170621 逆向-CrackMe之027

原创

奈沙夜影 于 2017-06-22 03:12:46 发布 364 收藏
分类专栏： CrackMe

本文链接：https://blog.csdn.net/whklhhhh/article/details/73557511


CrackMe 专栏收录该内容
83 篇文章 2 订阅
订阅专栏

1625-5 王子昂 总结《2017年6月21日》【连续第262天总结】

A. CrackMe

B. 大概了解了一下ELF文件，Linux平台下的可执行文件，使用终端可以执行

在win平台下可以直接使用IDA进行处理

不过IDA动态调试很懵，F5不是找不到入口函数就是未定义的函数，看的几篇writeup中都直接分析了代码和算法，并没有介绍开

没办法，拖了第27个CrackMe来练习，类型是CD-CHECK

光盘检测类型的，让我们来看看这是什么原理吧

首先PEiD检测，VC++写的

OD运行，有一个you lost的弹窗。那么很明显，查MessageBox的call，成功锁定失败弹窗

向上翻，看到了有一堆Push各种盘符的指令。估计是储存性质，然后循环检测吧

```
0040121A   .  68 9C304000   push Cosh_1.0040309C                ;  C:\
0040121F   .  8D4D A4       lea ecx,dword ptr ss:[ebp-0x5C]
00401222   .  E8 79040000   call <jmp.&MFC42.#537>
00401227   .  33DB          xor ebx,ebx
00401229   .  68 98304000   push Cosh_1.00403098               ;  D:\
0040122E   .  8D4D A8       lea ecx,dword ptr ss:[ebp-0x58]
00401231   .  895D FC       mov dword ptr ss:[ebp-0x4],ebx
00401234   .  E8 67040000   call <jmp.&MFC42.#537>
00401239   .  68 94304000   push Cosh_1.00403094               ;  E:\
0040123E   .  8D4D AC       lea ecx,dword ptr ss:[ebp-0x54]
00401241   .  C645 FC 01    mov byte ptr ss:[ebp-0x4],0x1
00401245   .  E8 56040000   call <jmp.&MFC42.#537>
```

```
0040124A   .  68 90304000   push Cosh_1.00403090                ;  F:\
0040124F   .  8D4D B0       lea ecx,dword ptr ss:[ebp-0x50]
00401252   .  C645 FC 02    mov byte ptr ss:[ebp-0x4],0x2
00401256   .  E8 45040000   call <jmp.&MFC42.#537>
0040125B   .  68 8C304000   push Cosh_1.0040308C                ;  G:\
00401260   .  8D4D B4       lea ecx,dword ptr ss:[ebp-0x4C]
00401263   .  C645 FC 03    mov byte ptr ss:[ebp-0x4],0x3
00401267   .  E8 34040000   call <jmp.&MFC42.#537>
0040126C   .  68 88304000   push Cosh_1.00403088                ;  H:\
00401271   .  8D4D B8       lea ecx,dword ptr ss:[ebp-0x48]
00401274   .  C645 FC 04    mov byte ptr ss:[ebp-0x4],0x4
00401278   .  E8 23040000   call <jmp.&MFC42.#537>
0040127D   .  68 84304000   push Cosh_1.00403084                ;  I:\
00401282   .  8D4D BC       lea ecx,dword ptr ss:[ebp-0x44]
00401285   .  C645 FC 05    mov byte ptr ss:[ebp-0x4],0x5
00401289   .  E8 12040000   call <jmp.&MFC42.#537>
0040128E   .  68 80304000   push Cosh_1.00403080                ;  J:\
00401293   .  8D4D C0       lea ecx,dword ptr ss:[ebp-0x40]
00401296   .  C645 FC 06    mov byte ptr ss:[ebp-0x4],0x6
0040129A   .  E8 01040000   call <jmp.&MFC42.#537>
0040129F   .  68 7C304000   push Cosh_1.0040307C                ;  K:\
004012A4   .  8D4D C4       lea ecx,dword ptr ss:[ebp-0x3C]
004012A7   .  C645 FC 07    mov byte ptr ss:[ebp-0x4],0x7
004012AB   .  E8 F0030000   call <jmp.&MFC42.#537>
004012B0   .  68 78304000   push Cosh_1.00403078                ;  L:\
004012B5   .  8D4D C8       lea ecx,dword ptr ss:[ebp-0x38]
004012B8   .  C645 FC 08    mov byte ptr ss:[ebp-0x4],0x8
004012BC   .  E8 DF030000   call <jmp.&MFC42.#537>
004012C1   .  68 74304000   push Cosh_1.00403074                ;  M:\
004012C6   .  8D4D CC       lea ecx,dword ptr ss:[ebp-0x34]
004012C9   .  C645 FC 09    mov byte ptr ss:[ebp-0x4],0x9
004012CD   .  E8 CE030000   call <jmp.&MFC42.#537>
004012D2   .  68 70304000   push Cosh_1.00403070                ;  N:\
004012D7   .  8D4D D0       lea ecx,dword ptr ss:[ebp-0x30]
004012DA   .  C645 FC 0A    mov byte ptr ss:[ebp-0x4],0xA
004012DE   .  E8 BD030000   call <jmp.&MFC42.#537>
004012E3   .  68 6C304000   push Cosh_1.0040306C                ;  O:\
004012E8   .  8D4D D4       lea ecx,dword ptr ss:[ebp-0x2C]
```

004012EB . C645 FC 0B mov byte ptr ss:[ebp-0x4],0xB

004012EF . E8 AC030000 call <jmp.&MFC42.#537>

004012F4 . 68 68304000 push Cosh_1.00403068 ; P:\

004012F9 . 8D4D D8 lea ecx,dword ptr ss:[ebp-0x28]

004012FC . C645 FC 0C mov byte ptr ss:[ebp-0x4],0xC

00401300 . E8 9B030000 call <jmp.&MFC42.#537>

再往下翻，找到了成功弹窗：

00401485 > \53 push ebx

00401486 . 68 34304000 push Cosh_1.00403034 ; You did it

0040148B . 68 20304000 push Cosh_1.00403020 ; Well done, Cracker

00401490 .^ E9 14FFFFFF jmp Cosh_1.004013A9

从00401485向上追踪，发现是0040138C，一个在失败弹窗上方不远处的跳转，那么这个je很明显就是关键跳了

再往上是一个CreateFileA的API，跟昨天检测KeyFile的方法类似吧，应该

再向上看看：

00401346 . FF75 E8 push dword ptr ss:[ebp-0x18] ; /RootPathName = "C:\"

00401349 . FF15 04204000 call dword ptr ds:[<&KERNEL32.GetDriveTy>; \GetDriveTypeA

0040134F . 83F8 03 cmp eax,0x3

00401352 . 74 3E je short Cosh_1.00401392 ; 关键跳x1

这个GetDriveTypeA的API是干什么的呢，硬盘类型，应该是用来判断光驱的吧

下断，运行试试

发现只要是本地存在的盘符（CDEFHI等） eax就会出3，使得直接跳过CreateFileA的API，然后再循环并使表示盘符的字符串指针指向下一个盘符。因此就是逐个判断之前push的盘符哪些是本地的抛弃掉，然后在非本地盘符的磁盘上查找CD_CHECK.DAT

因此爆破的话只需要在跳转的时候直接设置jmp到成功处即可

而如果想通过非爆破的形式，理论上来说只要插入一张根目录下有CD_CHECK.DAT文件的光盘应该就能解决。甚至其他软盘U盘网络驱动器应该也是可以的。

但是实验结果不行。因为爆破掉DriveType的结果，在该盘符下创建了CD_CHECK.DAT以后，CreateFileA的返回值仍然是-1导致失败。

查询了一下得知：

GetDriveTypeA (String drive) Library "kernel32.dll"

参数为一个盘符（如"C:"），

返回值：1表示未知，2表示软驱，3表示本地硬盘，4表示网络驱动器，5表示光驱。因此如下代码可以获得光盘的盘符：

```
HANDLE CreateFile(
    LPCTSTR lpFileName, //指向文件名的指针
    DWORD dwDesiredAccess, //访问模式（写/读）
    DWORD dwShareMode, //共享模式
    LPSECURITY_ATTRIBUTES lpSecurityAttributes, //指向安全属性的指针
    DWORD dwCreationDisposition, //如何创建
    DWORD dwFlagsAndAttributes, //文件属性
    HANDLE hTemplateFile //用于复制文件句柄
    );
```

其中dwCreationDisposition就是第三个push进的参数，即Mode

```
00401368  . 53          push ebx                        ; /hTemplateFile = NULL
00401369  . 53          push ebx                        ; |Attributes = 0
0040136A  . 53          push ebx                        ; |Mode = 0x0
0040136B  . 53          push ebx                        ; |pSecurity = NULL
0040136C  . 6A 01       push 0x1                        ; |ShareMode = FILE_SHARE_READ
0040136E  . 68 00000080 push 0x80000000                 ; |Access = GENERIC_READ
00401373  . 50          push eax                        ; |FileName = "D:\CD_CHECK.DAT"
00401374  . FF15 00204000 call dword ptr ds:[<&KERNEL32.CreateFile>; \CreateFileA
```

而dwCreationDisposition可以接受多个值，代表多种常量，诸如Create_Always、Open_Alaways等等。唯独没有0

也就是说，这个参数从生成的时候就已经错误了

因此无法正常通过验证

手动修改堆栈使Mode=3，即Read_Always则可以成功，使eax返回值为正数，最终成功

因此这题只能爆破了_(:з」∠)_

C. 明日计划

CrackMe(28)