



广西师大出校控制器/多运营 商拨号系统原理分析及实现 (基于QT和Python)

Author:xzppmail@gmail.com



目 录

1	设计目的	1
2	基础工具介绍	1
2.1	QT	1
2.2	WireShark	1
2.3	OllyDebug	2
2.4	Python	2
3	校园网部分协议分析与实现	3
3.1	总体原理分析	3
3.2	握手过程分析	5
3.2.1	0x1f数据包	5
3.2.2	0x20数据包	6
3.2.3	0x21数据包	7
3.2.4	0x22数据包	8
3.3	握手过程中的加密分析	9
3.4	握手出校认证的代码实现	10
3.4.1	UDP数据类	10
3.4.2	0x1f数据包的发送	10
3.4.3	0x20数据包的接收	11
3.4.4	0x21数据包的发送	12
3.4.5	0x22数据包的接收	13
3.5	心跳包分析与实现	15
3.6	心跳包的实现	16
3.7	费用信息的解析	17
4	宿舍宽带部分分析与实现	19
4.1	原理分析	19
4.2	校验码的计算	19
5	测试结果以及总结	24
6	总结	24

附录	26
附录 A 密文摘要中参数的计算	26
附录 B 密文摘要计算部分反汇编	26
附录 C Python版本源码	26

1 设计目的

鉴于学校官方的出校控制器只有Windows版本，而Linux下只提供命令行版本，并且不支持运营商宽带拨号功能，而且没有支持OS X版本，所以在Macbook上上网成了一个。考虑到QT的跨平台特性，所以采用了QT来进行开发。

2 基础工具介绍

虽然有相关的文档，但是只能针对桂电适用，并不能在我校百分百适用，总体来说，中间的通信机制类似一个黑盒，仍然需要借助许多工具来对这个“黑盒”进行逆向。而这些工具则包括了开发的平台QT，抓包软件Wireshark用于捕捉官方出校控制器的数据包，OllyDebug是一个动态的反汇编软件，用于反向官方出校控制器相关算法。Python则是一个编码风格简易的语言，这里用于数据包加密过程相关参数的破解计算。

2.1 QT

Qt是一个跨平台应用程序和UI开发框架。使用Qt只需要一次性开发程序，无须重新编写源代码，可跨不同桌面和嵌入式操作系统部署这些应用程序。经过多年发展，Qt不但拥有了完善的C++图形库，而且近年来的版本逐渐集成了数据库、OpenGL库、多媒体库、网络、脚本库、XML库、WebKit库等等，其核心库也加入了进程间通信、多线程等模块，极大的丰富了Qt开发大规模复杂跨平台应用程序的能力，真正意义上实现了其研发宗旨“Code Less; Create More; Deploy Anywhere.”。

2.2 WireShark

Wireshark是一个网络封包分析软件。网络封包分析软件的功能是抓取网络封包，并尽可能显示出最为详细的网络封包资料。Wireshark在Windows下使用WinPCAP作为接口，直接与网卡进行数据报文交换，在Linux/Unix下采用tcpdump 作为接口，直接与系统内核进行数据报文交换。

数据包分析软件的功能类似于万用表来量测电流、电压、电阻的工作，但是只是将场合移植到网络上，并将电线替换成网络线。在过去，网络封包分析软件是非常昂贵的软件。但是WireShark 的出现改变了这一切。在GNU/GPL通用许可证的保障范围底下，使用者可以以免费的代价取得软件与其源代码，所以WireShark 是目前全世界最广

泛的数据包分析软件之一。

Wireshark既可以作为黑客的工具，也可以做为初学者学习网络的工具，而在这里，我们作为一种分析工具，用来对出校控制器这个软件特定的协议进行分析。图(2.1)就是Wireshark 的图标。



图 2.1 Wireshark图标

2.3 OllyDebug

OllyDbg是一个新的动态追踪工具，将IDA与SoftICE结合起来的的思想，Ring 3级调试器，非常容易上手，已代替SoftICE成为当今最为流行的调试解密工具了。同时还支持插件扩展功能，是目前最强大的调试工具。界面如图(4.2)所示。

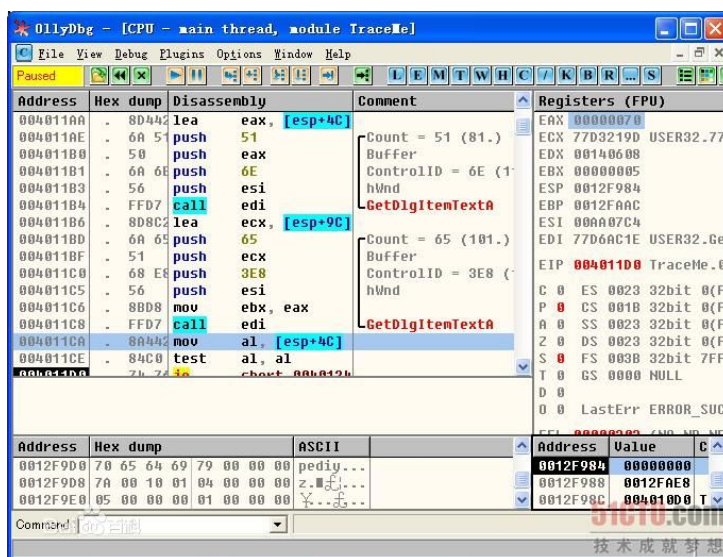


图 2.2 OllyDbg界面

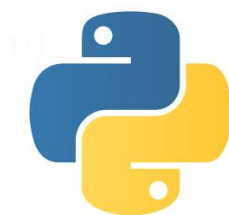
2.4 Python

Python是一种面向对象、解释型计算机程序设计语言，由Guido van Rossum于1989年底发明，第一个公开发行人版发行于1991年，Python 源代码同样遵循GPL(GNU General

Public License)协议。Python语法简洁而清晰，具有丰富和强大的类库。它常被昵称为胶水语言，能够把用其他语言制作的各种模块（尤其是C/C++）很轻松地联结在一起。

正因为Python跨平台、代码简洁的特性，所以Python在实现原型方面有着独特的优势，可以绕过所有底层的细节去进行功能验证。所以在这里采用Python进行特定数据细节分析，并且试着用Python实现了出校控制器的功能。

Python的英文原意是蟒蛇，所以用蛇作为其语言的Logo。



3 校园网部分协议分析与实现

3.1 总体原理分析

在测试的过程中发现，Windows和Linux版本在数据包结构和端口方面存在一定的差异，但是服务器应该可以识别出来，所以对最终的实现并没有影响。这里数据包的抓包都是基于官方Linux版本下取得的。

通过图(3.1)可以看出，目标服务器地址为202.193.160.123(广西师范大学)，采用UDP包来进行通信，其中No.5995, 5996, 6080, 6081这4个包为握手包，用于确认用户信息的正确性，大小为342 字节，除去包的头部信息之后，实际数据段为300字节。

No.	Time	Source	Destination	Protocol
5995	19.196942000	172.19.95.196	202.193.160.123	UDP
5996	19.200236000	202.193.160.123	172.19.95.196	UDP
6080	19.697268000	172.19.95.196	202.193.160.123	UDP
6081	19.704525000	202.193.160.123	172.19.95.196	UDP
14987	59.241931000	172.19.95.196	202.193.160.123	UDP
14988	59.245343000	202.193.160.123	172.19.95.196	UDP

图 3.1 握手包

而No.14987和No.14988为心跳包，包的的实际数据为500字节，这个数据包的发送表明了客户端还存活，然后服务器返回该客户的费用和流量信息。在数据包的发送过程中，如果包大小不正确，服务器是不会回应的。

根据桂电的相关文档, 以及图(3.2)可以知道 $No.5995$ 为连接请求包, 此时客户端向服务器5300端口发送一个包含用户名的连接请求, 服务器返回 $No.5996$ 包, 这个包里包含着用于计算摘要的密钥 Key^* , 客户端将这个 Key , 用户名和密码利用MD5算法计算得到相应的摘要密文, 并将这个密文发送给服务器, 即 $No.6080$ 包的数据(因为抓包时没有过滤, 可以看到 $No.5996$ 到 $No.6080$ 存在几个包的间隙, 也正是MD5计算过程导致的时延), 最终服务器返回 $No.6081$ 包中高速客户端是否开放成功, 从而完成一整套握手过程。

在开放成功后, 客户端每隔40秒和服务器交换信息, 端口为5301。事实上, 这个时间可以提高两分钟以减少客户端开销。 $No.14987$ 包中则是对客户端握手过程中的密钥 Key 进行简单计算后返回给服务器的。如果包正确, 则服务器返回 $No.14988$ 数据包, 里面带有当前ip的流量和余额信息。

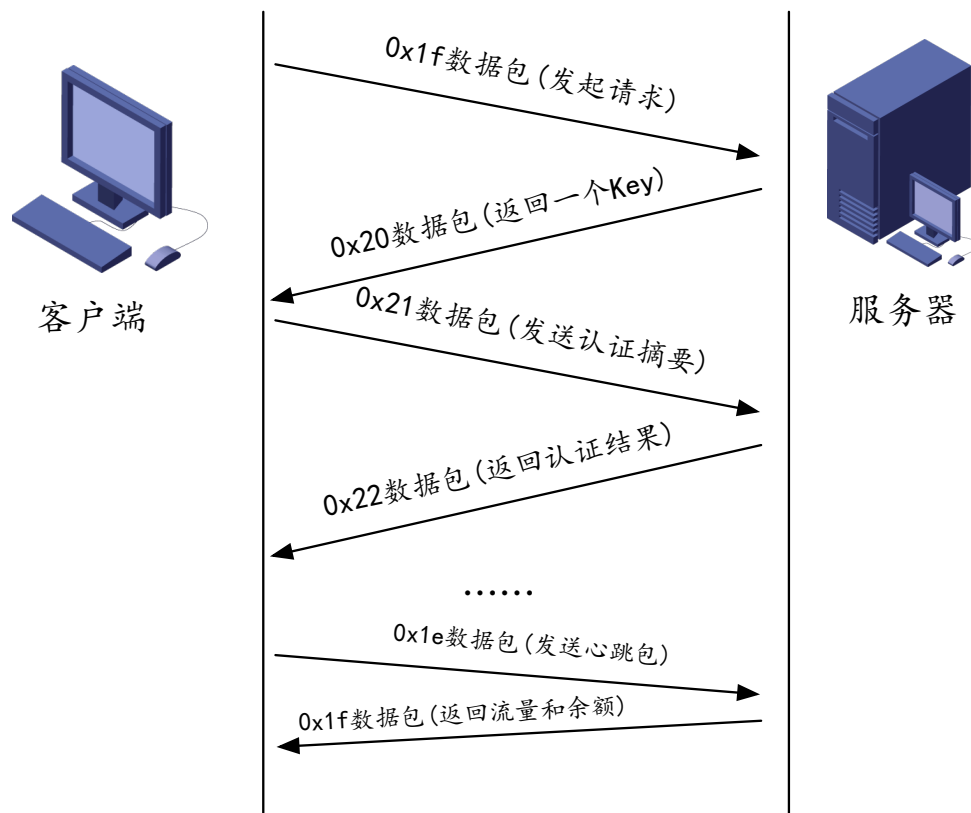


图 3.2 握手过程

*沿用桂电学生的说法, 严格意义来讲, 称做salt更合适, 本质上就是加盐计算摘要, 增加逆向的难度

3.2 握手过程分析

3.2.1 0x1f数据包

表(3-1)是请求数据包的数据，其中数据包的每个字段解释如下：

表 3-1 0x1f数据包

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	82	23	1f	00	00	00	00	00	00	00	00	0a	00	00	00	28
0x10	26	27	2a	26	27	26	2f	2d	26	0b	00	00	00	21	40	23
0x20	24	25	25	5e	26	2a	28	29	07	00	00	00	71	77	65	72
0x30	74	79	75	39	30	00	00	01	00	00	00	06	00	00	00	41
0x40	53	44	46	47	48	00	00	00	00	00	00	00	00	00	00	00
0x50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

0x00~0x01字节 所有与控制器相关的通信包都是0x82,0x23这两个常数

0x02字节 数据包的标号，这里是0x1f表示客户端请求开放ip地址

0x03~0x0A字节 全部为0，固定格式

0x0B~0x0E字节 0x0B字节代表下一段数据长度，即用户名长度，用户名为学号2014010970，所以0x0B字节数值为10(0x0a)

0x0F~0x18字节 用户名每一个字节的ASCII码减去10(0x0a)填入其中，例如'2'的ASCII码为0x32，变换后0x32-0x0a=0x28

0x19~0x1C字节 下一个数据段长度为11(0x0b)

0x1D~0x27字节 这一段的ASCII码为'#\$%%&()'是固定的，只是用来填充，并无实际意义

0x28~0x2B字节 下一个数据段长度为7(0x07)

0x2C~0x32字节 这一段的ASCII码为"qwerty",即键盘从左到右的顺序，估计只是用来进行填充，并无实际含义

0x33~0x44字节 这一段也只是字节的填充，其中0x3F~0x44的ASCII码为"ASDFGH",无实际意义，只需要的照搬即可

0x44~0x12B 全部为0

3.2.2 0x20数据包

表(3-2)是服务器对0x1f包的响应，包的标号为0x20，其中数据包的每个字段解释如下：

表 3-2 0x20数据包

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	82	23	20	00	00	00	00	00	00	00	00	0a	00	00	00	28
0x10	26	27	2a	26	27	26	2f	2d	26	0b	00	00	00	21	40	23
0x20	24	25	25	5e	26	2a	28	29	07	00	00	00	71	77	65	72
0x30	74	79	75	5c	20	00	00	01	00	00	00	06	00	00	00	02
0x40	00	00	00	47	48	00	00	00	00	00	00	00	00	00	00	00
0x50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

0x00~0x01字节 所有与控制器相关的通信包都是0x82,0x23这两个常数

0x02字节 数据包的标号，这里是0x20表示服务器响应的请求

0x03~0x0A字节 全部为0，固定格式

0x0B~0x0E字节 0x0B字节代表下一段数据长度，即用户名长度，用户名为学号2014010970，所以0x0B字节数值为10(0x0a)

0x0F~0x18字节 用户名每一个字节的ASCII码减去10(0x0a)填入其中，从0x1f包中原样返回

0x19~0x1C字节 下一个数据段长度为11(0x0b)

0x1D~0x27字节 这一段的ASCII码为’#\$\$%&()’是固定的，只是用来填充，并无实际意义，从0x1f包中原样返回

0x28~0x2B字节 下一个数据段长度为7(0x07)

0x2C~0x32字节 这一段的ASCII码为”qwerty”,即键盘从左到右的顺序，估计只是用来进行填充，并无实际含义，从0x1f包中原样返回

0x33~0x34字节 需要注意的是，这里是0x33~0x34字节里就是用于计算的密码摘要的密钥Key，用于计算相应的摘要，摘要的计算见第3.3节。接下来客户端将带用户名和密码的摘要返回给服务器进行认证。

0x35~0x12B字节 全部为0

3.2.3 0x21数据包

表(3-3)是服务器对0x1f包的响应，包的标号为0x20，其中数据包的每个字段解释如下：

表 3-3 0x21数据包

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	82	23	21	00	00	00	00	00	00	00	00	0e	00	00	00	39
0x10	67	64	74	34	33	37	34	35	77	72	77	71	72	1e	00	00
0x20	00	41	36	43	38	31	43	30	42	42	31	39	37	43	42	41
0x30	35	33	45	44	43	43	42	44	31	32	32	44	38	35	41	11
0x40	00	00	00	31	31	3a	32	32	3a	33	33	3a	34	34	3a	35
0x50	35	3a	36	36	2d	1f	d6	03	cc	f2	24	00	0a	00	00	00
0x60	71	77	65	72	74	79	75	69	6f	70	00	00	00	00	00	00
0x70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

0x00~0x01字节 所有与控制器相关的通信包都是0x82,0x23这两个常数

0x02字节 数据包的标号，这里是0x21表示客户端向服务器发出带用户名和密码的摘要

0x03~0x0A字节 全部为0，固定格式

0x0B~0x0E字节 0x0B处字节代表下一段数据长度，下一段数据段长度为14(0x0e)

0x0F~0x1C字节 字符串'9gdt43745wrwqr'，仍然是一段填充

0x1D~0x20字节 下一个数据段长度为30(0x1e)

0x21~0x3E字节 字符串'A6C81C0BB197CBA53EDCCBD122D85A'，为用户名和密码通过计算之后得到的摘要(算法见第3.3节)为了保证用户名和密码不以明文出现在网络上*，所以通过取Hash之后进行传输。

*事实上，这一套体系是存在问题的，即用户名进行明文传输的情况下，一旦知道了加密算法，就可以暴力破解出密码，见??

0x3F~0x42字节 下一段数据长度为17(0x11)

0x43~0x53字节 字符串'11:22:33:44:55:66'，估计设计时考虑到加入MAC地址所保留的字段，但是为简化编程难度，直接以11:22:33:44:55:66 替代

0x54~0x5B字节 不清楚含义，填充随机数都没有问题

0x5C~0x5F字节 下一段数据长度10(0x0a)

0x60~0x6A字节 字符串'qwertyuiop'，键盘最上面一排字母顺序，用于填充

0x6B~0x12B字节 全为0

3.2.4 0x22数据包

表(3-4)是0x22数据包，代表服务器的认证结果，其中数据包的每个字段解释如下

表 3-4 0x22数据包

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	82	23	22	00	00	00	00	00	00	00	00	0e	00	00	00	39
0x10	67	64	74	34	33	37	34	35	77	72	77	71	72	1e	00	00
0x20	00	41	36	43	38	31	43	30	42	42	31	39	37	43	42	41
0x30	35	33	45	44	43	43	42	44	31	32	32	44	38	35	41	11
0x40	00	00	00	31	31	3a	32	32	3a	33	33	3a	34	34	3a	35
0x50	35	3a	36	36	2d	1f	d6	3	cc	f2	24	00	0a	00	00	00
0x60	0c	00	00	00	bf	aa	b7	c5	49	50	b3	c9	b9	a6	23	30
0x70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

0x00~0x01字节 所有与控制器相关的通信包都是0x82,0x23这两个常数

0x02字节 数据包的标号，这里是0x22表示客户端的认证结果，通过试验可以知道0x00代表开放成功，0x14代表余额不足，0x63代表用户名或密码错误，0x20代表账号在别的机器上登录。

0x03字节 如果为0代表开放成果，如果为其它数值表明认证失败

0x0B~0x0E字节 0x0B处字节代表下一段数据长度，下一段数据段长度为14(0x0e)

0x0F~0x1C字节 字符串'9gdt43745wrwqr'，服务器原样返回

0x1D~0x20字节 下一个数据段长度为30(0x1e)

0x21~0x3E字节 字符串'A6C81C0BB197CBA53EDCCBD122D85A'，为用户名和密码通过计算之后得到的摘要，同理为服务器原样返回

0x3F~0x42字节 下一段数据长度为17(0x11)

0x43~0x53字节 字符串'11:22:33:44:55:66'，服务器原样返回

0x54~0x5B字节 不清楚含义，填充随机数都没有问题，也是原样返回

0x5C~0x5F字节 原样返回0x0a

0x60~0x6F字节 据说返回的是一段字符串，代表IP开放成功，可能是Unicode码，但是0x02字节中已经包含了认证结果，理解不理解已经不影响判定结果了。

0x70~0x12B字节 全为0

3.3 握手过程中的加密分析

需要说明的是，加密的计算过程在没有参考文献的情况下，需要通过反汇编来分析计算过程，而这里因为有桂电学生的相关文献进行参考，所以可以忽略这一步，但是对于某些参数仍需要通过数据包重新计算和确认。而学校运营商的宽带(第4节)，摘要的计算没有相关资料，只能通过反汇编进行跟踪。

下面说明密文摘要的计算过程，账号:2014010970，密码:042690

Step1 取得Key: Key为2字节数据，用unsigned short存储，来自0x20 数据包，见第3.2.2节，原始数据为5c 20，由于采用小端法，这里的

$$\text{Key}^* = 0x205C(8284_{10}) - 0x0d10(3344_{10}) = 0x134c(4940_{10})$$

Step2 第一轮MD5加密: 将Key的十进制(这里为4940)转换成字符串，与密码合并，计算MD5 HASH[†]

$$'4940' + '042690' = '4940042690'$$

$$\text{MD5}('4940042690') = 65e1daf13ad53e5b60b027be93d71663$$

*0x0d10这个参数可以根据已知数据算出，见附录 A，反汇编结果也能证明这个参数，见附录 B第7行

[†]MD5加密原理可以参考任何一本密码学教程

Step3 第二轮MD5加密: 将第一轮MD5的前5个字节全部变成大写, 并与用户名合并, 计算MD5 HASH

'65E1D'+ '2014010970' = '65E1D2014010970'

MD5('65E1D2014010970')=a6c81c0bb197cba53edccbd122d85ad2

Step4 发送数据包: 对第二轮MD5加密的结果取前30个字节, 全部变成大写, 即

A6C81C0BB197CBA53EDCCBD122D85A

取ASCII码, 填入0x20号数据包(见第3.2.3节)0x21~0x3E字节处, 并发送数据包

3.4 握手出校认证的代码实现

3.4.1 UDP数据类

```

usocket=new QUdpSocket( this );
ispUdp=new QUdpSocket( this );
server=QHostAddress( "202.193.160.123" );
usocket->connectToHost( server ,5300);
myIP=usocket->localAddress().toString();
timeoutf=new QTimer( this );
livepacket=new QUdpSocket( this );
livepacket->connectToHost( server ,5301);

```

1
2
3
4
5
6
7
8

QT中采用QUdpSocket来实现UDP协议, 采用connectToHost来连接到主机, 而事实上, 由于UDP协议是并不是面向连接的协议, 所以connectToHost只是用于确认服务器的ip地址和端口, 并不能像TCP/IP协议那样返回主机的生存状态。

在该程序中实例usocket用于校园网的握手过程中的数据包的发送和接收, 实例livepacket用于心跳包的发送和接收, ispUdp用于宿舍宽带部分的发送和接收。

3.4.2 0x1f数据包的发送

```

void MainWindow::login( char *username )
{
    // char ipAddress[] = "202.193.160.123";
    unsigned char tail[] = { 0x0b, 0x00, 0x00, 0x00, 0x21,
                             0x40, 0x23, 0x24, 0x25, 0x25,
                             0x5e, 0x26, 0x2a, 0x28, 0x29,
                             0x07, 0x00, 0x00, 0x00, 0x71,
                             0x77, 0x65, 0x72, 0x74, 0x79,

```

1
2
3
4
5
6
7
8

```

        0x75,0x39,0x30,0x00,0x00,
        0x01,0x00,0x00,0x00,0x06,
        0x00,0x00,0x00,0x41,0x53,
        0x44,0x46,0x47,0x48};
    unsigned char request[300]={0};
    /** Construct Packet Header**/
    request[0]=0x82;request[1]=0x23;request[2]=0x1f;
    for (int i=3;i<15;i++) request[i]=0x00;
    request[11]=name.size(); //get the length of string
    int charlen=name.size();
    for (int i=0;i<strlen(username);i++) request[15+i]=username[i]-10;
    for (int j=0;j<44;j++) request[15+charlen+j]=tail[j];
    QByteArray data;
    for (int i=0;i<300;i++) data.append(request[i]);
    connect(usocket,SIGNAL(readyRead()),this,SLOT(keyreceive()));
    connect(timeoutf,SIGNAL(timeout()),this,SLOT(displaytimeout()));
    usocket->write(data);
    timeoutf->start(2000);
    //keyreceive();
}

```

根据第3-1节的分析,可以发现0x19位置往后的格式都是固定的,所以第4行到第12行首先将这组数据给定义出来,在构建包的时候直接补上即可。由于数据包的长度为300,所以在第13行中采用unsigned char数组来初始化数据,且全部为0。第15行则直接定义了头部0x82,0x23,0x1f这三个固定格式的数据。第17行用于获得用户名的长度,填入0x0b处。第19行则将用户名的ASCII减去10,填充到数据包中。第20行则直接将固定格式的尾部数据填充入数据包。由于QT的UDP类发送的数据类型是QByteArray,所以21-22则是将unsigned char类型转换成QByteArray类型。23行是做好接受准备,准备接收服务器返回的数据包。24行则是超时计时器开始工作,如果收不到服务器返回的数据包,则返回超时。25行则是通过write方法将数据发送出去。至此完成了请求的第一步过程。

3.4.3 0x20数据包的接收

```

void MainWindow::keyreceive()
{
    char key[300]={0};
    int calckey=0;
    unsigned short staus=0xff;
    usocket->read(key,300);
    int flag=0;
    for (int i=0;i<300-3;i++)
    {

```

```

if (((unsigned char)key[i]==0x82) &&((unsigned char)key[i+1]==0x23) && ((
    unsigned char)key[i+2]==0x20))
{
    //cout<<"key:"<<endl;
    //cout<<hex<<(unsigned short)(unsigned char)key[i+51]<<endl;
    //cout<<hex<<(unsigned short)(unsigned char)key[i+52]<<endl;
    calckey=(unsigned short)(unsigned char)key[i+51+(name.size()-10)]+((
        unsigned short)(unsigned char)key[i+52+(name.size()-10)]<<8);
    // livekey=(unsigned short)(unsigned char)key[i+52+(name.size()-10)]+((
        unsigned short)(unsigned char)key[i+51+(name.size()-10)]<<8);
    calckey=calckey-3344;
    livekey=calckey+1500;
    livekey=((livekey>>8) & 0x00ff)+((livekey<<8) & 0xff00);
    cout<<hex<<livekey<<endl;
    flag=1;
    buildLive();
    timeoutf->stop();
    break;
}
.....
}
.....
}

```

数据包的接收采用统一槽函数，见第3.4.1中的第23行，在接收到数据后，对数据包进行分析，找到数据包的的特殊标志，再进行进一步的处理。

以下是0x20数据包的处理代码，其中第9行是通过0x82,0x23,0x20三个标志位来判断是否为0x20 数据包，第14，16行用于取出计算摘要的一组随机数。第17，18行则计算心跳包的key，22行则生成心跳包。第20行则是改变标记变量，因为只有收到0x20数据包才回复数据，而其它数据包则不回复数据，所以才改变flag变量。

3.4.4 0x21数据包的发送

```

if (flag==1) //Receive key packet and send password
{
    QString calc=QString::number(calckey);
    calc.append(password);
    qDebug()<<calc;
    QByteArray md51;
    QString md52;
    QByteArray md53;
    QString md54;
    md51=QCryptographicHash::hash(calc.toLatin1(), QCryptographicHash::Md5);
    md52=md51.toHex();
    md52=md52.toUpper();
}

```

```

md52.truncate(5);
md52=md52.append(name);
// qDebug()<<tr(md52.toLatin1());
md53=QCryptographicHash::hash(md52.toLatin1(), QCryptographicHash::Md5);
md54=md53.toHex();
md54=md54.toUpper();
md54.truncate(30);
char keysend[300]={0};
char *mdhash;
mdhash=md54.toLatin1().data();
char temple[]={0x82,0x23,0x21,0x00,0x00,0x00,
               0x00,0x00,0x00,0x00,0x00,0x0e,0x00,0x00,
               0x00,0x39,0x67,0x64,0x74,0x34,0x33,0x37,
               0x34,0x35,0x77,0x72,0x77,0x71,0x72,0x1e,
               0x00,0x00,0x00,0xFF,0x74,0x34,0x33,0x37,
               0x35,0x42,0x38,0x32,0x35,0x37,0x44,0x44,
               0x31,0x35,0x30,0x45,0xFF,0x44,0x37,0x36,
               0x44,0x31,0x35,0x46,0x33,0x35,0x46,0x30,
               0x44,0x11,0x00,0x00,0x00,0x31,0x31,0x3a,
               0x32,0x32,0x3a,0x33,0x33,0x3a,0x34,0x34,
               0x3a,0x35,0x35,0x3a,0x36,0x36,0x2d,0x1f,
               0xd6,0x03,0xcc,0xf2,0x24,0x00,0x0a,0x00,
               0x00,0x00,0x71,0x77,0x65,0x72,0x74,0x79,
               0x75,0x69,0x6f,0x70};
// qDebug()<<md54.toLatin1();
for (int i=0;i<106;i++) keysend[i]=temple[i];
for (int i=0;i<30;i++) keysend[i+33]=mdhash[i];
QByteArray keydata;
for (int i=0;i<300;i++) keydata.append(keysend[i]);
usocket->write(keydata);
}

```

代码的第10行是第一轮HASH,第12行是全部转换成大写,第13行是截断,取前5个字符,14行则是与用户名进行合并,16行是进行最后一轮HASH,19行是取前30个字符。第42行则是发送数据包。

3.4.5 0x22数据包的接收

```

if (((unsigned char)key[i]==0x82) &&((unsigned char)key[i+1]==0x23) && ((unsigned char)key[i+2]==0x22))
{
    //cout<<"key:"<<endl;
    //cout<<hex<<(unsigned short)(unsigned char)key[i+51]<<endl;
    //cout<<hex<<(unsigned short)(unsigned char)key[i+52]<<endl;
    cout<<"receive_result"<<endl;
    flag=2;
    staus=(unsigned short)(unsigned char)key[i+3];
}

```



```

cout<<hex<<"0x"<<staus<<endl;
if ( staus==0x00)
{
    ui->label_4->setText( tr ("连接成功"));
    disconnect(usocket ,SIGNAL(readyRead()),this ,SLOT(keyreceive()));
    connect(timer ,SIGNAL(timeout()),this ,SLOT(live()));
    // timer->start(1000);
    timer->start(120000);
    disconnect(timeoutf ,SIGNAL(timeout()),this ,SLOT(displaytimeout())
    );
    timeoutf->stop();
}
else if ( staus==0x63)
{
    cout<<diss<<endl;
    if ( diss==1)
    {
        ui->label_4->setText( tr ("断开成功"));
        diss=0;
    }
    else
        ui->label_4->setText( tr ("错误的用户名或密码"));

    timer->stop();
    disconnect(timeoutf ,SIGNAL(timeout()),this ,SLOT(displaytimeout()));
    timeoutf->stop();
}
else if ( staus==0x20)
{
    ui->label_4->setText( tr ("账号已经被使用"));
    timer->stop();
    disconnect(timeoutf ,SIGNAL(timeout()),this ,SLOT(displaytimeout()));
    timeoutf->stop();
}
else if ( staus==0x14)
{
    ui->label_4->setText( tr ("余额不足"));
    timer->stop();
    disconnect(timeoutf ,SIGNAL(timeout()),this ,SLOT(displaytimeout()));
    timeoutf->stop();
}
break;
}

```

0x22数据包则是返回认证结果，第1行用于判断数据包是否是0x82,0x23,0x22类型，第8行返回认证结果存入staus变量当中，如果为0x00，则表明连接成功，接下来首先断开连接，在13行中用disconnect函数实现，然后启动定时器，定时执行live()槽函数发

送心跳包(心跳包的实现见第3.5节)，在第14行中实现。第16行则是确定心跳发送间隔为120秒，即两分钟。第17行用于停止超时计时器。

第20行到第48行是识别各种认证失败的消息。例如0x20代表账号被使用，0x14代表余额不足，但是断开连接是通过发送一组错误的用户名和密码来实现断开连接的，因此需要对断开连接过程中的密码错误和输入过程中的密码错误进行区分，因此引入diss变量来实现，如果diss为1，则意味着是断开连接过程中导致的认证失败，因此显示“断开成功”，而diss为0，意味着是真正的用户名和密码错误，则显示“错误的用户名和密码”。

3.5 心跳包分析与实现

心跳包的设计是为了确认客户端是否存活，在客户端非正常退出的情况下，可以自动断开，避免造成更大的损失。而心跳包的结构如表(3-5)所示，其中心跳包的长度为500字节。

表 3-5 客户端发送的心跳包

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	82	23	1e	28	19	00	00	00	00	00	00	e4	3e	86	02	00
0x10	00	00	00	5c	8f	c2	f5	f0	a9	df	40	0a	00	00	00	32
0x20	30	31	34	30	31	30	39	37	30	09	00	00	00	53	70	69
0x30	64	65	72	6d	61	6e	00	00	00	00	00	00	00	00	00	00

0x00～0x01字节 所有与控制器相关的通信包都是0x82,0x23这两个常数

0x02字节 心跳包的标号为固定常数0x1e

0x03～0x04字节 心跳包的密钥，是原始的key-3344+1500获得

0x05～0x0A字节 全为0

0x0B～0x1A字节 固定字符串，不明白含义

0x1B～0x1E字节 用户名字符串长度为10(0x0a)

0x1F～0x28字节 用户名的ASCII码

0x29～0x2C字节 下一段数据长度为9(0x11)

0x2D~0x35字节 字符串'Spiderman'。

0x70~0x1F3字节 全为0

表 3-6 服务器返回的数据包

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	82	23	1f	28	19	00	00	00	00	00	00	00	00	00	00	00
0x10	00	00	00	d9	78	e9	3c	f7	d0	23	40	00	00	00	00	00
0x20	00	00	00	00	00	00	00	00	00	00	00	01	00	00	00	20
0x30	01	00	00	00	20	00	00	00	00	00	00	00	00	00	00	00

0x00~0x01字节 所有与控制器相关的通信包都是0x82,0x23这两个常数

0x02字节 返回标号为固定常数0x1f

0x03~0x04字节 心跳包的密钥,

0x05~0x0A字节 全为0

0x0B~0x12字节 返回流量, double型, 遵循IEEE754标准

0x13~0x1A字节 返回余额, double型, 遵循IEEE754标准

0x1B~0x34字节 固定格式

0x35~0x1F3字节 全为0

3.6 心跳包的实现

```

void MainWindow::live ()
{
    connect(livepacket,SIGNAL(readyRead()),this,SLOT(showinfo()));
    livepacket->write(livedata);
    qDebug()<<"数据长度:";
    qDebug()<<livedata.size();
}
void MainWindow::buildLive ()
{
    cout<<"live"<<endl;
    unsigned char liveframe[500]={0};
    unsigned char mid[]={0xe4,0x3e,0x86,0x02,

```

1
2
3
4
5
6
7
8
9
10
11
12

```

                                0x00,0x00,0x00,0x00,
                                0x5c,0x8f,0xc2,0xf5,
                                0xf0,0xa9,0xdf,0x40};
liveframe[0]=0x82;liveframe[1]=0x23;liveframe[2]=0x1e;
cout<<"live_key:_0x"<<hex<<livekey<<endl;
liveframe[3]=(unsigned char)((livekey & 0xff00)>>8);
liveframe[4]=(unsigned char)livekey;
for (int i=0;i<16;i++) liveframe[i+11]=mid[i];
int pos=0;
for (int i=0;i<name.size();i++)
{
    liveframe[31+i]=name.toLatin1().data()[i];
    pos=31+i;
}
unsigned char spider[]={0x09,0x00,0x00,0x00,
                        0x53,0x70,0x69,0x64,
                        0x65,0x72,0x6d,0x61,
                        0x6e};
for (int i=0;i<13;i++) liveframe[pos+i+1]=spider[i];
livedata.clear();
for (int i=0;i<500;i++) livedata.append(liveframe[i]);
}

```

心跳包的实现包括两个部分，一个是数据包的建立，另一个是数据包的发送，发送是用live()函数来实现的，相对比较简单，最核心的部分则是第4行，发送数据。而第3行则是准备接收服务器返回的流量和费用信息。

数据包的建立是使用的buildlive()函数，其中16行是填充头部数据，第18，19行是填充key，因为key是一个两字节的数据，而数据包是4字节的unsigned char 类型为基本单位，所以需要使用移位进行填充。22-26行是填充用户名，第31行和第20行是填充固定格式的数据。第33行则是将数据填充到Public变量livedata 里面去。

3.7 费用信息的解析

```

void MainWindow::showinfo()
{
    char money2[500]={0};
    livepacket->read(money2,500);
    unsigned char money[500]={0};
    unsigned char *p=money;
    typedef union tagOct
    {
        double dData;
        int nData[2];
    }Oct;
    Oct monnum;
}

```

```

13  for (int i=0;i<500;i++)
14  {
15      if (money2[i]!=0xff) //to filter 0xff
16      {
17          (*p)=(unsigned char)money2[i];
18          p++;
19      }
20  }
21  // for (int i=0;i<500;i++) cout<<hex<<(unsigned short)money[i];
22  for (int i=0;i<500;i++)
23  {
24      if (((unsigned char)money[i]==0x82) &&((unsigned char)money[i+1]==0x23)
25          && ((unsigned char)money[i+2]==0x1f))
26      {
27          qDebug()<<"receive";
28          for (int j=0;j<4;j++)
29          {
30              monnum.nData[0]=monnum.nData[0]|(money[i+19+j]<<(j*8));
31          }
32          for (int j=0;j<4;j++)
33          {
34              monnum.nData[1]=monnum.nData[1]|(money[i+23+j]<<(j*8));
35          }
36          QString moneyString=QString::number(monnum.dData);
37          int pn=moneyString.indexOf(".");
38          moneyString.truncate(pn+3);
39          ui->label_4->setText(tr("剩余金额:")+moneyString+tr("元"));
40          disconnect(timeoutf,SIGNAL(timeout()),this,SLOT(displaytimeout()));
41          disconnect(livepacket,SIGNAL(readyRead()),this,SLOT(showinfo()));
42          timeoutf->stop();
43          break;
44      }
45  }
46  }

```

由于c++不能对double类型的数据进行位操作，而数据包里传过来的数据类型都是unsigned char 型的，所以需要采用共用体来实现对数据的解析。

第4行用于接收服务器反馈回来的数据，存入money变量中，第7~12行则是定义了一个tagOct结构体，一个dData的double变量与两个元素的int数组共用同一块内存区域，正好为32位类型。代码的第13~19行则是剔除数据中干扰，因为在测试的过程中发现，接收到的数据会出现不明原因的0xff干扰，所以受限需要将0xff从数据中删除。24行用于找到数据包头部的准确位置。27~35行是通过移位操作将每一个字节填充到tagOct这个共用体当中去。最后36行则将double型转换成QString类型，并通过第37，38行的带包

保留2位小数，最后执行第39行输出到界面。

4 宿舍宽带部分分析与实现

4.1 原理分析

学校的出校控制器也可以支持宿舍的宽带的功能，宿舍宽带的是采用pppoe协议来进行拨号的，而pppoe协议是通过向局域网广播的方式来确定服务器的，但是因为存在3家运营商，所以服务器会发生冲突。因此在拨号之前，出校控制器向服务器发出请求，确定运营商，然后校园网计费服务器修改出口路由的ACL表，针对某mac地址开放某一特定运营商的拨号服务器，然后调用Windows自带的拨号软件来进行拨号。而一般的Windows, Linux, Mac都自带pppoe拨号器，所以我们只需要能够仿照已知的数据包的格式发出正确的开放请求包就可以了，接下来用户可以自己进行拨号。

图(4.1)则体现这一过程，表(4-7)则是请求数据包的结构。

0x00~0x1D字节 用户名，0x00~0x09为用户名的ASCII码，其余用0填充，事实上后面的pppoe拨号过程中的用户名与这里完全无关，所以这里用户名可以随便填写

0x1E~0x21字节 客户端的ip地址，这里ip地址为172.19.95.179，所以相应的十六进制为0xac,0x13,0x5f,0xb3。

0x22~0x32字节 MAC地址的ASCII码，这里MAC地址为00:22:FA:94:B5:0E，其中0x3a代表 ‘:’

0x33~0x37字节 固定格式，其中0x36字节代表运营商的编号，其中联通为0x01，电信为0x02，移动为0x03，0x36字节代表的信息非常重要，因为不同的运营商拨号服务器是不一样的

0x38~0x3B字节 这四个字节类似于校验码，通过一定的规则计算出来，计算不正确的情况下，服务器不会响应这个数据包并开放服务器。这个校验码的计算是通过反汇编得出来的，见第4.2节。

4.2 校验码的计算

校验码的计算是通过OllyDbg动态反汇编结合WireShark抓包得出来的，并结合内存中相应数据的改变，设置相应的断点反向出来的，过程不再赘述。最终定位到校验码

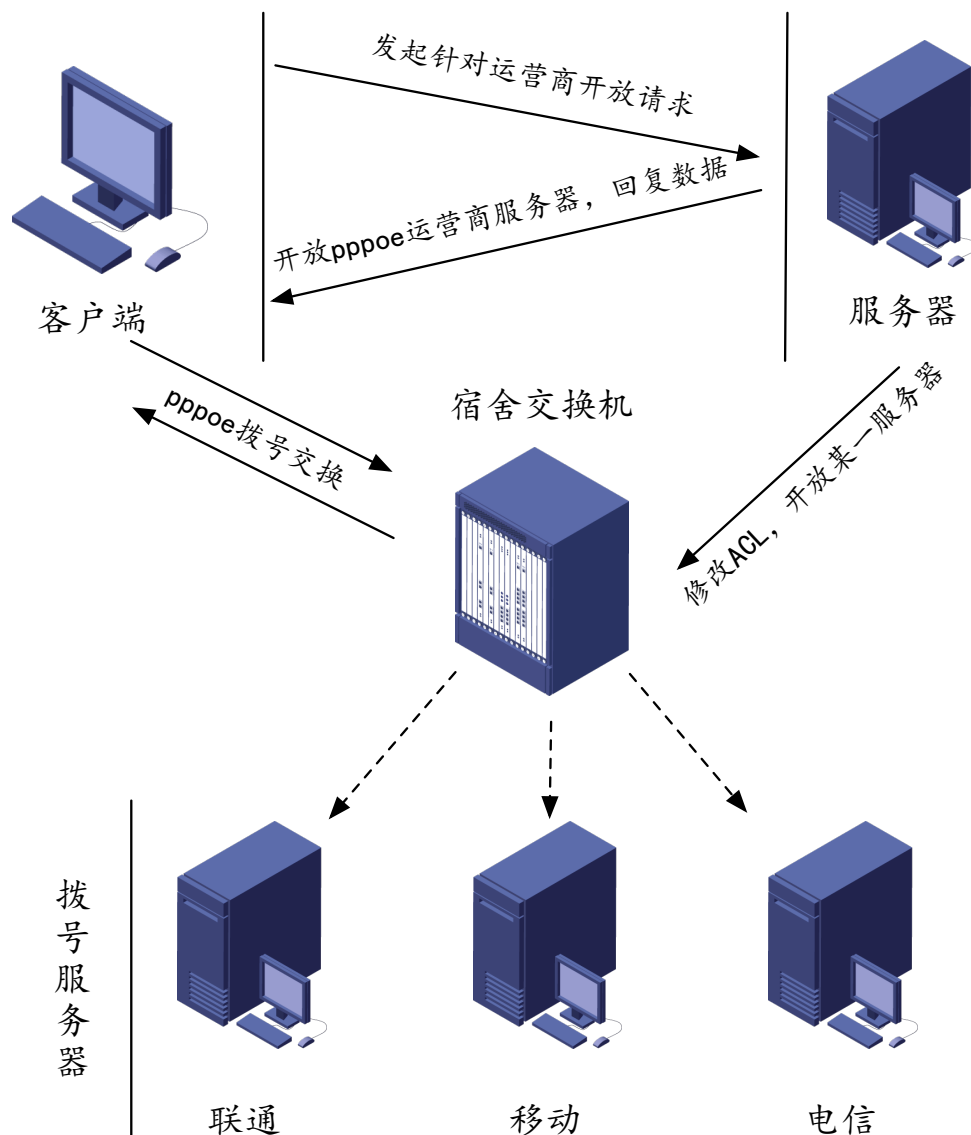


图 4.1 宽带握手过程

的计算算法位于0x00487479处，见图(4.2)，最终这一部分过程汇编代码见下，其中32行堆栈弹出，33行返回原过程，该子过程结束。该算法的资料极少，并且在代码的第6行第0x48746D处存在0x4E67C6A7这个常数，Google到是某一种特定的Hash算法的变异版本。

0048746B	MOV EAX,ESP	1
0048746D	MOV ECX,4E67C6A7	2
00487472	INC EDX	3
00487473	DEC EDX	4
00487474	TEST EDX,EDX	5
00487476	JL SHORT IPClient.004874A5	6
00487478	INC EDX	7
00487479	MOV ESI,ECX	8
0048747B	SHL ESI,5	9
0048747E	TEST ECX,ECX	10

表 4-7 宿舍宽带的的请求数据包

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	32	30	31	35	31	31	30	39	36	30	00	00	00	00	00	00
0x10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	ac	13
0x20	5f	b3	30	30	3a	32	32	3a	46	41	3a	39	34	3a	42	35
0x30	3a	30	45	00	00	00	01	00	8a	e4	e0	76				

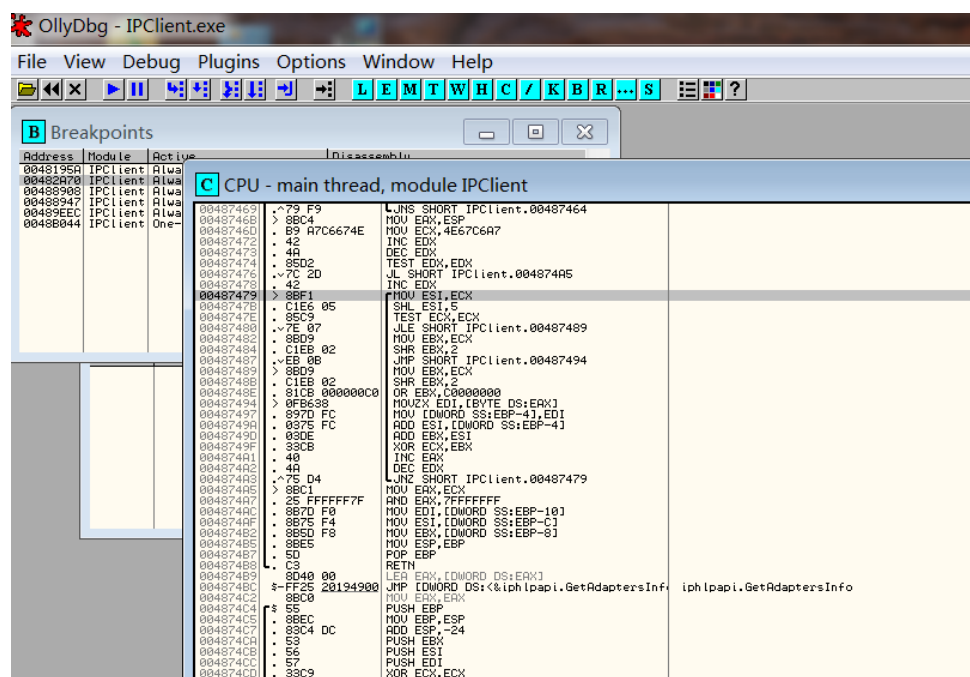


图 4.2 OllyDbg定位结果

00487480	JLE SHORT IPClient.00487489	11
00487482	MOV EBX, ECX	12
00487484	SHR EBX, 2	13
00487487	JMP SHORT IPClient.00487494	14
00487489	MOV EBX, ECX	15
0048748B	SHR EBX, 2	16
0048748E	OR EBX, C0000000	17
00487494	MOVZX EDI, [BYTE DS:EAX]	18
00487497	MOV [DWORD SS:EBP-4], EDI	19
0048749A	ADD ESI, [DWORD SS:EBP-4]	20
0048749D	ADD EBX, ESI	21
0048749F	XOR ECX, EBX	22
004874A1	INC EAX	23
004874A2	DEC EDX	24
004874A3	JNZ SHORT IPClient.00487479	25
004874A5	MOV EAX, ECX	26
004874A7	AND EAX, 7FFFFFFF	27
004874AC	MOV EDI, [DWORD SS:EBP-10]	28

004874AF MOV ESI , [DWORD SS:EBP-C]	29
004874B2 MOV EBX , [DWORD SS:EBP-8]	30
004874B5 MOV ESP , EBP	31
004874B7 POP EBP	32
004874B8 RETN	33

QT中的代码实现如下，其中最核心的校验码的计算部分则直接翻译汇编代码来完成，因此在代码的第7-10行定义了ECX,ESI,EBX,EAX代表四个寄存器，其中第4行用于获得本机的MAC地址。11-18行则预先定义了数据的0x00~0x37字节，23-26行则将MAC地址的ASCII码填充到数据中去，39-42行则将IP地址填充到数据当中。

```

int MainWindow::ispCon(int ispNum)
{
    CafesClient a;
    myMac=a.get_localmachine_mac(myIP);
    qDebug()<<myMac;
    int ispKey=0x4e67c6a7;
    int ECX;
    int ESI;
    int EBX;
    int EAX;
    unsigned char localInfo[]={0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0xac,0x10,0x40,0x12,0x30,0x30,0x3a,0x31,
                                0x46,0x3a,0x31,0x36,0x3a,0x32,0x32,0x3a,
                                0x42,0x38,0x3a,0x45,0x43,0x00,0x00,0x00,
                                0x02,0x00};

    int nInfo=sizeof(localInfo);
    int nMac=myMac.size();
    localInfo[nInfo-2]=(unsigned char)ispNum;
    qDebug()<<myIP;
    for (int i=0;i<nMac;i++)
    {
        localInfo[i+34]=(unsigned char)myMac[i].toLatin1();
    }
    QStringList ipList = myIP.split(".");

    if (ipList.size()!=4)
    {
        return -1;
    }
    int ip[4]={0};
    for (int i=0;i<4;i++)
    {
        ip[i]=ipList.at(i).toInt();
        qDebug()<<ip[i];
    }
}

```

```

    }
    for (int i=0;i<4;i++)
    {
        localInfo[i+30]=(unsigned char)ip[i];
    }
    /***** Calculating Key *****/
    ECX=ispKey;
    for (int i=0;i<nInfo;i++)
    {
        ESI=ECX;
        ESI=ESI<<5;
        if (ECX>0)
        {
            EBX=ECX;
            EBX=EBX>>2;
        }
        else
        {
            EBX=ECX;
            EBX=EBX>>2;
            EBX=EBX|(0xC0000000);
        }
        ESI=ESI+localInfo[i];
        EBX=EBX+ESI;
        ECX=ECX^EBX;
    }
    ECX=ECX&0x7FFFFFFF;
    QByteArray ispData;
    for (int i=0;i<nInfo;i++) ispData.append(localInfo[i]);
    for (int i=0;i<4;i++)
    {
        unsigned char keypart;
        keypart=(unsigned char)(ECX>>(i*8))&0x000000FF;
        ispData.append(keypart);
    }
    ispUdp->write(ispData);
    return 1;
}

```

44-64行则是直接将汇编翻成了C++语言，因为C++支持位操作，所以汇编中的与或非以及移位运算都可以直接用C++来表示，而jmp，test等跳转则翻成if语句。

因为最后最后的校验码是一个32bit数据，正好为4个字节，所以在代码的67-72行，通过移位以及与运算把每半个字节提出来，填充到数据包的末端当中去。73行则进行数据的发送。

5 测试结果以及总结



图 5.1 程序运行界面

最终程序界面见图(5.1)，可以看到在校园网连接成功之后，可以正确返回余额，并顺利上网。经过验证，该出校控制器可以做到Android, OS X, Windows, Linux全平台可用，也说明了QT在跨平台开发方面有着巨大的优势。

同时在python上的实现也可以顺利运行，见图(5.2)。学校没有出ipad上的控制器，

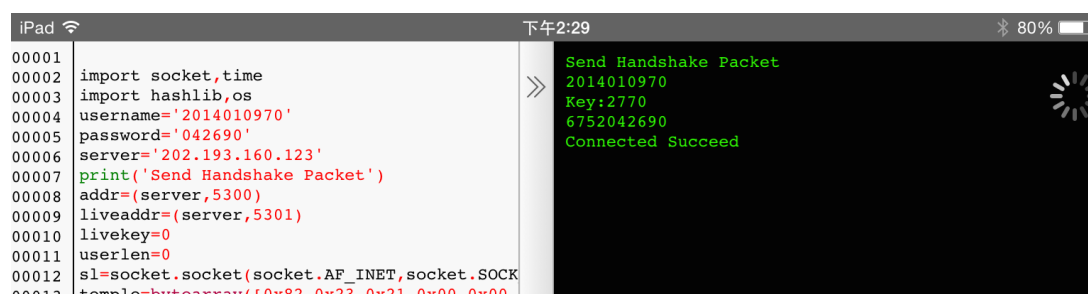
```
xuzhipengdeMacBook-Pro:Desktop xuzp$ python ipclient.py
IP Client for GXNU by Xu
.....
Sending Handshake Packet...
Receiving key packet...
Sending user information...
Receiving auth information...
Connected Succeed
□
```

图 5.2 Python版运行结果

导致即使接入无线网也不能上网局面，但是国外有人做出iPad上的Python解释器，所以Python也可以在ios系统上正常接入网络，如图(5.3)所示，但是由于ios不支持程序的后台运行，所以心跳包的发送受到影响，但是可以在接入后四五分钟内正常上网。

6 总结

原本做这个项目的最初目的只是为了能够在自己的Macbook上能够运行学校的出校控制器，但是并没有官方资料的可以提供，所以只能根据桂林电子科技大学相关同学



```
00001 import socket,time
00002 import hashlib,os
00003 username='2014010970'
00004 password='042690'
00005 server='202.193.160.123'
00006 print('Send Handshake Packet')
00007 addr=(server,5300)
00008 liveaddr=(server,5301)
00009 livekey=0
00010 userlen=0
00011 s=socket.socket(socket.AF_INET,socket.SOCK
00012 template=bytearray([0x02, 0x23, 0x21, 0x00, 0x00])
```

```
Send Handshake Packet
2014010970
Key:2770
6752042690
Connected Succeed
```

图 5.3 ipad上Python版运行结果

的工作成果上进行，而自己个人不单单用Windows，也会由于一些需要使用Linux和OS X系统，所以一个能够全平台运行的出校控制器也是必要的。

通过这一次实践，我又温习了汇编语言，并且亲自进行了一次反向工程的尝试，在调试代码的失败的挫折中锻炼了自己，使得自己获得了很大的提高。

附录

附录 A 密文摘要中参数的计算

```
#coding:utf-8
import hashlib
import os
stq=['1','2','3','4','5','6','7','8','9','0']
for i1 in stq:
    for i2 in stq:
        for i3 in stq:
            for i4 in stq:
                src = i1+i2+i3+i4+'042690'
                md51=hashlib.md5(src).hexdigest().upper()
                md52=md51[0:5]+'2014010970'
                md53=hashlib.md5(md52).hexdigest().upper()
                if md53[0:30]=='A6C81C0BB197CBA53EDCCBD122D85A':
                    print(8284-int(i1+i2+i3+i4))
```

附录 B 密文摘要计算部分反汇编

8049379:	8d 45 f8	lea	-0x8(%ebp),%eax	1
804937c:	03 85 bc ea ff ff	add	-0x1544(%ebp),%eax	2
8049382:	03 85 b8 ea ff ff	add	-0x1548(%ebp),%eax	3
8049388:	03 85 b4 ea ff ff	add	-0x154c(%ebp),%eax	4
804938e:	2d 19 09 00 00	sub	\$0x919,%eax	5
8049393:	8b 00	mov	(%eax),%eax	6
8049395:	2d 10 0d 00 00	sub	\$0xd10,%eax	7
804939a:	a3 ec c9 04 08	mov	%eax,0x804c9ec	8
804939f:	66 c7 85 c8 e9 ff ff	movw	\$0x2382,-0x1638(%ebp)	9
80493a6:	82 23			10
80493a8:	c6 85 ca e9 ff ff 21	movb	\$0x21,-0x1636(%ebp)	11
80493af:	c7 85 cc e9 ff ff 00	movl	\$0x0,-0x1634(%ebp)	12

附录 C Python版本源码

代码清单 1 IP Client 的Python版本源码

```
import socket,time
import hashlib,os
username='username'
password='password'
server='192.168.1.13'
print('IP_Client_for_GXNU_by_Xu')
print('.....')
```

```

print ('Sending_Handshake_Packet...')
addr=(server,5300)
liveaddr=(server,5301)
livekey=0
s1=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
temple=bytearray([0x82,0x23,0x21,0x00,0x00,0x00,
                    0x00,0x00,0x00,0x00,0x00,0x0e,0x00,0x00,
                    0x00,0x39,0x67,0x64,0x74,0x34,0x33,0x37,
                    0x34,0x35,0x77,0x72,0x77,0x71,0x72,0x1e,
                    0x00,0x00,0x00,0xFF,0x74,0x34,0x33,0x37,
                    0x35,0x42,0x38,0x32,0x35,0x37,0x44,0x44,
                    0x31,0x35,0x30,0x45,0xFF,0x44,0x37,0x36,
                    0x44,0x31,0x35,0x46,0x33,0x35,0x46,0x30,
                    0x44,0x11,0x00,0x00,0x00,0x31,0x31,0x3a,
                    0x32,0x32,0x3a,0x33,0x33,0x3a,0x34,0x34,
                    0x3a,0x35,0x35,0x3a,0x36,0x36,0x2d,0x1f,
                    0xd6,0x03,0xcc,0xf2,0x24,0x00,0x0a,0x00,
                    0x00,0x00,0x71,0x77,0x65,0x72,0x74,0x79,
                    0x75,0x69,0x6f,0x70])
livepack=bytearray(500)
def send_handshake():
    global livekey
    s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    s.connect(addr)
    pack_send=bytearray(300)
    pack_send[0]=0x82;pack_send[1]=0x23;pack_send[2]=0x1f
    pack_send[11]=len(username)
    userlen=len(username)
    #print(username)
    tail=bytearray([0x0b,0x00,0x00,0x00,0x21,0x40,0x23,0x24,0x25,0x25,0x5e
                    ,0x26,0x2a,0x28,0x29,0x07,0x00,0x00,0x00,0x71,0x77,0x65,0x72,0x74,0
                    x79,0x75,0x39,0x30,0x00,0x00,0x01,0x00,0x00,0x00,0x06,0x00,0x00,0
                    x00,0x41,0x53,0x44,0x46,0x47,0x48])
    for i in range(0,userlen):
        pack_send[15+i]=ord(username[i])-10;
    for i in range(0,44):
        pack_send[15+userlen+i]=tail[i];
    try:
        s.send(pack_send)
        datarec = s.recv(1024)
    except socket.error:
        print ('Network_error')
        quit()

```

```

datarec=bytearray ( datarec )
for i in range(0,300-3):
    if datarec[i]==0x82 and datarec[i+1]==0x23 and datarec[i+2]==0x20:
        print('Receiving_key_packet...')
        calckey=datarec[i+51+userlen-10]+(datarec[i+52+userlen-10]<<8)
        #print ( 'Key:%x' %calckey )
        calckey=calckey-3344;
        livekey=datarec[i+52+userlen-10]+(datarec[i+51+userlen-10]<<8)
        if livekey>0x3407:livekey=livekey-0x3407
        else:livekey=livekey-0x3408
        livekey=livekey&(0x0000ffff)
        src=str(calckey)+password
        #print(src)
        src=src.encode('utf-8')
        md51=hashlib.md5(src).hexdigest().upper()
        md52=md51[0:5]+username
        md52=md52.encode('utf-8')
        md53=hashlib.md5(md52).hexdigest().upper()
        md53=md53[0:30]
        md53=md53.encode('utf-8')
        md53=bytearray(md53)
        pass_pack=bytearray(300)
        for i in range(0,len(temple)):pass_pack[i]=temple[i]
        for i in range(0,30):pass_pack[i+33]=md53[i]
        s.send(pass_pack)
        print('Sending_user_information...')
        datarec = s.recv(1024)
        break
datarec=bytearray ( datarec )
for i in range(0,300):
    if datarec[i]==0x82 and datarec[i+1]==0x23 and datarec[i+2]==0x22:
        print('Receiving_auth_information...')
        if datarec[i+3]==0x00:
            print('Connected_Succeed')
            buildlive()
            sendlive()
        elif datarec[i+3]==0x63:
            print('Wrong_username_or_password')
        elif datarec[i+3]==0x20:
            print('Account_is_be_occupied')
        elif datarec[i+3]==0x14:
            print('No_money')
def buildlive():

```

```
global livekey
mid=bytearray([0xe4,0x3e,0x86,0x02,
               0x00,0x00,0x00,0x00,
               0x5c,0x8f,0xc2,0xf5,
               0xf0,0xa9,0xdf,0x40])
livepack[0]=0x82;livepack[1]=0x23;livepack[2]=0x1e;
livepack[3]=(livekey&0xff00)>>8;
# print((livekey&0xff00)>>8)
livepack[4]=(livekey&0x00ff)
for i in range(0,15):livepack[i+11]=mid[i]
pos=0
for i in range(0,len(username)):
    livepack[31+i]=ord(username[i])
    pos=31+i
spider=bytearray([0x09,0x00,0x00,0x00,0x53,0x70,0x69,0x64,0x65,0x72,0
                  x6d,0x61,0x6e])
for i in range(0,13):livepack[pos+i+1]=spider[i]
def sendlive():
    while True:
        time.sleep(5)
        sl.sendto(livepack, liveaddr)
#print('Live')

if __name__=="__main__":
    send_handshake()
```