Star Hacking Part2

wangyao@cs.hit.edu.cn
ipconfigme@gmail.com

接着前面的协议分析，这一部分进行代码的分析，代码分析需要网络编程的知识、Unix 环境编程的知识，以及一点密码学知识。

锐捷代码实战篇

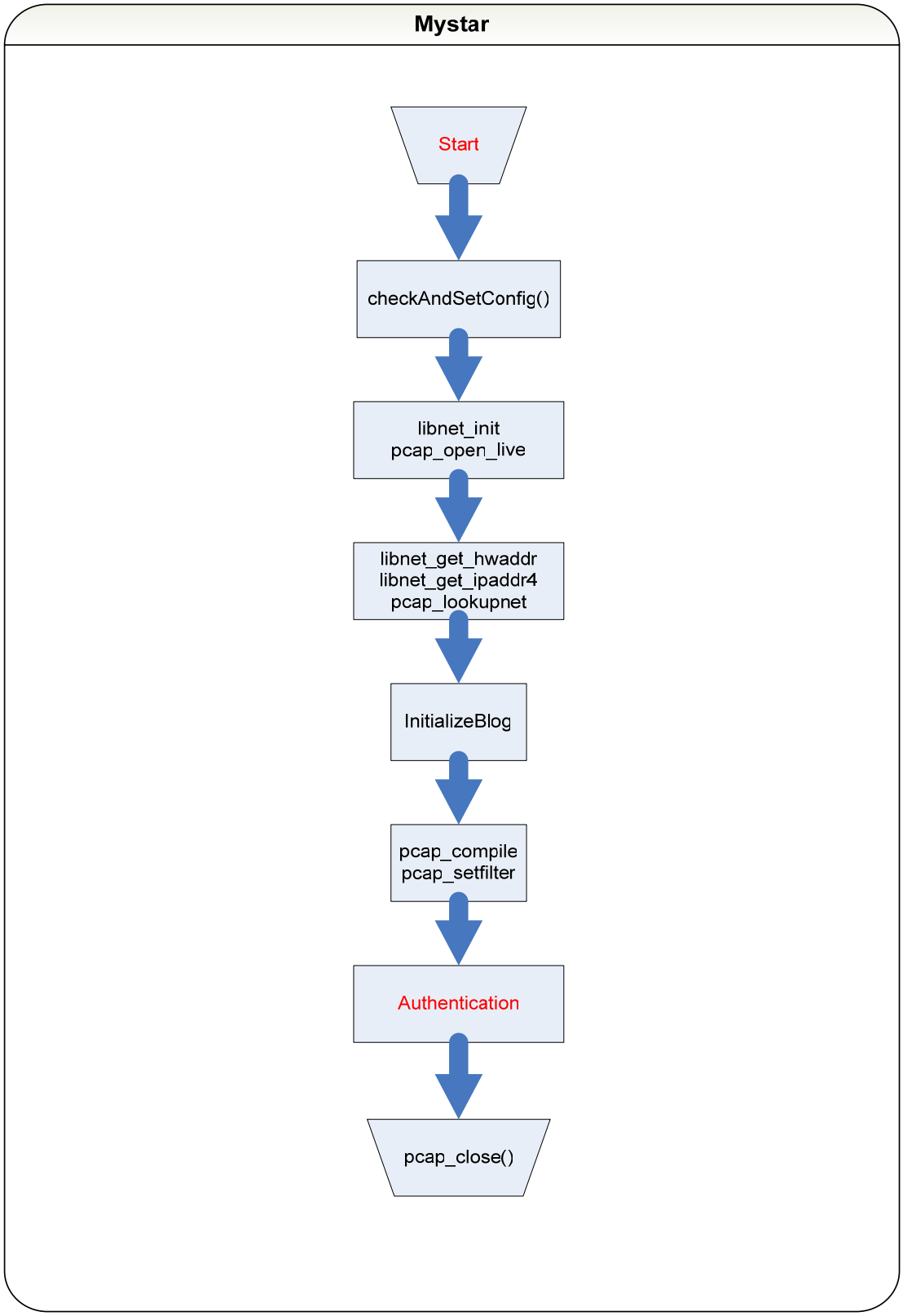
关于代码的实战，我想结合 mystar 源码来讲。这样的话，如果协议发生了变化，可以通过捕包分析，再进行修改。

看代码要首先由一个宏观的概念，由了解宏观，就得看 Makefile。

```
# This file is generated manually by NetXRay@byhh.
# It isn't very elagent. Don't mock me :)

CC=gcc
Flags=-O2

all:   mystar

mystar: mystar.o md5.o myerr.o blog.o sendpacket.o conn_monitor.o
	$(CC) $(Flags) -o $@    $^ -static -lnet -lpcap -lpthread

conn_monitor.o: conn_monitor.c conn_monitor.h
	$(CC) $(Flags) -o $@ -c $<

md5.o:      md5.c md5.h
	$(CC) $(Flags) -o $@ -c $<

myerr.o: myerr.c myerr.h
	$(CC) $(Flags) -o $@ -c $<

blog.o:    blog.c blog.h    myerr.h
	$(CC) $(Flags) -o $@ -c $<

sendpacket.o: sendpacket.c sendpacket.h md5.h global.h blog.h
	$(CC) $(Flags) -o $@ -c $<

mystar.o: mystar.c mystar.h sendpacket.h myerr.h blog.h global.h
	$(CC) $(Flags) -o $@ -c $<

clean:
	rm -f *.o mystar
rebuild:
	make clean all
```

从上面的 Makefile，我们可以看出 mystar 大致分几个模块：md5、blog、sendpacket、conn_monitor、myerr、mystar 模块，这我们主要讲 mystar、blog 和 sendpacket 模块。

mystar 流程图：



Blog 部分

mystar.c 基本上就是按照那个流程走的。这里需要对几个地方进行说明。
首先是 InitializeBlog(m_ip,m_netmask,m_netgate,m_dns1);
InitializeBlog 函数定义在 blog.c 中可以进行查看。

```
//configure the 4 parameters Blog() and FillNetParameter() need.
//Compute the circleCheck[2]
void InitializeBlog(const unsigned char *m_ip, const unsigned char
*m_netmask,const unsigned char *m_netgate, const unsigned char *m_dns1)
{
    memcpy(m_IP,m_ip,4);
    memcpy(m_NetMask,m_netmask,4);
    memcpy(m_NetGate,m_netgate,4);
    memcpy(m_DNS1,m_dns1,4);

    Blog(); // Compute the circleCheck[2]

    blogIsInitialized=1;
}
```

InitializeBlog 函数的作用主要就是设置参数，并计算 circleCheck[2]，这是两个 Magic Number。

下面我们就来看一下这个 Blog 算法：

```
//那帮家伙们，单靠这个算法就想区别实达客户端和非实达客户端-_- !!
//The only use of function Blog() is to work out circleCheck[2],
//with and only with the help of 4 parameters----m_IP, m_NetMask,
m_NetGate, m_DNS1
static void Blog(void)
{
    static unsigned char Table[]={
    0x00,0x00,0x21,0x10,0x42,0x20,0x63,0x30,0x84,0x40,0xA5,0x50,0xC6,0x
60,0xE7,0x70,
    0x08,0x81,0x29,0x91,0x4A,0xA1,0x6B,0xB1,0x8C,0xC1,0xAD,0xD1,0xCE,
0xE1,0xEF,0xF1,
    0x31,0x12,0x10,0x02,0x73,0x32,0x52,0x22,0xB5,0x52,0x94,0x42,0xF7,0x
72,0xD6,0x62,
    0x39,0x93,0x18,0x83,0x7B,0xB3,0x5A,0xA3,0xBD,0xD3,0x9C,0xC3,0xFF,0
xF3,0xDE,0xE3,
    0x62,0x24,0x43,0x34,0x20,0x04,0x01,0x14,0xE6,0x64,0xC7,0x74,0xA4,0x
44,0x85,0x54,
    0x6A,0xA5,0x4B,0xB5,0x28,0x85,0x09,0x95,0xEE,0xE5,0xCF,0xF5,0xAC,0
xC5,0x8D,0xD5,
    0x53,0x36,0x72,0x26,0x11,0x16,0x30,0x06,0xD7,0x76,0xF6,0x66,0x95,0x
56,0xB4,0x46,
```

```
    0x5B,0xB7,0x7A,0xA7,0x19,0x97,0x38,0x87,0xDF,0xF7,0xFE,0xE7,0x9D,0
xD7,0xBC,0xC7,
    0xC4,0x48,0xE5,0x58,0x86,0x68,0xA7,0x78,0x40,0x08,0x61,0x18,0x02,0x
28,0x23,0x38,
    0xCC,0xC9,0xED,0xD9,0x8E,0xE9,0xAF,0xF9,0x48,0x89,0x69,0x99,0x0A,0
xA9,0x2B,0xB9,
    0xF5,0x5A,0xD4,0x4A,0xB7,0x7A,0x96,0x6A,0x71,0x1A,0x50,0x0A,0x33,0
x3A,0x12,0x2A,
    0xFD,0xDB,0xDC,0xCB,0xBF,0xFB,0x9E,0xEB,0x79,0x9B,0x58,0x8B,0x3B,
0xBB,0x1A,0xAB,
    0xA6,0x6C,0x87,0x7C,0xE4,0x4C,0xC5,0x5C,0x22,0x2C,0x03,0x3C,0x60,0
x0C,0x41,0x1C,
    0xAE,0xED,0x8F,0xFD,0xEC,0xCD,0xCD,0xDD,0x2A,0xAD,0x0B,0xBD,0x68
,0x8D,0x49,0x9D,
    0x97,0x7E,0xB6,0x6E,0xD5,0x5E,0xF4,0x4E,0x13,0x3E,0x32,0x2E,0x51,0
x1E,0x70,0x0E,
    0x9F,0xFF,0xBE,0xEF,0xDD,0xDF,0xFC,0xCF,0x1B,0xBF,0x3A,0xAF,0x59,0
x9F,0x78,0x8F,
    0x88,0x91,0xA9,0x81,0xCA,0xB1,0xEB,0xA1,0x0C,0xD1,0x2D,0xC1,0x4E,
0xF1,0x6F,0xE1,
    0x80,0x10,0xA1,0x00,0xC2,0x30,0xE3,0x20,0x04,0x50,0x25,0x40,0x46,0x
70,0x67,0x60,
    0xB9,0x83,0x98,0x93,0xFB,0xA3,0xDA,0xB3,0x3D,0xC3,0x1C,0xD3,0x7F,
0xE3,0x5E,0xF3,
    0xB1,0x02,0x90,0x12,0xF3,0x22,0xD2,0x32,0x35,0x42,0x14,0x52,0x77,0x
62,0x56,0x72,
    0xEA,0xB5,0xCB,0xA5,0xA8,0x95,0x89,0x85,0x6E,0xF5,0x4F,0xE5,0x2C,0
xD5,0x0D,0xC5,
    0xE2,0x34,0xC3,0x24,0xA0,0x14,0x81,0x04,0x66,0x74,0x47,0x64,0x24,0x
54,0x05,0x44,
    0xDB,0xA7,0xFA,0xB7,0x99,0x87,0xB8,0x97,0x5F,0xE7,0x7E,0xF7,0x1D,0
xC7,0x3C,0xD7,
    0xD3,0x26,0xF2,0x36,0x91,0x06,0xB0,0x16,0x57,0x66,0x76,0x76,0x15,0x
46,0x34,0x56,
    0x4C,0xD9,0x6D,0xC9,0x0E,0xF9,0x2F,0xE9,0xC8,0x99,0xE9,0x89,0x8A,0
xB9,0xAB,0xA9,
    0x44,0x58,0x65,0x48,0x06,0x78,0x27,0x68,0xC0,0x18,0xE1,0x08,0x82,0x
38,0xA3,0x28,
    0x7D,0xCB,0x5C,0xDB,0x3F,0xEB,0x1E,0xFB,0xF9,0x8B,0xD8,0x9B,0xBB,
0xAB,0x9A,0xBB,
    0x75,0x4A,0x54,0x5A,0x37,0x6A,0x16,0x7A,0xF1,0x0A,0xD0,0x1A,0xB3,0
x2A,0x92,0x3A,
    0x2E,0xFD,0x0F,0xED,0x6C,0xDD,0x4D,0xCD,0xAA,0xBD,0x8B,0xAD,0xE8
,0x9D,0xC9,0x8D,
```

```
    0x26,0x7C,0x07,0x6C,0x64,0x5C,0x45,0x4C,0xA2,0x3C,0x83,0x2C,0xE0,0
x1C,0xC1,0x0C,
    0x1F,0xEF,0x3E,0xFF,0x5D,0xCF,0x7C,0xDF,0x9B,0xAF,0xBA,0xBF,0xD9,0
x8F,0xF8,0x9F,
    0x17,0x6E,0x36,0x7E,0x55,0x4E,0x74,0x5E,0x93,0x2E,0xB2,0x3E,0xD1,0
x0E,0xF0,0x1E};

        static unsigned char sCircleBase[0x15]={
    0x00,0x00,0x13,0x11,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
            int iCircle=0x15;
            int i,ax=0,bx=0,dx=0;

            sCircleBase[0x05] = m_IP[0];
            sCircleBase[0x06] = m_IP[1];
            sCircleBase[0x07] = m_IP[2];
            sCircleBase[0x08] = m_IP[3];
            sCircleBase[0x09] = m_NetMask[0];
            sCircleBase[0x0a] = m_NetMask[1];
            sCircleBase[0x0b] = m_NetMask[2];
            sCircleBase[0x0c] = m_NetMask[3];
            sCircleBase[0x0d] = m_NetGate[0];
            sCircleBase[0x0e] = m_NetGate[1];
            sCircleBase[0x0f] = m_NetGate[2];
            sCircleBase[0x10] = m_NetGate[3];
            sCircleBase[0x11] = m_DNS1[0];
            sCircleBase[0x12] = m_DNS1[1];
            sCircleBase[0x13] = m_DNS1[2];
            sCircleBase[0x14] = m_DNS1[3];

            for ( i=0 ; i<iCircle ; i++ )
            {
                dx = ax;
                bx = 0;
                bx = (bx&0xff00) | sCircleBase[i]; // add "( )" by cdx
                dx &= 0xffff;
                dx >>= 8;
                dx ^= bx;
                bx = 0;
                bx &= 0x00ff;
                bx |= (ax&0xff)<<8;

                ax = Table[dx*2] | Table[dx*2+1]<<8;
                ax ^= bx;
```

```
        }
        circleCheck[0] = (unsigned char) ((ax&0xff00)>>8);
        circleCheck[1] = (unsigned char) (ax&0x00ff);
}
```

上面的 Blog 算法是整个认证中最为关键的地方了，单单靠抓取数据包是推不出来的，因此需要进行反汇编来得到上面的 Table。如果这个算法发生了变化，破解认证将变得有些麻烦。

已经看到 blog.c 了，就顺便看一下其他的函数吧！

FillNetParamater 函数和 Alog 函数。其中 Alog 函数是一个基础，用来变换参数的，FillNetParamater 调用 Ablog 函数，将设置好的网络参数进行混乱化，并用来设置 Star 专有认证数据包。这样直接观察从网络上捕到的数据包，将看不出什么实质的信息。

```
//Fill in some additional information   Ruijie Corp. required.
//You should call InitializeBlog() before calling this function.
void FillNetParamater(unsigned char ForFill[])
{
    if(blogIsInitialized==0) err_quit("Blog algorithm not initialied yet \n");

    ForFill[ 0] = Alog( m_IP[0] );
    ForFill[ 1] = Alog( m_IP[1] );
    ForFill[ 2] = Alog( m_IP[2] );
    ForFill[ 3] = Alog( m_IP[3] );
    ForFill[ 4] = Alog( m_NetMask[0] );
    ForFill[ 5] = Alog( m_NetMask[1] );
    ForFill[ 6] = Alog( m_NetMask[2] );
    ForFill[ 7] = Alog( m_NetMask[3] );
    ForFill[ 8] = Alog( m_NetGate[0] );
    ForFill[ 9] = Alog( m_NetGate[1] );
    ForFill[10] = Alog( m_NetGate[2] );
    ForFill[11] = Alog( m_NetGate[3] );
    ForFill[12] = Alog( m_DNS1[0] );
    ForFill[13] = Alog( m_DNS1[1] );
    ForFill[14] = Alog( m_DNS1[2] );
    ForFill[15] = Alog( m_DNS1[3] );
    ForFill[16] = Alog( circleCheck[0] );
    ForFill[17] = Alog( circleCheck[1] );
}

//A transformation of one-byte-for-one-byte
unsigned char Alog(unsigned char BForAlog)
{
    int a=0,b=0,c=0,d=0,iRt;

    a=BForAlog;
```

```
    c=a;
    d=a;
    c&=0x40;
    b=a;
    d>>=2;
    c|=d;
    d=a;
    c>>=2;
    d&=0x20;
    c|=d;
    d=a;
    c>>=2;
    d&=0x10;
    c|=d;

    d=a;
    d&=2;
    b<<=2;
    d|=b;
    b=a;
    d<<=2;
    b&=4;
    a&=8;
    d|=b;
    d<<=2;
    d|=a;

    c>>=1;
    d<<=1;
    c|=d;
    iRt=(~c)&0xff;
    return iRt;
}
```

Ablog算法也是需要反汇编才能够得出来的，如果这个算法发生了变化，也会使破解变得比较麻烦。

Sendpacket部分

我们再回到mystar.c中接着我们前面的地方讲下去。

```
//set the filter. Here I'm sure filter_buf is big enough.
snprintf(filter_buf,sizeof(filter_buf),FILTER_STR,
m_localMAC[0],m_localMAC[1],m_localMAC[2],m_localMAC[3],m_localMAC[4]
,m_localMAC[5]);
if(pcap_compile(p, &filter_code,filter_buf, 0, p_netmask)==-1)
{ err_msg("pcap_compile(): %s", pcap_geterr(p));    goto err2; }
```

```
if(pcap_setfilter(p, &filter_code)==-1)
{ err_msg("pcap_setfilter(): %s", pcap_geterr(p)); goto err2; }
pcap_freecode(&filter_code); // avoid   memory-leak
```

上面是使用libpcap来抓取EAPOL的数据包，其中的FILTER_STR在mystar.h中定义了。

```
#define FILTER_STR "ether[12:2]=0x888e and ether dst
%02x:%02x:%02x:%02x:%02x:%02x"
```

要理解上面的这个定义，需要了解BPF过滤规则。上面定义的字符串描述的是从12字节开始共2个字节是0x888E，并且目的MAC地址是%02x:%02x:%02x:%02x:%02x:%02x的数据包。

下面是mystar.c中关于发送数据包部分，根据阶段来发送相应的数据包，与服务器进行交互。

这一部分有一些关于信号的处理的知识，对一些不熟悉Linux环境编程的人理解起来有些困难。不过没有关系，这些都是一些很基本的，稍微讲一下就能够明白的。

信号处理：

```
(void)signal(SIGINT,sig_intr);   //We can exit with Ctrl+C
(void)sigfillset(&sigset_full);
(void)sigprocmask(SIG_BLOCK,&sigset_full,NULL); //block all signals.
```

上面是一些信号的设置。

signal(SIGINT,sig_intr)是在接收到SIGINT信号后执行sig_intr函数。

sigfillset(&sigset_full)是设置信号集sigset_full，将所有的信号都添加进信号集sigset_full中。

sigprocmask(SIG_BLOCK,&sigset_full,NULL)是阻塞信号集sigset_full中的信号，也就是说阻塞所有的信号。如果希望保存原来的信号的话，可以设置第三个参数，传进一个sigset_t类型的数据结构。

关于信号的处理，简单的了解可以查看《深入理解计算机系统》8.5节；或者是《Unix环境高级编程》的第10章。

pselect：

```
(void)sigfillset(&sigset_full);
(void)sigdelset(&sigset_full,SIGINT);
FD_ZERO(&read_set);   FD_SET(p_fd, &read_set);
timeout.tv_sec =1;     timeout.tv_nsec =0; // 1 second

//wait with all signals(except SIGINT) blocked.
switch ( pselect(p_fd+1,&read_set,NULL,NULL,&timeout,&sigset_full) )
{
case -1: //Normally, this case should not happen since sig_intr() never returns!
    goto err2;
case 0:   //timed out
    switch(m_state)
    {
```

```
    case 0:
        if(++packetCount_SentFindServer>3)
        { puts("Restarting authenticaton!"); goto beginAuthentication; }
        (void)SendFindServerPacket(l);
        continue; //jump to next loop of while(1) to receive next packet
    case 1:
        if(++packetCount_SentName>3)
        { puts("Restarting authenticaton!"); goto beginAuthentication; }
        (void)SendNamePacket(l, pkt_data);
        continue;
    case 2:
        if(++packetCount_SentPassword>3)
        { puts("Restarting authenticaton!"); goto beginAuthentication; }
        (void)SendPasswordPacket(l, pkt_data);
        continue;
    default:
        goto err2;
    }
}
//Here return value of pselect must be 1
.....
```

FD_ZERO清空文件描述符集，FD_SET添加指定的文件描述符。

pselect的定义：

```
int    pselect(int    n,    fd_set    *readfds,    fd_set    *writefds,    fd_set
        *exceptfds, const struct timespec *timeout, const sigset_t *sigmask);
```

pselect跟select的使用基本相同，主要是用于多路IO复用的情况。先简单的了解一下select函数。select的作用就是使主程序阻塞，等待read_set文件描述符集中任何一个文件描述符准备好读、写或者是超时之后，立即返回，并进行处理。

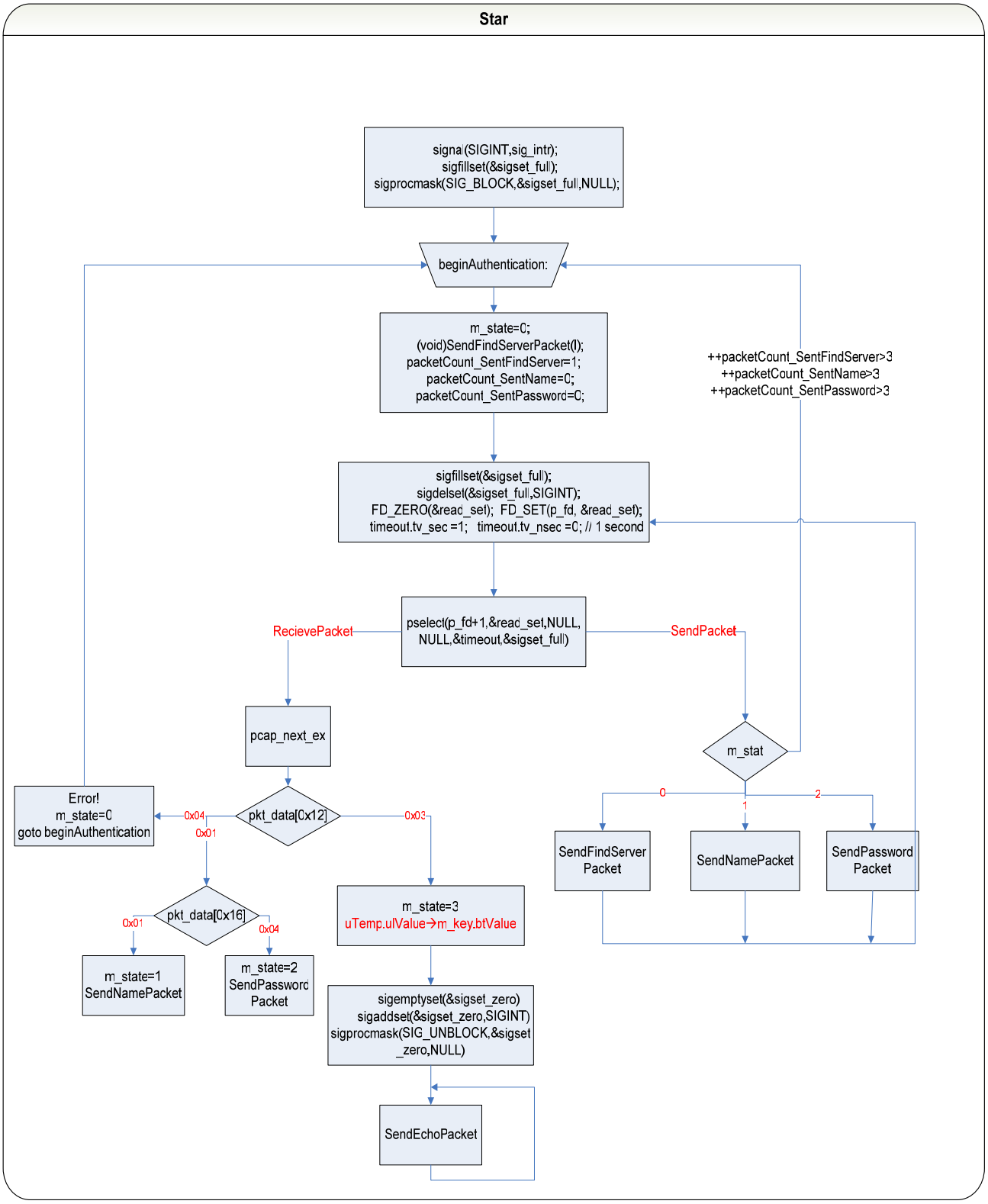
select的返回值有如下情况：

1. 正常情况下返回就绪的文件描述符个数；
2. 经过了timeout时长后仍无设备准备好，返回值为0；
3. 如果select被某个信号中断，它将返回-1并设置errno为EINTR；
4. 如果出错，返回-1并设置相应的errno。

pselect只是select的改进，避免了一种race conditions。例如一个信号的test紧接着一个select调用，那么如果信号正好在select和test之间发生，select将会不确定性的暂停。使用pselect就可以解决这个问题。

pselect比select函数多了一个参数，就是第六个参数const sigset_t *sigmask，设置阻塞的信号，pselect函数执行时，首先阻塞设置的sigmask，然后再运行pselect函数。这样的话，就不会出现前面select可能出现的race conditions了，因为我们已经将那些信号阻塞了，当然我们可以通过设置sigmask，使得我们的程序能够处理我们指定的信号。比如可以通过signal函数设置当用户按下CTRL+C（即INTR信号）时中断并退出。

关于select的问题和pselect的问题，详细介绍可以查看《深入理解计算机系统》13.2节，《Unix环境高级编程》12.5节，以及《Unix网络编程 卷I》6.9节。



Sendpacket

Sendpacket是在Star专有数据包的基础之上构建数据包，然后通过libnet发送出去。根据不同的认证阶段，来填充Star专有数据包中的一些部分。

SendFindServerPacket

```
int SendFindServerPacket(libnet_t *l)
```

```
{
    static uint8_t broadPackage[0x3E8] = {          //广播包，用于寻找服务器
        //0x00-0x11
        0x00,0x00,0x00,0x00,0x00,0x00,// Destination MAC
        0x00,0x00,0x00,0x00,0x00,0x00,// Source MAC
        0x88,0x8E,// Ethertype = 0x888E (8021X)
        0x01,// Version = 1
        0x01,// Packet Type   0x00 ;0x01,EAPOL-Start ;0x02 ;0x03 ;0x04
        0x00,0x00,// Packet Body Length
    };

    uint8_t StandardAddr[] = {0x01,0x80,0xC2,0x00,0x00,0x03}; //标准组播
地址
    uint8_t StarAddr[]      = {0x01,0xD0,0xF8,0x00,0x00,0x03}; //实达私有
组播地址

    extern uint8_t   m_localMAC[6];
    extern int      m_authenticationMode;

    if (m_authenticationMode==1)
        memcpy(broadPackage,StarAddr,6);
    else
        memcpy(broadPackage,StandardAddr,6);
    memcpy( broadPackage+6, m_localMAC, 6 );    //填充MAC地址

    FillNetParamater( &ackShida[0x05] );
    memcpy( ackShida+0x82, m_localMAC, 6 );
    memcpy( broadPackage+0x17,ackShida,0x90);

    fputs(">> Searching for server...\n",stdout);

    return (libnet_write_link(l,broadPackage, 0x3E8)==0x3E8)?0:-1;
}
```

SendNamePacket

```
int SendNamePacket(libnet_t *l, const u_char *pkt_data)
{
    static uint8_t ackPackage[0x3E8] = {          //应答包，包括用户名和MD5

    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x88,0x
8E,0x01,0x00,

    0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00
    };
```

```
    extern char *m_name;
    extern uint8_t   m_destMAC[6];
    extern uint8_t   m_localMAC[6];
    int nameLen;

    nameLen=strlen(m_name);
    memcpy(ackPackage,m_destMAC,6);   //将目的MAC地址填入组织回复的
包
    memcpy(ackPackage+6,m_localMAC,6);   //将本机MAC地址填入组织回复
的包
    ackPackage[0x12]=0x02;                  //code,2代表应答
    ackPackage[0x13]=pkt_data[0x13];   //id, HERE as if it's alway 1 from
ShiDa ??
    *(short *)(ackPackage+0x10) = htons((short)(5+nameLen));//len
    *(short *)(ackPackage+0x14) = *(short *)(ackPackage+0x10);//len
    memcpy(ackPackage+0x17,m_name,nameLen); //填入用户名

    FillNetParamater( &ackShida[0x05] );
    memcpy( ackShida+0x82, m_localMAC, 6 );
    memcpy(ackPackage+0x17+nameLen,ackShida,0x90);

    fputs(">> Sending user name...\n",stdout);

    return (libnet_write_link(l,ackPackage, 0x3E8)==0x3E8)?0:-1;
}
```

前面两个sendpacket都没有什么特别需要注意的地方。接下来，我们看一下SendPasswordPacket和SendEchoPacket。

SendPasswordPacket

```
int SendPasswordPacket(libnet_t *l,const u_char *pkt_data)
{
    static uint8_t ackPackage[0x3E8] = {          //应答包，包括用户名和MD5

    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x88,0x
8E,0x01,0x00,
    0x00,0x00,0x02,0x01,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0xFF,
        };

        unsigned char    md5Data[256]; //密码,md5 buffer
        unsigned char   *md5Dig;          //result of md5 sum
        int                md5Len=0;

        extern char *m_name;
```

```
        extern char *m_password;
        extern uint8_t   m_destMAC[6];
        extern uint8_t   m_localMAC[6];
        int nameLen,passwordLen;

        nameLen=strlen(m_name); passwordLen=strlen(m_password);

        memcpy(ackPackage,m_destMAC,6);
        memcpy(ackPackage+6,m_localMAC,6); //将本机MAC地址填入组织回复的包

        ackPackage[0x12] = 0x02;                    //code,2代表应答
        ackPackage[0x13]=pkt_data[0x13];            //id
        *(ackPackage+0x16) = *(pkt_data+0x16);   //type，即应答方式,HERE should alway be 4

        *(short *)(ackPackage+0x10) = htons((short)( 22+nameLen)); //len
        *(short *)(ackPackage+0x14) = *(short *)( ackPackage+0x10 );

        md5Data[md5Len++] = ackPackage[0x13];//ID
        memcpy(md5Data+md5Len,m_password,passwordLen);
md5Len+=passwordLen; //密码
        memcpy(md5Data+md5Len,pkt_data+0x18,pkt_data[0x17]);
md5Len+=pkt_data[0x17]; //密匙
        md5Dig =ComputeHash( md5Data, md5Len);

        ackPackage[0x17]=16;                    //length of md5sum is always 16.
        memcpy(ackPackage+0x18,md5Dig,16);

        memcpy(ackPackage+0x28,m_name,nameLen);

        FillNetParamater( &ackShida[0x05] );
        memcpy( ackShida+0x82, m_localMAC, 6 );
        memcpy(ackPackage+0x28+nameLen,ackShida,0x90);

        fputs(">> Sending password... \n",stdout);
        return (libnet_write_link(l,ackPackage, 0x3E8)==0x3E8)?0:-1;
}
```

SendPasswordPacket数据包格式如下：

```
static byte ackPackage[0x3E8] = { //应答包，包括用户名和MD5
    //0x00-0x11
    0x00,0x00,0x00,0x00,0x00,0x00, //Destination MAC
    0x00,0x00,0x00,0x00,0x00,0x00, //Source MAC
```

```
    0x88,0x8E,                  // Ethertype = 0x888E (8021X)
    0x01,                       // Version = 1
    0x00,                       // Packet Type
0x00 ;0x01,EAPOL-Start ;0x02 ;0x03 ;0x04
    0x00,0x00,                  // Packet Body Length
    //0x12
    0x02                        //表示应答
    //0x13
    0x00                        //ID(根据前面的服务器响应进行设置)
    //0x14-0x15
    0x00,0x00                   //Packet Body Length(同上面的x10-0x11)
    //0x16
    0x04                        //Type(MD5-Challenge)
    //0x17
    0x00            //Value-size(一般为16)
    //0x18-(0x18+Value-size-1)
    散列序列
注：散列序列的形成:     Hash(ID || Password || ReciveKey)
                其中ID、ReciveKey均是通过上一次会话从服务器获得的
    //（x18+Value-size）-(0x18+Value-size + NameLen-1)
    用户名
    //(0x18+Value-size + NameLen)-
    同前面FindServer的数据包;那个随机的字符串是一样的
    RuijieExtra[144];
}
```

```
int SendEchoPacket(libnet_t *l,const u_char *pkt_data)
{
    static uint8_t echoPackage[] ={     //echo包，用于每n秒钟激活一次

    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x88,0x
8E,0x01,0xBF,
    0x00,0x1E,0xFF,0xFF,0x37,0x77,0x7F,0x9F,0xF7,0xFF,0x00,0x00,0xFF,0x
FF,0x37,0x77,
    0x7F,0x9F,0xF7,0xFF,0x00,0x00,0xFF,0xFF,0x37,0x77,0x7F,0x3F,0xFF
    };
        ULONG_BYTEARRAY uCrypt1,uCrypt2,uCrypt1_After,uCrypt2_After;
        extern ULONG_BYTEARRAY    m_serialNo;
        extern ULONG_BYTEARRAY    m_key;
        extern uint8_t    m_destMAC[6];
        extern uint8_t    m_localMAC[6];

        m_serialNo.ulValue++;
        //m_serialNo is initialized at the beginning of main() of mystar.c, and
```

```
        //m_key is initialized in mystar.c when the 1st Authentication-Success
packet is received.

        //可以跟《Star Hacking Part1》中协议分析部分结合着看。
        uCrypt1.ulValue = m_key.ulValue + m_serialNo.ulValue;
        uCrypt2.ulValue = m_serialNo.ulValue;

        memcpy( echoPackage, m_destMAC, 6 );
        memcpy( echoPackage+6, m_localMAC, 6 );

        uCrypt1_After.ulValue = htonl( uCrypt1.ulValue );
        uCrypt2_After.ulValue = htonl( uCrypt2.ulValue );

        echoPackage[0x18] = Alog(uCrypt1_After.btValue[0]);
        echoPackage[0x19] = Alog(uCrypt1_After.btValue[1]);
        echoPackage[0x1a] = Alog(uCrypt1_After.btValue[2]);
        echoPackage[0x1b] = Alog(uCrypt1_After.btValue[3]);
        echoPackage[0x22] = Alog(uCrypt2_After.btValue[0]);
        echoPackage[0x23] = Alog(uCrypt2_After.btValue[1]);
        echoPackage[0x24] = Alog(uCrypt2_After.btValue[2]);
        echoPackage[0x25] = Alog(uCrypt2_After.btValue[3]);

        return (libnet_write_link(l,echoPackage, 0x2d)==0x2d)?0:-1;
}
```

注意mystar.c中的m_key的生成。

```
//Get A group Key number for echo packets(when the 1st
Authentication-Success packet is received.)
offset=ntohs( *((u_int16_t*)(pkt_data+0x10)) );
uTemp.ulValue = *((u_int32_t *)(pkt_data+(0x11+offset)-0x08));
m_key.btValue[0] = Alog(uTemp.btValue[3]);
m_key.btValue[1] = Alog(uTemp.btValue[2]);
m_key.btValue[2] = Alog(uTemp.btValue[1]);
m_key.btValue[3] = Alog(uTemp.btValue[0]);
```

另外，MD5模块和智能重连模块不是Star Hacking的重点，我们就不再描述了。感兴趣的话，可以自己阅读源代码。

代码大致浏览了一遍，最后需要留意几点，方便以后协议发生变化后改：

1、Identify需要在服务器端的返回数据包中取，并设置在以后的数据包中
2、第一次认证成功后，存取m_key，便于后面的echopacket的构造
3、留意认证过程中每一个阶段的数据包的长度，以后可能会增加或者是减少一些字段

OK! Every boy ,Enjoy Yourself!

参考资料：

1、《Unix环境高级编程》　W.Richard Stevents编著 机械工业出版社
2、《Unix网络编程 卷1》　W.Richard Stevents编著 机械工业出版社
3、《深入理解计算机系统》 Randal E.Bryant & David O'Hallaron编著　中国电力出版社
4、《网络安全开发包详解》 刘涛编著 电子工业出版社
5、Mystar源代码
6、Mento Supplicant源代码