302
#define GAZE_TCP_LINK_FAIL -303

#define GAZE_DYLIB_FAIL -400
#define GAZE_DYLIB_SYMBLE_FAIL -401

#endif
```

1) dlfcn.h

这是 win32 平台要包含的文件，里面主要是对网卡的识别。因为在所用的函数中，它不能明确指明这是以太网网卡还是无线网卡，这是 windows 下 npcap 函数的毛病，所以用了其他的方式来获得网卡更明确的称谓。

```c
#ifndef DLFCN_H
#define DLFCN_H

/* POSIX says these are implementation-defined.
 * To simplify use with Windows API, we treat them the same way.
 */

#define RTLD_LAZY   0
#define RTLD_NOW    0

#define RTLD_GLOBAL (1 << 1)
#define RTLD_LOCAL  (1 << 2)

/* These two were added in The Open Group Base Specifications Issue 6.
 * Note: All other RTLD_* flags in any dlfcn.h are not standard compliant.
 */

#define RTLD_DEFAULT    0
#define RTLD_NEXT       0

void *dlopen ( const char *file, int mode );
int   dlclose( void *handle );
void *dlsym  ( void *handle, const char *name );
char *dlerror( void );
int GetNameOfNetServiceWin( void );

#endif /* DLFCN_H */
```

```c
typedef struct global_object {
    HMODULE hModule;
    struct global_object *previous;
    struct global_object *next;
} global_object;

static global_object first_object;

/* These functions implement a double linked list for the global objects. */
static global_object *global_search( HMODULE hModule )
{
    global_object *pobject;

    if( hModule == NULL )
        return NULL;

    for( pobject = &first_object; pobject ; pobject = pobject->next )
        if( pobject->hModule == hModule )
            return pobject;

    return NULL;
}

static void global_add( HMODULE hModule )
{
```

```c
static void global_add( HMODULE hModule )
{
    global_object *pobject;
    global_object *nobject;

    if( hModule == NULL )
        return;

    pobject = global_search( hModule );

    /* Do not add object again if it's already on the list */
    if( pobject )
        return;

    for( pobject = &first_object; pobject->next ; pobject = pobject->next );

    nobject = malloc( sizeof(global_object) );

    /* Should this be enough to fail global_add, and therefore also fail
```

```c
    nobject = malloc( sizeof(global_object) );

    /* Should this be enough to fail global_add, and therefore also fail
     * dlopen?
     */
    if( !nobject )
        return;

    pobject->next = nobject;
    nobject->next = NULL;
    nobject->previous = pobject;
    nobject->hModule = hModule;
}
```

```c
static void global_rem( HMODULE hModule )
{
    global_object *pobject;

    if( hModule == NULL )
        return;

    pobject = global_search( hModule );

    if( !pobject )
        return;

    if( pobject->next )
        pobject->next->previous = pobject->previous;
    if( pobject->previous )
        pobject->previous->next = pobject->next;

    free( pobject );
}
```

```c
static char error_buffer[65535];
static char *current_error;

static int copy_string( char *dest, int dest_size, const char *src )
{
    int i = 0;

    /* gcc should optimize this out */
    if( !src && !dest )
        return 0;

    for( i = 0 ; i < dest_size-1 ; i++ )
    {
        if( !src[i] )
            break;
        else
            dest[i] = src[i];
    }
    dest[i] = '\0';

    return i;
}
```

```c
static void save_err_str( const char *str )
{
    DWORD dwMessageId;
    DWORD pos;

    dwMessageId = GetLastError( );

    if( dwMessageId == 0 )
        return;

    /* Format error message to:
     * "<argument to function that failed>": <Windows localized error message>
     */
    pos  = copy_string( error_buffer,     sizeof(error_buffer),     "\"" );
    pos += copy_string( error_buffer+pos, sizeof(error_buffer)-pos, str );
    pos += copy_string( error_buffer+pos, sizeof(error_buffer)-pos, "\": " );
    pos += FormatMessage( FORMAT_MESSAGE_FROM_SYSTEM, NULL, dwMessageId,
                          MAKELANGID( LANG_NEUTRAL, SUBLANG_DEFAULT ),
                          error_buffer+pos, sizeof(error_buffer)-pos, NULL );

    if( pos > 1 )
    {
```

```c
    if( pos > 1 )
    {
        /* POSIX says the string must not have trailing <newline> */
        if( error_buffer[pos-2] == '\r' && error_buffer[pos-1] == '\n' )
            error_buffer[pos-2] = '\0';
    }

    current_error = error_buffer;
}

static void save_err_ptr_str( const void *ptr )
{
    char ptr_buf[19]; /* 0x<pointer> up to 64 bits. */

    sprintf( ptr_buf, "0x%p", ptr );

    save_err_str( ptr_buf );
}
```

```c
    hModule = GetModuleHandle( NULL );

    if( !hModule )
        save_err_ptr_str( file );
}
else
{
    char lpFileName[MAX_PATH];
    int i;

    /* MSDN says backslashes *must* be used instead of forward slashes. */
    for( i = 0 ; i < sizeof(lpFileName)-1 ; i++ )
    {
        if( !file[i] )
            break;
        else if( file[i] == '/' )
            lpFileName[i] = '\\';
        else
            lpFileName[i] = file[i];
    }
    lpFileName[i] = '\0';
```

```c
        */
        if( !hModule )
            save_err_str( lpFileName );
        else if( (mode & RTLD_GLOBAL) )
            global_add( hModule );
    }

    /* Return to previous state of the error-mode bit flags. */
    SetErrorMode( uMode );

    return (void *) hModule;
}

int dlclose( void *handle )
{
    HMODULE hModule = (HMODULE) handle;
    BOOL ret;

    current_error = NULL;

    ret = FreeLibrary( hModule );

    /* If the object was loaded with RTLD_GLOBAL, remove it from list of g
     * objects.
     */
    if( ret )
```

```c
        global_rem( hModule );
    else
        save_err_ptr_str( handle );

    /* dlclose's return value in inverted in relation to
    ret = !ret;

    return (int) ret;
}

void *dlsym( void *handle, const char *name )
{
    FARPROC symbol;

    current_error = NULL;

    symbol = GetProcAddress( handle, name );

    if( symbol == NULL )
    {
        HMODULE hModule;
```

```c
    hModule = GetModuleHandle( NULL );

    if( hModule == handle )
    {
        global_object *pobject;

        for( pobject = &first_object; pobject ; pobject = pobject->next )
        {
            if( pobject->hModule )
            {
                symbol = GetProcAddress( pobject->hModule, name );
                if( symbol != NULL )
                    break;
            }
        }
    }

    CloseHandle( hModule );
}
```

```c
        CloseHandle( hModule );
    }

    if( symbol == NULL )
        save_err_str( name );

    return (void*) symbol;
}

char *dlerror( void )
{
    char *error_pointer = current_error;

    /* POSIX says that invoking dlerror( ) a second time, immediately following
     * a prior invocation, shall result in NULL being returned.
     */
    current_error = NULL;

    return error_pointer;
}
```

```c
int GetNameOfNetServiceWin()
{
    // 初始化winsock
    PIP_ADAPTER_INFO pAdapterInfo;
    PIP_ADAPTER_INFO pAdapter = NULL;
    ULONG ulOutBufLen = sizeof(IP_ADAPTER_INFO);
    pAdapterInfo = (PIP_ADAPTER_INFO)malloc(ulOutBufLen);
    DWORD dwRetVal = GetAdaptersInfo(pAdapterInfo, &ulOutBufLen);
    // 第一次调用GetAdapterInfo获取ulOutBufLen大小
    if (dwRetVal == ERROR_BUFFER_OVERFLOW)
    {
        free(pAdapterInfo);
        pAdapterInfo = (IP_ADAPTER_INFO*)malloc(ulOutBufLen);
        dwRetVal = GetAdaptersInfo(pAdapterInfo, &ulOutBufLen);
    }
    if (dwRetVal == NO_ERROR)
    {
        pAdapter = pAdapterInfo;
        while (pAdapter)
        {
```

```
    {
        pAdapter = pAdapterInfo;
        while (pAdapter)
        {
            printf("Adapter Desc: \t%s\n", pAdapter->Description);
            printf("Adapter Name: \t%s\n", pAdapter->AdapterName);
            printf("\n");
            pAdapter = pAdapter->Next;
        }// end while
    }
    else
    {
        printf("Call to GetAdaptersInfo failed.\n");
    }
    if (pAdapterInfo != NULL)
    {
        free(pAdapterInfo); //释放资源
    }
    return 0;
}
```

m) MakeFIle

这是本程序的编译的 makefile 文件，里面包含了在不同环境下的库文件的使用，根据不同的环境可以产生在不同平台的可执行文件。

```
.PHONY : mingw linux macos undefined

CFLAGS := -g -Wall -std=gnu99
LDFLAGS :=
LIBS :=
TARGET := MySniff
INCLUDES := -I./include/ -I./src/
SRC := \
    src/main.c \
    src/eth.c \
    src/ip.c \
    src/mp.c \
    src/arp.c \
    src/tcp.c \
    src/udp.c \
    src/checksum.c \
    src/protocol_based_tcp.c \
    src/hash.c \
    src/link.c \
    src/output.c

UNAME=$(shell uname)
SYS=$(if $(filter Linux%, $(UNAME)), linux,\
        $(if $(filter CYGWIN%, $(UNAME)), cygwin,\
            $(if $(filter MINGW%, $(UNAME)), mingw,\
                $(if $(filter Darwin%, $(UNAME)), macos,\
                undefined))))
```

```
all: $(SYS)

undefined:
    @echo "please do 'make PLATFORM' where PLATFORM is one of these:"
    @echo "      macos linux mingw"

mingw : CFLAGS += -DHAVE_REMOTE -DMINGW -Wformat=0
mingw : INCLUDES += -I./winpcap/include -I./dlfcn -I./winpcap/dll
mingw : LDFLAGS += -lmingw32 -lws2_32 -lpthread -liphlpapi
mingw : LIBS += ./winpcap/lib/x64/Packet.lib ./winpcap/lib/x64/wpcap.lib
mingw : SRC += dlfcn/dlfcn.c
mingw : $(SRC) $(TARGET)

linux : CFLAGS += -DLINUX
linux : INCLUDES += -I./libpcap/include
linux : LDFLAGS += -ldl
linux : LIBS += ./libpcap/lib/libpcap.linux.a -lpthread
linux : $(SRC) $(TARGET)


$(TARGET) :
    gcc $(CFLAGS) -o $(TARGET) $(SRC) $(INCLUDES) $(LDFLAGS) $(LIBS)

clean :
    @(rm -f MySniff src/*.o)
```

3.1.7 处理流程

【说明】可采用框图+文字叙述或 PAD 图+文字叙述。只要表达得清晰准确即可。

3.1.8 应说明的问题与限制

【说明】说明使用视图和触发器的情况，出错信息（获取手段、分类编码）及处理方法，隐含的假设，容易出现二义性的概念，应该如何，不容许如何……

3.1.9 屏幕布局设计与说明

【说明】参看《概要设计说明书》。可采取两种方式：

绘图：使用计算机绘图工具。

拷贝屏幕：为减少文件的长度，应保存单色图形，但必须具有多级灰度，以保证显示与打印的效果。

对屏幕中的组件，应逐一进行说明。

第二节　前端模块

3.2.1 模块编号与中文注释

【说明】应与第二章第二节的表述一致。

3.2.2 功能描述与性能描述

【说明】功能描述：本叶模块的主要功能。

性能描述：精度指标、响应速度指标、数据吞吐量指标……

前端模块的主要功能即为：实现一个前端界面，将前端界面与后台实现的核心功能进行连接，并且显示主要窗口，在窗口中显示我们程序的各种核心功能，比如抓包，更改各类参数等。这里前端大体分为几个部分，首先一个主要窗口，这个窗口有主要的设置信息，包括限制嗅探环境、设置 IP 地址、设置 PORT、设置包类型等。并且支持在进行一次嗅探后更改参数多次嗅探，主要界面如下图所示：

在主界面中，可以多次更改需要的参数，而直到点击最终的开始嗅探，所有的参数才会被确定，在本次嗅探结束前不可更改。

主界面的性能描述：这个界面不需要实时处理太多参数，最终显示的嗅探结果也不再这个界面上，因此只需要保证正常的输出不会导致界面卡死即可。

其次是一个显示网卡的界面的主要功能：这个界面主要为了使用户确定自己需要嗅探的网卡，这个程序支自定义持网卡，这也是十分关键的一部分，如果用户不输入网卡参数，会使用默认网卡进行嗅探，可能会导致得到与用户期望不符的数据，因此不论进行什么类型包的嗅探工作，都建议首先执行此功能进行网卡查看与选择，这里点击后会跳转到另一个 python 程序中，这个程序会新建一个全新的界面，对本机的网卡进行扫描，并返回扫描到的结果，将其打印在窗口中，并且，会在这个窗口中将各个网卡进行命名，可以根据命名的数字结果进行选择。

选择网卡的界面如下图：

显示网卡的性能描述：这里显示的网卡功能，仅仅在程序开始时需要执行一次，确定网卡后如果不需要更改则无需重新打开这个界面，因此对性能要求并不高，能够正常执行并显示参数即可。

最后一个界面是开始嗅探的界面：这个界面的功能就是在用户选择完成相应参数信息后，可以选择这个功能开始嗅探工作，就会跳转到这个页面对刚才用户的选择结果的对应包进行嗅探工作，这里首先和上面一样需要一个能够显示信息的窗口，用来接收嗅探的结果，这里采用和上面一样的窗口，接受的数据由 shell 中获得，即将 shell 中本应显示的数据显示在这个界面上，在停止这个功能后本窗口就关闭回到主页面等待下次执行。

显示嗅探界面如下：

开始嗅探界面性能描述：对于这个界面，是本项目几个界面中要求最够的一个界面，j 不仅需要在窗口实时显示各种类型的数据包的内容，将这些打印数据从 shell 中重定向到这个界面，而且还需要保证在数据量极大的情况下这个窗口依旧能够准确接收每个包的内容，不能发生卡死的情况，这里仅使用一个管道加重定向数据流的方式不足以接收这个大量的数据内容，因此我们采取了一种更少见但是有效的方式，使用了队列来接收内容，首先定义一个队列，结合线程来使用缓存部分接收读取到的内容，当内容达到一定数量的时候，就将缓存中的数据进行打印，并更新缓存重新接受内容，使用这个方式就完美的解决了当数据量大时无法及时打印的问题

3.2.3 与本模块相关的代码表和表

【说明】

名称	中文注释	类型		作用
		代码表	表	
main.py	主界面	✔		这是程序的主界面，显示各种参数
showDevice.py	显示网卡	✔		这是显示网卡界面
beginSniff.py	开始嗅探	✔		这是显示嗅探的界面

作用：input、 output 、update 等。

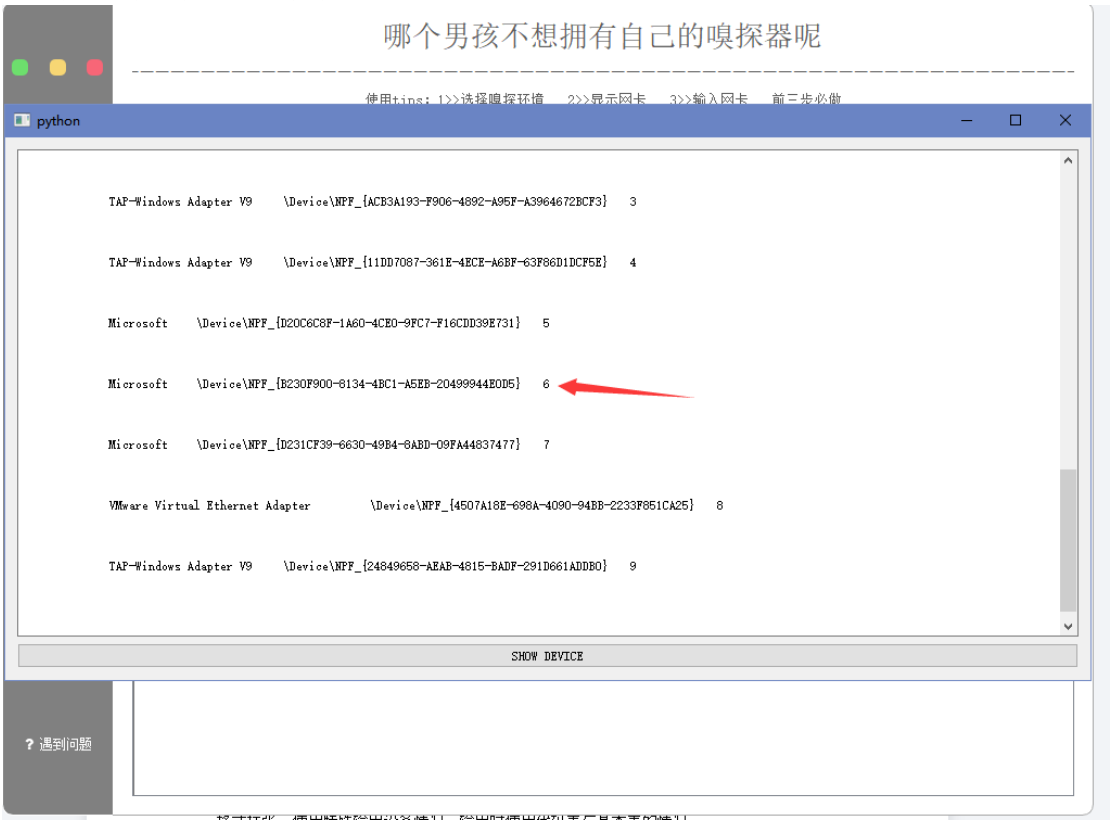
3.2.4 输入信息

【说明】参数（含参数名、中文注释、缺省值、格式）、数据文件的格式与权限、输入频度。使用特殊输入设备情况。输入时使用代码表与基本表的情况。

这里定义了需要显示的所有参数，首先在开始执行这个程序的时候不需要定义任何额外参数，就可以打开主界面，如下：

随后按照执行顺序，首先需要选择：嗅探环境选择按钮，对当前的嗅探环境进行定义，这里可以选择两种支持的嗅探环境：Linux 和 Windows 环境：

默认使用 Windows 环境，随后必须选择显示网卡界面，查看当前环境下的所有网卡信息，这里的信息会根据顺序由数字标识：



这里选择使用的 6 号网卡，随后需要在选择网卡界面输入确定的网卡的数字，如果这里不输入信息，将按照默认使用 1 号网卡

剩下的所有输入信息均为可选输入，如果均不输入则为默认状态

这里一一进行演示，首先输入 IP 信息：

可以选择对端口进行限制，这里选择 443 端口：

其次选择包类型，这里选择 TCP：

MD5 解码，可以选择开启或者关闭：

保存数据为默认打开选项

以上为所有需要输入的参数信息

3.2.5 输出信息

【说明】参数（含参数名、中文注释、缺省值、格式）、数据文件的格式、输出频度、报表格式样张。使用特殊输出设备情况。输出时使用代码表与基本表的情况。

本模块所有输出均在界面上，包括主界面的输出，显示网卡的输出和嗅探结果的输出，这里分别进行展示

3.2.6 算法

【说明】包括计算公式与说明、某些设定的或必然的逻辑关系。对于函数，要着重说明。

这里详细说明实现这个界面的核心算法部分：

首先创建一个新窗口，定义相关部件，比如定义左侧按钮，以及按钮上的字，和相应的点击函数

```
def init_ui(self):
    self.frame = []  # 存图片
    self.detectFlag = False  # 检测flag
    self.cap = []
    self.timer_camera = QTimer()     #定义定时器

    self.setFixedSize(1024,768)
    self.main_widget = QtWidgets.QWidget()  # 创建窗口主部件
    self.main_layout = QtWidgets.QGridLayout()  # 创建主部件的网格布局
    self.main_widget.setLayout(self.main_layout)  # 设置窗口主部件布局为网格布局

    self.left_widget = QtWidgets.QWidget()  # 创建左侧部件
    self.left_widget.setObjectName('left_widget')
    self.left_layout = QtWidgets.QGridLayout()  # 创建左侧部件的网格布局层
    self.left_widget.setLayout(self.left_layout) # 设置左侧部件布局为网格

    self.right_widget = QtWidgets.QWidget() # 创建右侧部件
    self.right_widget.setObjectName('right_widget')
    self.right_layout = QtWidgets.QGridLayout()
    self.right_widget.setLayout(self.right_layout) # 设置右侧部件布局为网格

    self.main_layout.addWidget(self.left_widget,0,0,12,2) # 左侧部件在第0行第0列，占8行3列
    self.main_layout.addWidget(self.right_widget,0,2,12,10) # 右侧部件在第0行第3列，占8行9列
    self.setCentralWidget(self.main_widget) # 设置窗口主部件

    self.left_close = QtWidgets.QPushButton("") # 关闭按钮
    self.left_visit = QtWidgets.QPushButton("") # 空白按钮
    self.left_mini = QtWidgets.QPushButton("")  # 最小化按钮
    # self.left_maxi = QtWidgets.QPushButton("")  # 最da化按钮
    self.left_close.clicked.connect(self.close)
    self.left_visit.clicked.connect(self.showMaximized)
    # self.left_maxi.clicked.connect(self.showMaximized)
    self.left_mini.clicked.connect(self.showMinimized)
    self.left_button_1 = QtWidgets.QPushButton("嗅探环境选择")
    self.left_button_1.clicked.connect(self.ChoSniffEnv)
    self.left_button_1.setObjectName('left_button')
    self.left_button_2 = QtWidgets.QPushButton("输入目的IP")
    self.left_button_2.setObjectName('left_button')
    self.left_button_2.clicked.connect(self.input_IP)
    self.left_button_3 = QtWidgets.QPushButton("限定端口")
    self.left_button_3.setObjectName('left_button')
    self.left_button_3.clicked.connect(self.input_port)
    self.left_button_4 = QtWidgets.QPushButton("选择目的包类型")
    self.left_button_4.setObjectName('left_button')
    self.left_button_4.clicked.connect(self.type_datapackage)
    self.left_button_5 = QtWidgets.QPushButton("MD5解码")
    self.left_button_5.clicked.connect(self.needMD5)
    self.left_button_5.setObjectName('left_button')
    self.left_button_6 = QtWidgets.QPushButton("保存数据")
    self.left_button_6.setObjectName('left_button')
    self.left_button_6.clicked.connect(self.save_data)
    self.left_button_7 = QtWidgets.QPushButton("开始嗅探")
    self.left_button_7.clicked.connect(self.startSniff)
    self.left_button_7.setObjectName('left_button')
```

定义整个窗口为两个部分，分为左右，两部分分别使用网格布局，在右侧布局上面首先写入几个简单的使用说明，在说明下面需要创建一个可以滚动的多行文本框，设置不可输入，这里用来显示各种定义后的参数信息。

```python
self.right_bar_widget = QtWidgets.QWidget() # 右侧顶部搜索框部件
self.right_bar_layout = QtWidgets.QGridLayout() # 右侧顶部搜索框网格布局
self.right_bar_widget.setLayout(self.right_bar_layout)
self.right_layout.addWidget(self.right_bar_widget, 0,0,0,0)#使用说明的文本换行自适应大小

self.ins = QtWidgets.QLabel("哪个男孩不想拥有自己的嗅探器呢")#这里需要添加读取一个txt文件的内容并显示
self.right_bar_layout.addWidget(self.ins,0,0,1,0)
self.ins.setStyleSheet("color:rgb(94,94,94)")
self.ins.setFont(QFont("MFLiHei_Noncommercial-Regular",20))
self.ins.setAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignTop)

self.segmentation = QtWidgets.QLabel("------------------------------------------------------------------------------------------------------------------------
self.right_bar_layout.addWidget(self.segmentation,1,0,1,1)
# self.segmentation.setStyleSheet("color:red")
# self.segmentation.setFont(QFont("方正姚体 常规",12,63))
self.segmentation.setAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignTop)
self.segmentation.setStyleSheet("color:rgb(94,94,94)")
self.segmentation.setFont(QFont("MFLiHei_Noncommercial-Regular",20))


self.segmentation3 = QtWidgets.QLabel("使用tips：1>>选择嗅探环境   2>>显示网卡   3>>输入网卡   前三步必做")#这里需要添加读取一个txt文件的内容并显示
self.right_bar_layout.addWidget(self.segmentation3,2,0,1,1)
# self.segmentation.setStyleSheet("color:red")
# self.segmentation.setFont(QFont("方正姚体 常规",12,63))
self.segmentation3.setAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignTop)
self.segmentation3.setStyleSheet("color:rgb(94,94,94)")
self.segmentation3.setFont(QFont("MFLiHei_Noncommercial-Regular",10))
self.segmentation4 = QtWidgets.QLabel("使用tips：其余各项可以按需输入>>>敏感包类型为:tcp==default(http\\ftp\\telnet)")#这里需要添加读取一个txt文件的内容并显示
self.right_bar_layout.addWidget(self.segmentation4,3,0,1,1)
# self.segmentation.setStyleSheet("color:red")
# self.segmentation.setFont(QFont("方正姚体 常规",12,63))
self.segmentation4.setAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignTop)
self.segmentation4.setStyleSheet("color:rgb(94,94,94)")
self.segmentation4.setFont(QFont("MFLiHei_Noncommercial-Regular",10))

self.segmentation2 = QtWidgets.QLabel("------------------------------------------------------------------------------------------------------------------------
self.right_bar_layout.addWidget(self.segmentation2,4,0,1,1)
# self.segmentation.setStyleSheet("color:red")
# self.segmentation.setFont(QFont("方正姚体 常规",12,63))
self.segmentation2.setAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignTop)
self.segmentation2.setStyleSheet("color:rgb(94,94,94)")
self.segmentation2.setFont(QFont("MFLiHei_Noncommercial-Regular",20))

self.scrollArea_2 = QtWidgets.QScrollArea()
self.right_bar_layout.addWidget(self.scrollArea_2,5,0,1,1)
# self.scrollArea_2.setGeometry(QtCore.QRect(10, 20, 731, 301))
self.scrollArea_2.setWidgetResizable(True)
self.scrollArea_2.setObjectName("scrollArea_2")
self.scrollAreaWidgetContents_2 = QtWidgets.QWidget()
self.scrollAreaWidgetContents_2.setGeometry(QtCore.QRect(0, 0, 729, 320))
self.scrollAreaWidgetContents_2.setObjectName("scrollAreaWidgetContents_2")
self.textEdit = QtWidgets.QTextEdit(self.scrollAreaWidgetContents_2)
# self.textEdit.adjustSize()
self.textEdit.setGeometry(QtCore.QRect(0, 0, 870, 650))
self.textEdit.setObjectName("textEdit")
self.textEdit.setReadOnly(True)
self.scrollArea_2.setWidget(self.scrollAreaWidgetContents_2)
```

随后设置整个窗口的相关参数，使得整体窗口的美观得以提升

```
        self.left_widget.setStyleSheet('''
    QPushButton{border:none;color:white;}
    QPushButton#left_label{
        border:none;
        border-bottom:1px solid white;
        font-size:18px;
        font-weight:700;
        font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
    }
    QPushButton#left_button:hover{border-left:4px solid red;font-weight:700;}
    QWidget#left_widget{
    background:gray;
    border-top:1px solid white;
    border-bottom:1px solid white;
    border-left:1px solid white;
    border-top-left-radius:10px;
    border-bottom-left-radius:10px;

')
        self.right_widget.setStyleSheet('''
    QWidget#right_widget{
        color:#232C51;
        background:white;
        border-top:1px solid darkGray;
        border-bottom:1px solid darkGray;
        border-right:1px solid darkGray;
        border-top-right-radius:10px;
        border-bottom-right-radius:10px;
    }
    # QLabel#right_lable{
    #     border:none;
    #     font-size:16px;
    #     font-weight:700;
    #     font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
    # }
    ins{
        background:red;
    }
')
        # self.setWindowOpacity(0.9) # 设置窗口透明度
```

为两个点开后有新的单选框的按钮函数创建新类，并且设置相关的点击结果，这个类是定义的选择嗅探环境，默认选择为 Windows 环境，当点击另一个按钮的时候，会把这个参数传进一个全部变量，方便其他函数进行调用

```
class secWin_choSniffEnv(QWidget):
    def __init__(self):
        super(secWin_choSniffEnv, self).__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Select the sniffing environment')
        self.resize(300, 50)

        layout = QHBoxLayout()  # 水平布局

        self.btn1 = QRadioButton('Linux')
        self.btn1.toggled.connect(self.buttonState)  # 设置按钮的槽函数
        layout.addWidget(self.btn1)

        self.btn2 = QRadioButton('Windows')
        self.btn2.setChecked(True)  # 设置默认选中状态
        self.btn2.toggled.connect(self.buttonState)   # 设置按钮的槽函数
        layout.addWidget(self.btn2)

        self.setLayout(layout)  # 设置布局

    def buttonState(self):
        global envOfSniff
        radioButton = self.sender()  # 事件发送者，获得信号的发送者
        if radioButton.isChecked():  # 按钮是否被选中
            envOfSniff = radioButton.text()
            print("ENVIRONMENT >>>" + radioButton.text())
```

是否需要 MD5 加密的按钮与上一个按钮类同理：

```python
class ThiWin_needMD5(QWidget):
    def __init__(self):
        super(ThiWin_needMD5, self).__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Select the sniffing environment')
        self.resize(300, 50)

        layout = QHBoxLayout()  # 水平布局

        self.btn1 = QRadioButton('Yes')
        self.btn1.toggled.connect(self.buttonState)  # 设置按钮的槽函数
        layout.addWidget(self.btn1)

        self.btn2 = QRadioButton('No')
        self.btn2.setChecked(True)  # 设置默认选中状态
        self.btn2.toggled.connect(self.buttonState)   # 设置按钮的槽函数
        layout.addWidget(self.btn2)

        self.setLayout(layout)  # 设置布局

    def buttonState(self):
        global md5OfSniff
        radioButton = self.sender()  # 事件发送者，获得信号的发送者
        if radioButton.isChecked():  # 按钮是否被选中
            envOfSniff = radioButton.text()
            print("NEED MD5 CODING>>>" + radioButton.text())
```

最后在窗口类中完善每一个点击函数，包括各种参数设置的函数，点击后都将对参数进行保存，在嗅探时统一调用使用

```python
class MainUi(QtWidgets.QMainWindow):

    def __init__(self):
        super().__init__()
        self.init_ui()

    def input_IP(self):
        global ipOfSniff
        ipOfSniff, okPressed = QInputDialog.getText(self, "Sniff information input","The IP you want to sniff:", QLineEdit.Normal, "")
        print("IP >>>",ipOfSniff)

    # def hiddde(self):
    #     # self.input.setVisible(False)
    #     # print("1")
    #     pass

    def input_port(self):
        global portOfSniff
        portOfSniff, okPressed = QInputDialog.getText(self, "Sniff information input","The POrt you want to sniff:", QLineEdit.Normal, "")
        print("PORT >>>",portOfSniff)

    def addcourierinfo(self):
        os.system("python insertcourier.py")
```

```
def needMD5(self):
    thiWin.show()

def startSniff(self):
    print("当前的环境为>>>",envOfSniff)
    print("嗅探目标IP为>>>",ipOfSniff)
    print("嗅探目标端口为>>>",portOfSniff)
    print("是否解码MD5编码的用户名和密码>>>",md5OfSniff)
    print("嗅探包类型为>>>",packOfSniff)
    if(envOfSniff == 'Linux'):
        os.system("python3 beginSniff.py %s %s %s %s %s %s" % (envOfSniff,ipOfSniff,portOfSniff,md5OfSniff,packOfSniff,choTheDevice))
    elif (envOfSniff == 'Windows'):
        os.system("python beginSniff.py %s %s %s %s %s %s" % (envOfSniff,ipOfSniff,portOfSniff,md5OfSniff,packOfSniff,choTheDevice))
    # os.system("python beginSniff.py %s %s %s %s %s %s" % (envOfSniff,ipOfSniff,portOfSniff,md5OfSniff,packOfSniff,choTheDevice))
    print("..................")

def outputWritten(self, text):
    cursor = self.textEdit.textCursor()
    cursor.movePosition(QtGui.QTextCursor.End)
    cursor.insertText(text)
    self.textEdit.setTextCursor(cursor)
    self.textEdit.ensureCursorVisible()

def type_datapackage(self):
    global packOfSniff
    packOfSniff, okPressed = QInputDialog.getText(self, "Sniff information input","The data_packet you want to sniff:", QLineEdit.Normal, "")
    print("data_packet",packOfSniff)
    # print("okPressed is",okPressed)


def save_data(self):
    os.system("python change.py")

def chooseDevice(self):
    global choTheDevice
    choTheDevice, okPressed = QInputDialog.getText(self, "Sniff information input","The DEVICE you want to sniff:", QLineEdit.Normal, "")
    print("DEVICE Num >>>",choTheDevice)
```

最后实现一个开始函数，创建整个程序的起始地点：

```
if __name__ == '__main__':

    # main()
    app = QtWidgets.QApplication(sys.argv)
    gui = MainUi()
    secWin = secWin_choSniffEnv()
    thiWin = ThiWin_needMD5()
    gui.show()
    sys.exit(app.exec_())
```

对于点击开始嗅探的函数：

首先接收对于前面 main.py 中的各种参数，随后根据参数对调用的 c 文件的命令行进行构造

```
def startSniff():
    global start
    env = sys.argv[1]
    ip = sys.argv[2]
    port = sys.argv[3]
    md5 = sys.argv[4]
    pack = sys.argv[5]
    flag = sys.argv[6]
    if(env == 'Windows'):
        if(ip == 'unknown' and port == 'unknown'):
            if(pack == 'unknown'):
                print('全局模式')
                start = 'MySniff.exe --eth=' + flag + ' --debug '
            elif(pack == 'tcp=fault' or pack == 'tcp=http' or pack == 'tcp=ftp' or pack == 'tcp=telnet'):
                print('抓取敏感数据模式')
                start = 'MySniff.exe --eth=' + flag + ' -- ' + pack + ' --debug'
            else:
                print(pack+' 协议模式')
                start = 'MySniff.exe --eth=' + flag + ' --' + pack + ' --debug'
        elif(ip != 'unknown' and port == 'unknown'):
            if(pack == 'unknown'):
                print('IP全局模式')
                start = 'MySniff.exe --eth=' + flag + ' --debug ' + ' --ip ' + ip
            elif(pack == 'tcp=fault' or pack == 'tcp=http' or pack == 'tcp=ftp' or pack == 'tcp=telnet'):
                print('IP抓取敏感数据模式')
                start = 'MySniff.exe --eth=' + flag + ' -- ' + pack + ' --debug' + ' --ip ' + ip
```

在开始前首先需要初始化队列和线程以及相应的重定向输出的函数

```
    # Create Queue and redirect sys.stdout to this queue
    queue = Queue()
    sys.stdout = WriteStream(queue)

    # Create QApplication and QWidget
    qapp = QApplication(sys.argv)
    app = MainUi()
    app.show()

    # Create thread that will listen on the other end of the queue, and send
    thread = QThread()
    my_receiver = MyReceiver(queue)
    my_receiver.mysignal.connect(app.append_text)
    my_receiver.moveToThread(thread)
    thread.started.connect(my_receiver.run)
    thread.start()

    qapp.exec_()

if __name__ == '__main__':
    startSniff()
```

重写部分类，使得输出能够成功打印在指定框中

```
class WriteStream(object):
    def __init__(self,queue):
        self.queue = queue

    def write(self, text):
        self.queue.put(text)

# A QObject (to be run in a QThread) which sits waiting for data to come through a Queue.Queue().
# It blocks until data is available, and one it has got something from the queue, it sends
# it to the "MainThread" by emitting a Qt Signal
class MyReceiver(QObject):
    mysignal = pyqtSignal(str)

    def __init__(self,queue,*args,**kwargs):
        QObject.__init__(self,*args,**kwargs)
        self.queue = queue

    @pyqtSlot()
    def run(self):
        while True:
            text = self.queue.get()
            self.mysignal.emit(text)
```

简单设计一个窗口，支持上下滑动，并且能够接收相应的数据

```
class MainUi(QWidget):
    def __init__(self,*args,**kwargs):
        QWidget.__init__(self,*args,**kwargs)
        self.layout = QVBoxLayout(self)
        self.setMinimumHeight(500)  #窗体最小高度
        self.setMinimumWidth(1000) #窗体最小宽度
        self.textedit = QTextEdit()
        self.button = QPushButton('start sniff')
        self.button.clicked.connect(self.start_thread)
        self.layout.addWidget(self.textedit)
        self.layout.addWidget(self.button)
        self.textedit.setReadOnly(True)
        # self.layout.setStretchFactor(self.textedit,4)
        # self.layout.setStretchFactor(self.button,2)

    @pyqtSlot()
    # def run(self):
    #     for i in range(1000):
    #         print(i)
    @pyqtSlot(str)
    def append_text(self,text):
        self.textedit.moveCursor(QTextCursor.End)
        self.textedit.insertPlainText( text )

    @pyqtSlot()
    def start_thread(self):
        self.thread = QThread()
        self.long_running_thing = LongRunningThing()
        self.long_running_thing.moveToThread(self.thread)
        self.thread.started.connect(self.long_running_thing.run)
        self.thread.start()
```

对于开启显示网卡的文件：

同样首先定义相应的队列和重定向函数

```
def startSniff():
    global start
    env = sys.argv[1]
    # Create Queue and redirect sys.stdout to this queue
    if(env == 'Windows'):
        start = 'MySniff.exe --eth'
    elif(env == 'Linux'):
        start = './MySniff --eth'
    queue = Queue()
    sys.stdout = WriteStream(queue)

    # Create QApplication and QWidget
    qapp = QApplication(sys.argv)
    app = MainUi()
    app.show()

    # Create thread that will listen on the other end of the queue, and send the text to the textedit in our app
    thread = QThread()
    my_receiver = MyReceiver(queue)
    my_receiver.mysignal.connect(app.append_text)
    my_receiver.moveToThread(thread)
    thread.started.connect(my_receiver.run)
    thread.start()

    qapp.exec_()

if __name__ == '__main__':
    startSniff()
```

接收传过来的参数，这里由于 linux 和 windows 在处理逻辑上不同，因此需要在使用这个函数之前首先确定嗅探环境，否则可能导致程序崩溃，重写类和重定向函数

```
class WriteStream(object):
    def __init__(self,queue):
        self.queue = queue

    def write(self, text):
        self.queue.put(text)

# A QObject (to be run in a QThread) which sits waiting for data to come through a Queue.Queue(
# It blocks until data is available, and one it has got something from the queue, it sends
# it to the "MainThread" by emitting a Qt Signal
class MyReceiver(QObject):
    mysignal = pyqtSignal(str)

    def __init__(self,queue,*args,**kwargs):
        QObject.__init__(self,*args,**kwargs)
        self.queue = queue

    @pyqtSlot()
    def run(self):
        while True:
            text = self.queue.get()
            self.mysignal.emit(text)

# An example QObject (to be run in a QThread) which outputs information with print
class LongRunningThing(QObject):
    @pyqtSlot()
    def run(self):
        subp=subprocess.Popen(start,shell=True,stdout=subprocess.PIPE)
        c=subp.stdout.readline()
        while c:
```

3.2.7 处理流程

【说明】可采用框图+文字叙述或 PAD 图+文字叙述。只要表达得清晰准确即可。

这里说明一下整个系统的运转流程：首先启动 main.py 即开始整个系统的工作，此时仅仅 main.py 进行工作，在这里进行相应参数设置，当选择显示网卡时，showDevice.py 被调用打开，显示一个新窗口，在新窗口中调用一个 C 语言文件显示网卡信息，关闭这个窗口就关闭这两个文件，返回主函数窗口，点击开始嗅探即调用 beginSniff.py 文件，对刚才传进来的参数进行处理并且开始嗅探功能，嗅探功能连接到一个 C 语言文件进行处理，返回的结果在这个新窗口中显示。关闭这个窗口后返回主界面继续等待，主界面可以一直进行参数修改多次进行嗅探。

3.2.8 应说明的问题与限制

【说明】说明使用视图和触发器的情况，出错信息（获取手段、分类编码）及处理方法，隐含的假设，容易出现二义性的概念，应该如何，不容许如何……

说明:这里对主界面的几个关键部分进行说明:首先在打开嗅探或者显示网卡的工作时，主界面将被挂起等待,这时不能提供任何功能或者选择任何参数,如果需要对参数进行修改，那么需要将打开的窗口关闭后才能继续设置。

另外，由于上文所说，主界面在执行其他功能时会被挂起，因此在关闭新窗口后需要适当等待几秒钟，待挂起结束后方可对主界面进行操作，如果出现未响应请勿关闭窗口，会导致程序崩溃，等待几秒钟后就可恢复正常运行。

3.2.9 屏幕布局设计与说明

整体布局类型为将整个界面分为左右两个部分，每个部分分别使用网格布局单独设置

整体效果如上，左侧首先设置三个按钮，分别对应最小化最大化关闭，下面是顺序排列的几个按钮，右侧首先是标题和简要使用说明，下面是一个框体，用来显示设置参数的结果。