Reverse 部分 WP/出题思路

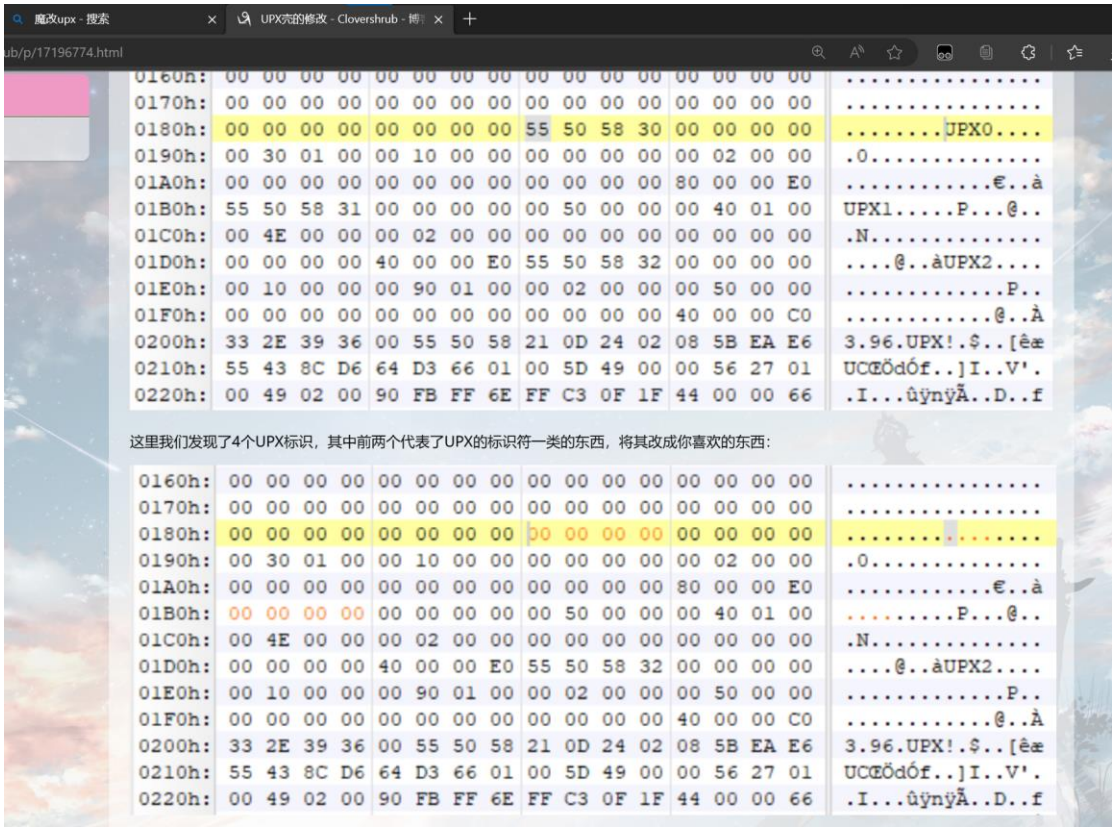
Babymath

Checkin
二十一元一次方程，随机数生成的式子，有多解，上网找个 z3 解多解的脚本就行。

babysmc

checkin
解出来的人数没有想象中的多
UPX 加壳，稍微改了一下

对 UPX 壳最基本的改动，能删的貌似不止这些，设想是做题多点应该能遇见过这玩意，或者上网搜魔改 upx 能搜出来一大堆：



这题是将 UPX 改成了 0x2e0x2e0x2e，改回来就能一键脱壳了。

当然也可以手动脱，@Libr 看的一个教程 https://www.anquanke.com/post/id/272639。

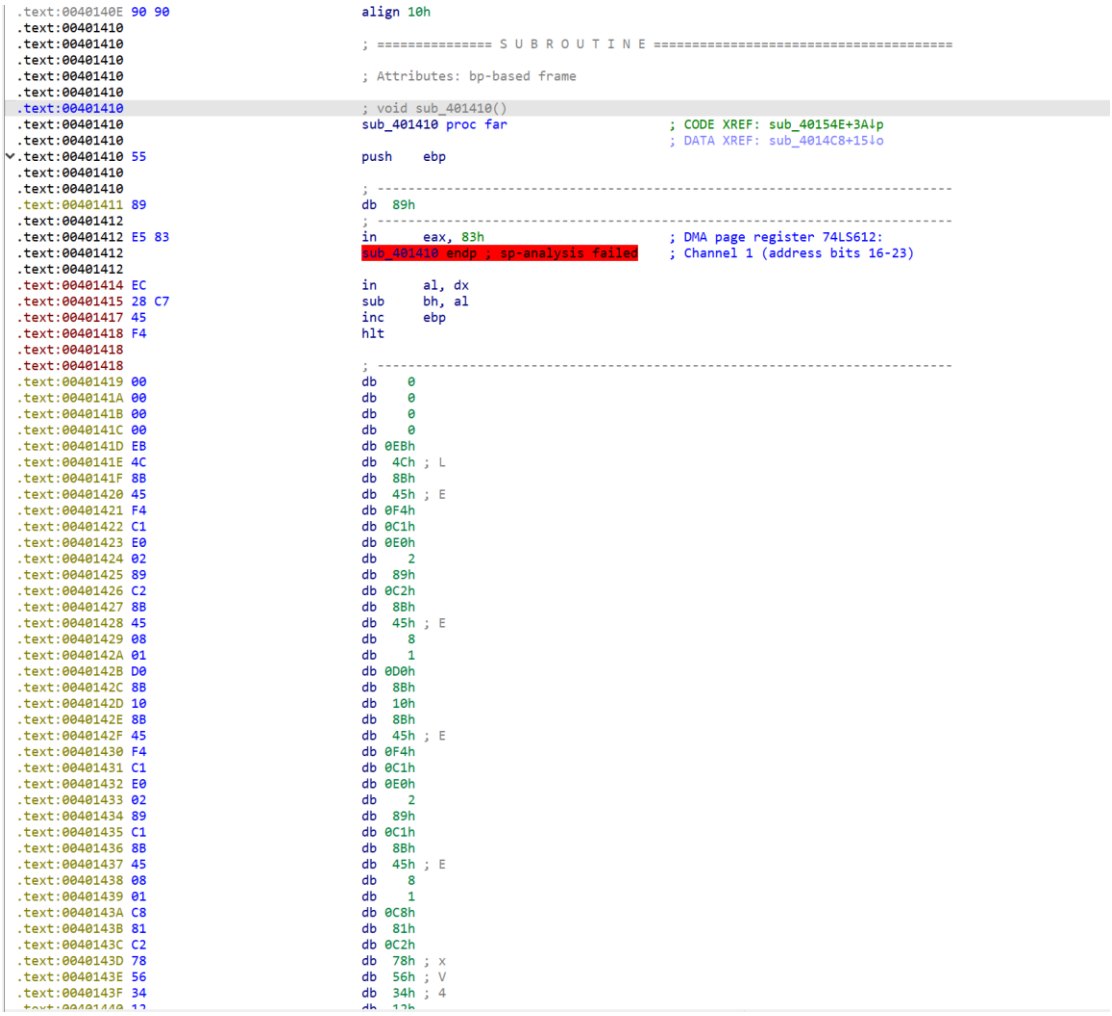
脱完后拖进 ida 看，有个 smc （Self Modifying Code）函数

```
int sub_4014C8()
{
  int result; // eax
  DWORD flOldProtect[2]; // [esp+1Ch] [ebp-1Ch] BYREF
  int v2; // [esp+24h] [ebp-14h]
  int (__usercall *v3)@<eax>(char@<pf>); // [esp+28h] [ebp-10h]
  int i; // [esp+2Ch] [ebp-Ch]

  flOldProtect[0] = (DWORD)malloc(8u);
  v3 = sub_401410;
  v2 = 184;
  flOldProtect[1] = (unsigned int)sub_401410 & 0xFFFFF000;
  VirtualProtect((LPVOID)((unsigned int)sub_401410 & 0xFFFFF000), 0x1000u, 0x80u, flOldProtect);
  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i >= v2 )
      break;
    *((_BYTE *)v3 + i) ^= 0x2Eu;
  }
  return result;
}
```

就是对 check 函数进行异或 0x2e 的操作，异或回来就行。

异或又有两种办法，一种是用 ida 脚本静态，一种是直接调试下断点到 check 函数让程序自行解密，解密完长这样:



把这一段代码选中按 C 键分析，然后 P 键生成函数

注意要先把函数 sub_401410 undefine 了不然会干扰 ida 的分析

生成函数长这样:

```
int __cdecl sub_401410(int a1)
{
  int j; // [esp+18h] [ebp-10h]
  int i; // [esp+1Ch] [ebp-Ch]

  for ( i = 0; i <= 9; ++i )
  {
    *(_DWORD *)(4 * i + a1) += 305419896;
    *(_DWORD *)(4 * i + a1) ^= 0x87654321;
  }
  for ( j = 0; j <= 9; ++j )
  {
    if ( *(_DWORD *)(4 * j + a1) != dword_405020[j] )
    {
      printf("Wrong!");
      exit(0);
    }
  }
  return printf("Right!");
}
```

用密文逆回去就行了。

Babyhash

原本感觉太简单了，已经出了两个签到题，强行撸成了 c++增大分析难度，用"12345"+6 位 key1+"67890"作为 rc4 加密的 key 对 flag 进行加密，6 位 key1 限制了是小写字母，python 脚本爆五分钟能爆出来

```
import hashlib
for a1 in range(97,123):
    print(a1)
    for a2 in range(97,123):
        for a3 in range(97,123):
            for a4 in range(97,123):
                for a5 in range(97,123):
                    for a6 in range(97,123):
                        key=''
                        key='12345'+chr(a1)+chr(a2)+chr(a3)+chr(a4)+chr(a
5)+chr(a6)+'67890'
                        key_md5 = hashlib.md5(key.encode(encoding='utf-
8')).hexdigest()
                        if(key_md5=='f27d71f53ae17679fb352baa5ea326db'):
                          print(key)
```

```
107
108
109
12345maikei67890
110
```

得到 key 后对密文进行 rc4 加密即可。

Ezdebug

elf 文件加了点简单的反调试（难的不会）
一开始 base64 码表直接露出来的，后来有点害羞，决定加个盒隐藏一下。

```
void alarmHandler(int sig)
{
//    printf("time is up, quicker next time\n");
   exit(1);
}
__attribute__((constructor))void setupSig(void)
{//设置程序一开始就执行
   signal(SIGALRM, alarmHandler);
   alarm(3);
}
```

用来限时的，ida 调试时能忽略掉。

```
int checkdebug()
{
    if(ptrace(PTRACE_TRACEME, 0, 0, 0) ==-1)
    {
        return 1;
    }
    return 0;
}
```

原理是一个程序只能被 PTRACE_TRACEME 一次，之后就返回-1 了。
挺好绕，nop 掉就行，但是这题去了符号表+静态编译不好直接看，估计得调试过程中发现。

根据 checkdebug 得到的正在调试与否的信息，对 tea 加密的 key 和 s 盒进行相应的操作

```
152  void Debug()
153  {
154      int debugflag=checkdebug();
155      if(debugflag==0)//正常
156      {
157          for(int i=0;i<4;i++)
158          {
159              teakey[i]^=0x2e;
160          }
161          for(int i=0;i<256;i++)
162          {
163              S[i]^=0xff;
164          }
165          strncpy(realtable,table1,64);
166          box2char(realtable);
167          printf("good boy :)\n");
168      }
169      if(debugflag==1)//调试
170      {
171          for(int i=0;i<4;i++)
172          {
173              teakey[i]^=0xe2;
174          }
175          strncpy(realtable,table2,64);
176          box2char(realtable);
177          printf("bad boy :(\n");
178      }
179
180  }
```

然后是对 input 的加密操作，先 tea 加密，再查 s 盒得出 base64 码表，然后进行 base64 加密，再把加密结果换回 s 盒对应值，结果与 flag 加密值进行比较。

```
181 int check(char *input)
182 {
183     char *base64enc_flag;
184     for(int i=0;i<4;i++)
185     {
186         teaencrypt((int*)&input[i*8],teakey);
187     }
188 
189     strncpy(realtable,table1,64);
190     box2char(realtable);
191     base64enc_flag=base64_encode(input);
192     char2box(base64enc_flag);
193 
194     if(strcmp(boxencflag,base64enc_flag)==0)
195         return 1;
196     return 0;
197 }
```