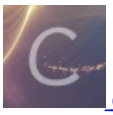
攻防世界Reverse进阶区-srm-50-writeup

原创

y4ung 于 2020-09-22 23:19:31 发布 330 收藏 1

分类专栏： ctf 文章标签： ctf

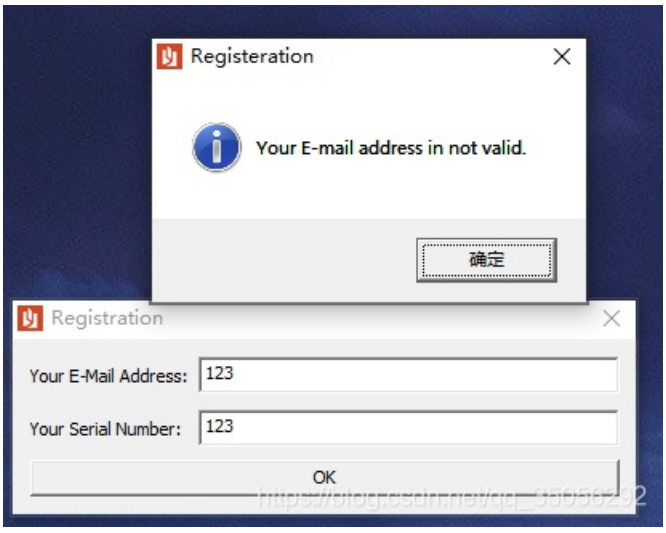


1. 介绍

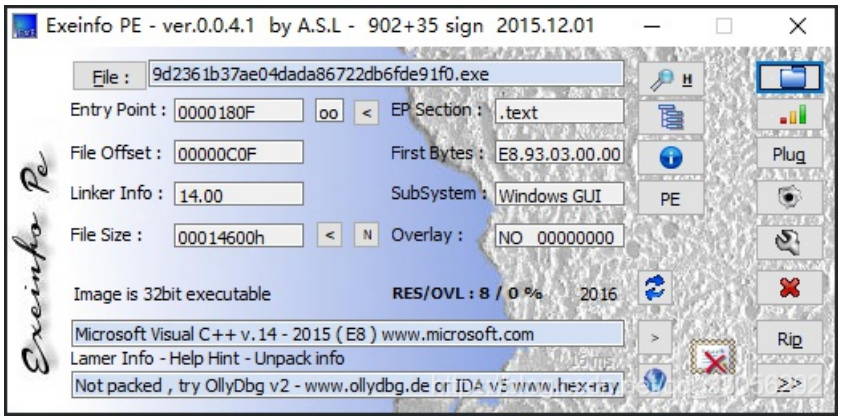
本题是xctf攻防世界中Reverse的进阶区的题srm-50
题目来源： suctf-2016

2. 分析

运行一下：



查壳：



扔进IDA，strings windows里查找e-mail，找到一个字符串`Your E-mail address in not valid.`在DialogFunc函数中被使用。

首先映入眼帘的是一个最外层的if判断，根据这行可以知道String是保存用户输入的email的。除了校验`@`和`.`以外没有其他特别的。

```
if ( strstr(&String, "@") && strstr(&String, ".") && strstr(&String, ".")[1] && strstr(&String, "@")[1] != '.' )
{
    ...
}
```

总得找找输入在哪里吧。往前看，发现GetDlgItemTextA函数，其实就是GetDlgItemText函数，原型为：`int GetDlgItemText( HWND hDlg , int nID, LPTSTR lpStr, int nMaxCount) const;`。用来从文本框中读取文本的。因此，可以判断出v11保存了用户输入的serial number。

```
GetDlgItemTextA(hDlg, 1001, &String, 256);    // 获取用户输入的邮箱
GetDlgItemTextA(hDlg, 1002, v11, 256);        // 获取用户输入的serial number
```

往下看，主要是在内层的if判断中，检查用户输入的serial number，每个条件都必须不满足，为0，才能输出正确的结果。
很明显可以看到的是v11的长度必须16。
奇怪的是，还检查了v12~v23的内容，并没有对这部分空间进行赋值的操作呀。想到v11和v12v23是连续的地址空间，往上看一下v11的定义，发现v11占4个字节，v12~v23只占1个字节。`4+(23-12+1)*1=16`。刚刚好。

```
CHAR String; // [esp+8h] [ebp-340h]
CHAR v11[4]; // [esp+108h] [ebp-240h]
char v12; // [esp+10Ch] [ebp-23Ch]
char v13; // [esp+10Dh] [ebp-23Bh]
char v14; // [esp+10Eh] [ebp-23Ah]
char v15; // [esp+10Fh] [ebp-239h]
char v16; // [esp+110h] [ebp-238h]
char v17; // [esp+111h] [ebp-237h]
char v18; // [esp+112h] [ebp-236h]
char v19; // [esp+113h] [ebp-235h]
char v20; // [esp+114h] [ebp-234h]
char v21; // [esp+115h] [ebp-233h]
char v22; // [esp+116h] [ebp-232h]
char v23; // [esp+117h] [ebp-231h]
```

再去看看读取v11的代码：`GetDlgItemTextA(hDlg, 1002, v11, 256);`，发现是把整个v11都读进来，没有做长度校验。因此这里存在一个溢出的问题，即v11如果输入的是16个字节，那么刚好会覆盖到v23。

其中，v11有4个字节，其余的只有1个字节。读取serial number时，存在溢出。那么我们只要照着if条件给的内容，一个一个去推算，就能把正确的serial number计算出来了。

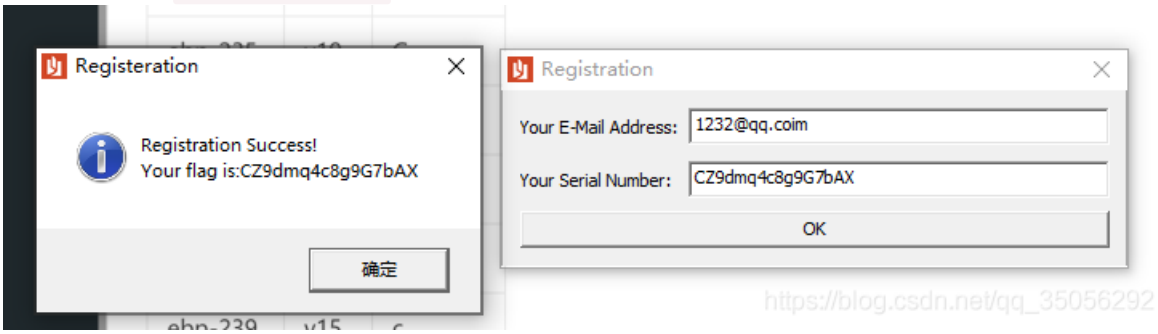
v11~v23在栈中的情况如下：

内存地址	数据	具体内容
ebp-231	v23	X
ebp-232	v22	A
ebp-233	v21	b
ebp-234	v20	7
ebp-235	v19	G
ebp-236	v18	9
ebp-237	v17	g

内存地址	数据	具体内容
ebp-238	v16	8
ebp-239	v15	c
ebp-23A	v14	4
ebp-23B	v13	q
ebp-23C	v12	m
ebp-240	v11	CZ9d

```
if ( a2 != 273 || (unsigned __int16)a3 != 1 )
  return 0;
memset(&String, (unsigned __int16)a3 - 1, 256u);
memset(v11, 0, 256u);
memset(&Text, 0, 256u);
GetDlgItemTextA(hDlg, 1001, &String, 256);    // 获取用户输入的邮箱
GetDlgItemTextA(hDlg, 1002, v11, 256);        // 获取用户输入的serial number
if ( strstr(&String, "@") && strstr(&String, ".") && strstr(&String, ".")[1] && strstr(&String, "@")[1] != '.' )//  根据这行可以判断String应该是用户输入的邮箱
{
  v28 = xmmword_410AA0;                       // 140074663727952857703359243155644310866
  v29 = 1701999980;
  *(_OWORD *)Src = xmmword_410A90;            // 132202756324631589636768955722772211026
  v30 = 46;
  v26 = xmmword_410A80;                       // 43072078423526198075316474294016632163
  v27 = 3830633;
  if ( strlen(v11) != 16                      // 只要有一个条件满足，即失败，因此serial number必须16位
    || v11[0] != 'C'                          // v11[0]为C，下面的依次类推。
    || v23 != 'X'                             // v23为X
    || v11[1] != 'Z'                          // v11[1]为Z，90
    || v11[1] + v22 != 155                    // v22 = 155-v11[1] = 155-90=65=A
    || v11[2] != '9'                          // v11[2]为'9'
    || v11[2] + v21 != 155                    // v21=155-v11[2] = 155-57 = b
    || v11[3] != 'd'                          // v11[3]为d
    || v20 != '7'
    || v12 != 'm'
    || v19 != 'G'
    || v13 != 'q'
    || v13 + v18 != 170                       // v18=170-v13=170-'q'=170-113=9
    || v14 != '4'
    || v17 != 'g'
    || v15 != 'c'
    || v16 != '8' )
  {
    strcpy_s(&Text, 0x100u, (const char *)&v28);
  }
  else
  {
    strcpy_s(&Text, 0x100u, Src);
    strcat_s(&Text, 0x100u, v11);
  }
}
else
{
  strcpy_s(&Text, 0x100u, "Your E-mail address in not valid.");
}
MessageBoxA(hDlg, &Text, "Registeration", 0x40u);
return 1;
}
```


可以看出serial number为：`CZ9dmq4c8g9G7bAX` ，输入程序测试一下吧~



因此最终的flag为：`CZ9dmq4c8g9G7bAX`