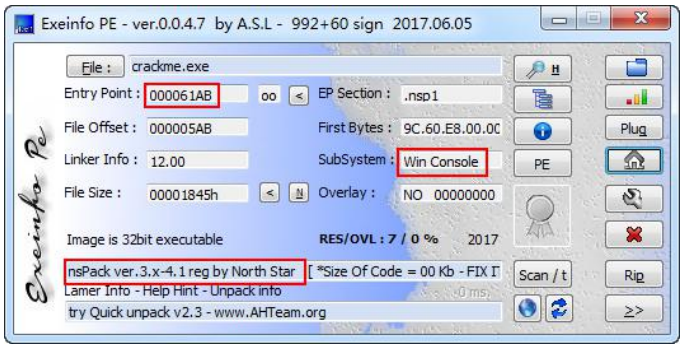
nsPack 手动脱壳

运行程序，无法运行，使用 010editor 打开，发现魔术字"pe"为小写，改为大写"PE"，可正常运行，然后查壳：



这是一个 32 位控制台程序，使用 nsPack 加壳，EP（000061AB）肯定不是 OEP 了。运行此程序，如下：

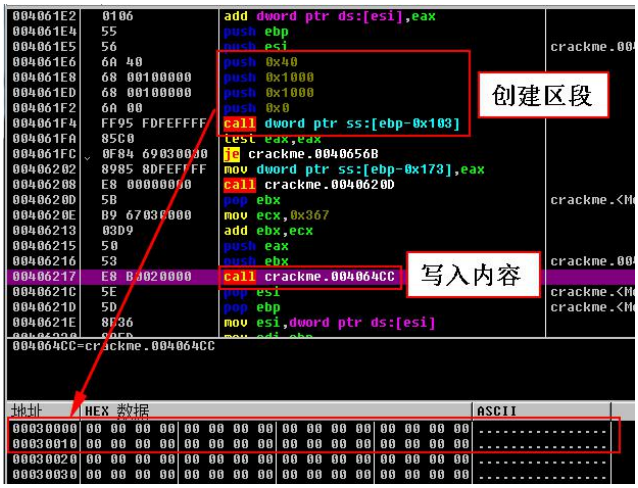
```
C:\Users\nsfocus-cyf>C:\Users\nsfocus-cyf\Desktop\crackme.exe
Please Input Flag:aaaa
error!

C:\Users\nsfocus-cyf>
```

输入 flag，然后校验，错误，返回"error!"，猜测应该使用了标准输出函数，如 printf 等。下面分析壳代码：

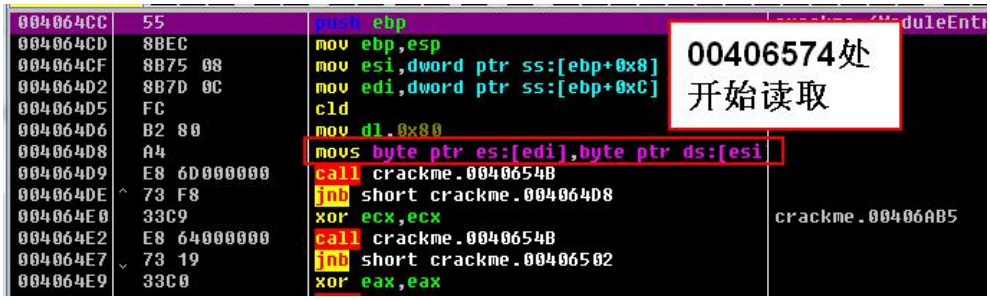
首先是一个保存寄存器的压栈操作：

```
004061AB   9C              pushfd
004061AC   60              pushad
004061AD   E8 00000000     call crackme.004061B2
004061B2   5D              pop ebp
004061B3   83ED 07         sub ebp,0x7
004061B6   8D8D D1FEFFFF   lea ecx,dword ptr ss:[ebp-0x12F]
004061BC   8039 01         cmp byte ptr ds:[ecx],0x1
004061BF   0F84 42020000   je crackme.00406407
004061C5   C601 01         mov byte ptr ds:[ecx],0x1
004061C8   8BC5            mov eax,ebp
```

继续跟进，动态申请了一块内存，生成一个动态的区块：



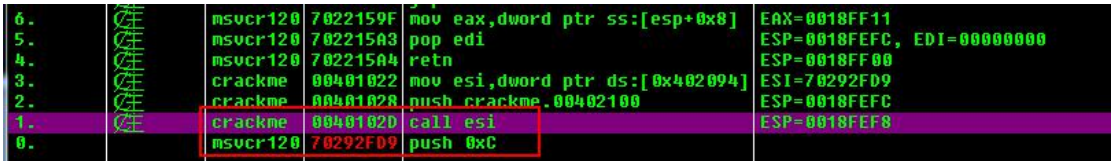
返回地址是 0x00030000，大小是 0x1000，接着在 0x00406217 处调用函数，来填充该区段，crackme.004064cc 处代码如下：



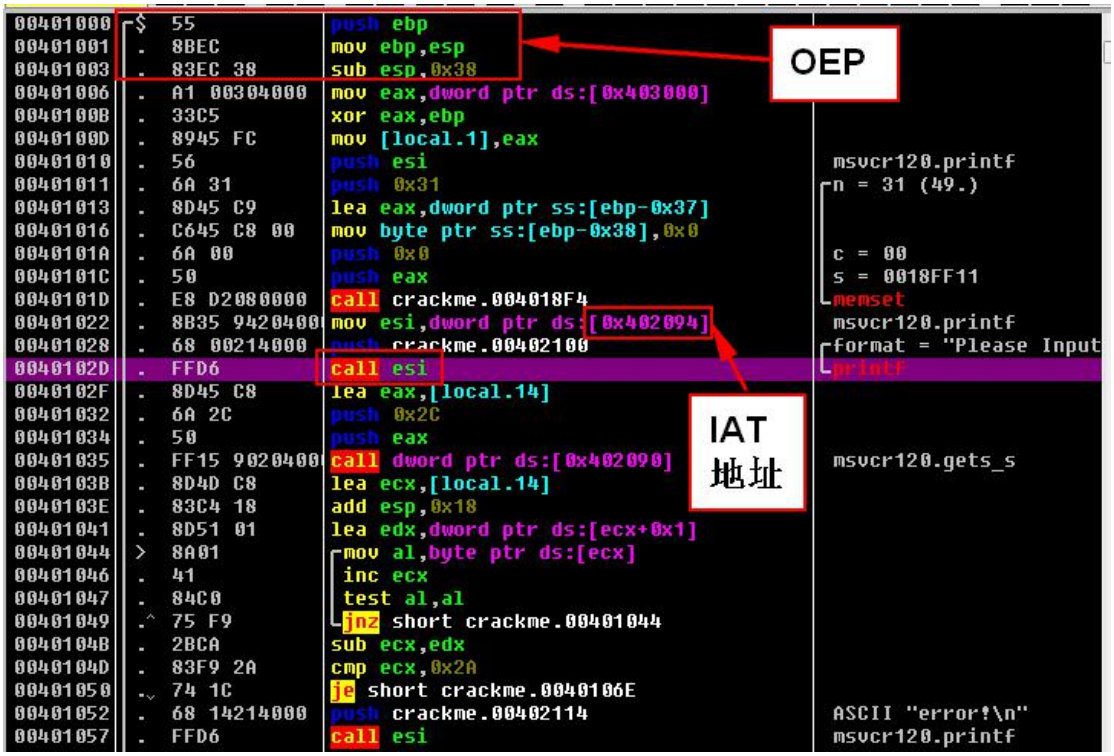
然后把 0x003005AA 放入 eax，执行 call eax，然后在 eip=0x00406289 处 free 掉动态申请的内存。

这些解密例程都不是我们关心的，我们关心的只有两件事：OEP 和 IAT，首先 OEP 不太好找，先找 IAT，找到 IAT 就离 OEP 不远了，首先全速运行程序，调用了 msvcr120.printf 函数，查找该函数地址：

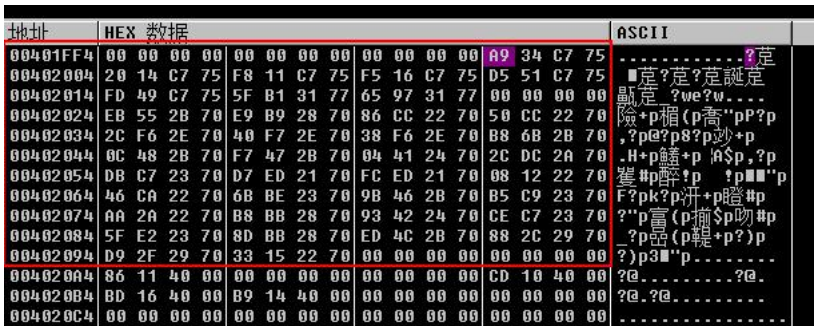
在该地址处下断，并且启用跟踪步入，结果如下：



可以看到，是从 0x0040102d 的 call esi 来调用的 printf，跟进 0x0040102d，如下：

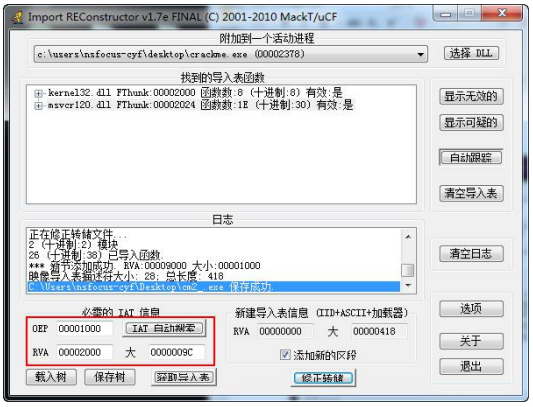


开头是压栈和 ebp 赋值，后续是程序逻辑，所以这里就是 OEP。然后看调用 printf 的语句 call esi，查找 esi 值得来源，即 0x00402094 地址中的值，则 0x00402094 就是 IAT 表的位置。查看 IAT 表：



从 0x00402000 处开始，到 0x0040209c 结束，size=0x0040209c-0x00402000=0x9c。所需要素均具备：OEP：0x00401000；IAT-RVA：0x00402000；IAT-size：0x9c。接下来 dump 内存：

保存为 cm_dump.exe，然后使用 impREC 重组 IAT：



保存为 cm_ump_.exe，直接运行该程序，可正常运行：

接下来逆向找 flag，放入 IDA 中，查看逻辑，或者直接 F5，如下：

```
signed int start()
{
  signed int result; // eax@2
  signed int v1; // eax@3
  char Buf; // [sp+4h] [bp-38h]@1
  char Dst; // [sp+5h] [bp-37h]@1

  Buf = 0;
  memset(&Dst, 0, 0x31u);
  printf("Please Input Flag:");
  gets_s(&Buf, 0x2Cu);
  if ( strlen(&Buf) == 42 )
  {
    v1 = 0;
    while ( (*(&Buf + v1) ^ byte_402130[v1 % 16]) == dword_402150[v1] )
    {
      if ( ++v1 >= 42 )
      {
        printf("right!\n");
        goto LABEL_8;
      }
    }
    printf("error!\n");
LABEL_8:
    result = 0;
  }
  else
  {
    printf("error!\n");
    result = -1;
  }
  return result;
}
```

在nsp0段中

写出 c 正向算法：

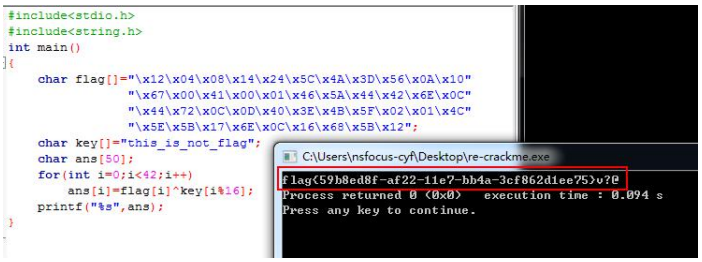
```
#include<stdio.h>
#include<string.h>
int main()
{
    char ans[50];
    char flag[]="0x12,0x04,0x08,0x14,0x24,0x5C,0x4A,0x3D,0x56,0x0A
    char key[]="this_is_not_flag";
    scanf("%s",ans);
    if(strlen(ans)==42)
    {
        int i=0;
        while(ans[i]^key[i%16]==flag[i])
        {
            ++i;
            if(i>=42)
            {
                printf("right.\n");
                result=0;
            }
        }
        printf("error.\n");
    }
    else
    {
        printf("error.\n");
        result=-1;
    }
    return result;
}
```

这个算法大致思路是：存储一个数组 flag，里面是乱码，存有一个字符串 key “this_is_not_flag”，然后取下标 i 与 16 取余，去除 key 中该余数下表的字符，然后与下标为 i 的 flag 中字符去异或，得到 flag。

然后写逆算法：

```
#include<stdio.h>
#include<string.h>
int main()
{
    char flag[]="\x12\x04\x08\x14\x24\x5C\x4A\x3D\x56\x0A\x10"
                "\x67\x00\x41\x00\x01\x46\x5A\x44\x42\x6E\x0C"
                "\x44\x72\x0C\x0D\x40\x3E\x4B\x5F\x02\x01\x4C"
                "\x5E\x5B\x17\x6E\x0C\x16\x68\x5B\x12";
    char key[]="this_is_not_flag";
    char ans[50];
    for(int i=0;i<42;i++)
        ans[i]=flag[i]^key[i%16];
    printf("%s",ans);
}
```

编译逆算法 c 文件，运行，即可得到 flag：


#include<stdio.h>
#include<string.h>
int main()
{
char flag[]="\x12\x04\x08\x14\x24\x5C\x4A\x3D\x56\x0A\x10"
"\x67\x00\x41\x00\x01\x46\x5A\x44\x42\x6E\x0C"
"\x44\x72\x0C\x0D\x40\x3E\x4B\x5F\x02\x01\x4C"
"\x5E\x5B\x17\x6E\x0C\x16\x68\x5B\x12";
char key[]="this_is_not_flag";
char ans[50];
for(int i=0;i<42;i++)
ans[i]=flag[i]^key[i%16];
printf("%s",ans);
}
C:\Users\nsfocus-cyf\Desktop\re-crackme.exe
flag{59b8ed8f-af22-11e7-bb4a-3cf862d1ee75}v?@
Process returned 0 (0x0) execution time : 0.094 s
Press any key to continue.