[原创] windows10 patchguard绕过讨论 [复制链接]

mengwuji

升级 100%

220 主题 | 66 精华 | 1478 积分

发消息

发表于 3 天前 | 只看该作者 | 只看大图

楼主 电梯直达

梦织未来(www.mengwuji.net)

作者：mengwuji

最近很少发帖，主要是不知道写什么好。有的坛友建议我写帖子多活跃下气氛，我思来想去，还是写点儿别人没写过的吧。windows10 patchguard我在网络上搜索一圈基本没看到分析文章也没解决方案，有的都是需要修改内核文件永久性绕过，而非动态绕过。当然如果使用虚拟化或者intel processor trace来绕过的不在本贴讨论范围之内，本贴只是讨论不提供绕过代码，下面进入主题~

以前讨论过windows7上绕过patchguard的方法，比较常用的有hook关键地方绕过，或者解密pg的context后，修改pg的检测代码来绕过。我用的方法是解密pg的context修改关键点绕过的，这种方法之后也经过了大量考验的，非常稳定。windows7解密context的方法是CmpAppendDllSection函数完成的，算法很简单。windows10中，pg执行开始点也是从CmpAppendDllSection开始的，所以我把代码复制上来说说这种加密方式的弊端。

```
INIT:0000000140741AF0                         CmpAppendDllSection proc near           ; DATA XREF: sub_14072DACC+1AC3To
INIT:0000000140741AF0                                 db      2Eh
INIT:0000000140741AF0 2E 48 31 11                     xor     [rcx], rdx
INIT:0000000140741AF4 48 31 51 08                     xor     [rcx+8], rdx
INIT:0000000140741AF8 48 31 51 10                     xor     [rcx+10h], rdx
INIT:0000000140741AFC 48 31 51 18                     xor     [rcx+18h], rdx
INIT:0000000140741B00 48 31 51 20                     xor     [rcx+20h], rdx
INIT:0000000140741B04 48 31 51 28                     xor     [rcx+28h], rdx
INIT:0000000140741B08 48 31 51 30                     xor     [rcx+30h], rdx
INIT:0000000140741B0C 48 31 51 38                     xor     [rcx+38h], rdx
INIT:0000000140741B10 48 31 51 40                     xor     [rcx+40h], rdx
INIT:0000000140741B14 48 31 51 48                     xor     [rcx+48h], rdx
INIT:0000000140741B18 48 31 51 50                     xor     [rcx+50h], rdx
INIT:0000000140741B1C 48 31 51 58                     xor     [rcx+58h], rdx
INIT:0000000140741B20 48 31 51 60                     xor     [rcx+60h], rdx
INIT:0000000140741B24 48 31 51 68                     xor     [rcx+68h], rdx
INIT:0000000140741B28 48 31 51 70                     xor     [rcx+70h], rdx
INIT:0000000140741B2C 48 31 51 78                     xor     [rcx+78h], rdx
INIT:0000000140741B30 48 31 91 80 00 00 00            xor     [rcx+80h], rdx
INIT:0000000140741B37 48 31 91 88 00 00 00            xor     [rcx+88h], rdx
INIT:0000000140741B3E 48 31 91 90 00 00 00            xor     [rcx+90h], rdx
INIT:0000000140741B45 48 31 91 98 00 00 00            xor     [rcx+98h], rdx
INIT:0000000140741B4C 48 31 91 A0 00 00 00            xor     [rcx+0A0h], rdx
INIT:0000000140741B53 48 31 91 A8 00 00 00            xor     [rcx+0A8h], rdx
INIT:0000000140741B5A 48 31 91 B0 00 00 00            xor     [rcx+0B0h], rdx
INIT:0000000140741B61 48 31 91 B8 00 00 00            xor     [rcx+0B8h], rdx
INIT:0000000140741B68 48 31 91 C0 00 00 00            xor     [rcx+0C0h], rdx
INIT:0000000140741B6F 31 11                           xor     [rcx], edx
INIT:0000000140741B71 48 8B C2                        mov     rax, rdx
INIT:0000000140741B74 48 8B D1                        mov     rdx, rcx
INIT:0000000140741B77 8B 8A C4 00 00 00               mov     ecx, [rdx+0C4h]
INIT:0000000140741B7D
INIT:0000000140741B7D                         loc_140741B7D:                          ; CODE XREF: CmpAppendDllSection+98↓j
INIT:0000000140741B7D 48 31 84 CA C0 00 00 00         xor     [rdx+rcx*8+0C0h], rax
INIT:0000000140741B85 48 D3 C8                        ror     rax, cl
INIT:0000000140741B88 E2 F3                           loop    loc_140741B7D
INIT:0000000140741B8A 8B 82 58 05 00 00               mov     eax, [rdx+558h]
INIT:0000000140741B90 48 03 C2                        add     rax, rdx
INIT:0000000140741B93 48 83 EC 28                     sub     rsp, 28h
INIT:0000000140741B97 FF D0                           call    rax
INIT:0000000140741B99 48 83 C4 28                     add     rsp, 28h
INIT:0000000140741B9D 4C 8B 80 08 01 00 00            mov     r8, [rax+108h]
INIT:0000000140741BA4 48 8D 88 08 05 00 00            lea     rcx, [rax+508h]
INIT:0000000140741BAB BA 01 00 00 00                  mov     edx, 1
INIT:0000000140741BB0 41 FF E0                        jmp     r8
INIT:0000000140741BB0                         CmpAppendDllSection endp
```


代码以xor [rcx], rdx开始，其中rcx实际是CmpAppendDllSection地址，rdx是context的解密key，我们要想解密context必须拿到这个key才行，那么大家看上面代码能想出办法破解出key吗？请大家静静的思考几分钟看看~(当初我对着pg代码看了一个星期，才想到解密方法...><)

上面代码有个逻辑问题，我们可以利用这种逻辑漏洞找出key。假设现在我们知道CmpAppendDllSection的地址，但是CmpAppendDllSection本身被加密了，我们现在只知道CmpAppendDllSection加密后的密文内容，和CmpAppendDllSection没加密的内容(ida看看就知道了明文内容)，如何解密呢？我们要用到一个非常简单的算法，既是：K = A ^ B ，那么A = B ^ K。现在我们来设几个值，设：

CmpAppendDllSection+0x0的密文内容为A，大小8字节；
CmpAppendDllSection+0x0的明文内容为B，大小8字节；
CmpAppendDllSection+0x8的密文内容为C，大小8字节；
CmpAppendDllSection+0x8的明文内容为D，大小8字节；
解密KEY定为K；
因为解密CmpAppendDllSection+0x0和解密CmpAppendDllSection+0x8用到的key都是相同的，那按照图中代码，他们存在这种关系：K = A ^ B，K = C ^ D，所以能得出A ^ B == C ^ D。那么解法就出来了，我们遍历系统内存时，为了判断此内存是否是pg的context，我们就可以读出内容，然后使A ^ B == C ^ D关系成立(为了保险你还可以多判断些字节的内容，方法是一样的)，判断成立后就可以直接用K = A ^ B解出key的值，进而解出整个context，修改完成后再给加密回去，解密与加密算法代码如下：

```
01.
02.  static void AttackPatchGuardEncryptCode(PUCHAR Context, ULONG_PTR ContextKey, ULONG_PTR
     ContextSizeOfBytes)
03.  {
04.  auto pTempMem = reinterpret_cast<PULONG_PTR>(ExAllocatePool(NonPagedPool, ContextSizeOfBytes));
05.  RtlCopyMemory(pTempMem, Context, ContextSizeOfBytes);
06.  //首先解密出context头部的CmpAppendDllSection解密函数
07.  for (auto i  = 0;i < 0xC8/sizeof(ULONG_PTR); i++)
08.  {
09.    pTempMem[i] ^= ContextKey;
10.  }
11.  auto FollowContextSize = pTempMem[0xC0 / sizeof(ULONG_PTR)] >> 32;
12.  auto TempSize = FollowContextSize;
13.  auto FollowContextKey = ContextKey;
14.  //解密剩下的部分
15.  do {
16.    pTempMem[(0xC0 / sizeof(ULONG_PTR)) + TempSize] ^= FollowContextKey;
17.    auto RorBit = static_cast<UCHAR>(TempSize);
18.    FollowContextKey = ROR(FollowContextKey, RorBit, 64);
19.  } while (--TempSize);
20.  //以上解密完成，我们接下去修改context内容
21.  auto TempContext = reinterpret_cast<UCHAR*>(pTempMem);
22.  for (auto i = 0;i < ContextSizeOfBytes; i++)
23.  {
24.    if ((i + 0x84 + 0x16) < ContextSizeOfBytes && \
25.     memcmp(TempContext + i + 0x84,
     "\x48\x8B\xD1\x8B\x8A\xC4\x00\x00\x00\x48\x31\x84\xCA\xC0\x00\x00\x00\x48\xD3\xC8\xE2\xF3", 0x16) == 0)
26.    {
27.     LOG_DEBUG(" -- CmpAppendDllSection address:%p", TempContext + i);
28.     LOG_DEBUG(" -- CmpAppendDllSection address content:%p", *(ULONG_PTR*)(TempContext + i));
29.    }
30.  }
31.  //头加密回去
32.  for (auto i = 0; i < 0xC8 / sizeof(ULONG_PTR); i++)
33.  {
34.    pTempMem[i] ^= ContextKey;
35.  }
36.  TempSize = FollowContextSize;
37.  FollowContextKey = ContextKey;
38.  //尾加密回去
39.  do {
40.    pTempMem[(0xC0 / sizeof(ULONG_PTR)) + TempSize] ^= FollowContextKey;
41.    auto RorBit = static_cast<UCHAR>(TempSize);
42.    FollowContextKey = ROR(FollowContextKey, RorBit, 64);
43.  } while (--TempSize);
44.  RtlCopyMemory(Context, pTempMem, ContextSizeOfBytes);
45.  ExFreePool(pTempMem);
46.  }
47.
```

复制代码

调用AttackPatchGuardEncryptCode的示例代码如下：

```
01.
02.  auto TempKey = *reinterpret_cast<ULONG_PTR*>(StartAddress + i + 0x78) ^ 0x31000000C0913148;
03.     if ((*(ULONG_PTR*)(StartAddress + i + 0x78 + 0x08) ^ 0x8BD18B48C28B4811) == TempKey &&
04.      (*(ULONG_PTR*)(StartAddress + i + 0x78 + 0x10) ^ 0x843148000000C48A) == TempKey &&
05.      (*(ULONG_PTR*)(StartAddress + i + 0x78 + 0x18) ^ 0xC8D348000000C0CA) == TempKey)
06.     {
07.      //以上条件满足说明找到了密文，我们来接着找contextkey进行解密
08.      auto ContextKey = (*(ULONG_PTR*)(StartAddress + i + 0x8)) ^ 0x1851314810513148;
09.      auto ContextSizeOfBytes = (((*(ULONG_PTR*)(StartAddress + i + 0xc0)) ^ ContextKey) >> 32) * 0x8;
    //context+0xc4是保存从+0xc8偏移context后面整个加密长度(换算成字节乘以0x8),注意只有4字节
10.      ContextSizeOfBytes += 0xC8;  //加上前面的长度
11.      LOG_DEBUG("ContextKey:%p    ContextSizeOfBytes:%x\n", ContextKey, ContextSizeOfBytes);
12.      if ((i + ContextSizeOfBytes) <= SizeOfBytes)
13.      {
14.       AttackPatchGuardEncryptCode(StartAddress + i, ContextKey, ContextSizeOfBytes);
15.      }
16.     }
17.
    复制代码
```

以上是windows7 pg绕过方法，下面我们来讨论windows10的。
前几天开始研究windows10的pg时，直接把内核模块拖入ida中发现CmpAppendDllSection函数依然存在，所以确定windows10的pg context解密算法没变，于是套用了windows7的方法，结果没找到context。开始以为是内核内存有些给疏漏了，所以没找到，花了两天时间去研究pg可能申请内存的方法。发现pg在某些情况下是通过ExAllocatePoolWithTag申请非分页内存，某些情况下是通过MmAllocateIndependentPages申请内存的。于是我研究了下MmAllocateIndependentPages申请内存的遍历方法，当然大多数情况都是通过ExAllocatePoolWithTag申请的内存来存放context。可惜的是，遍历后依然没有找到context，我十分郁闷，难道还能跑到分页内存池去不成，绝对不可能好吧！经过一些技巧性实验，终于发现了问题所在。
windows10在CmpAppendDllSection解密算法不变的情况下，又多加了一层算法来解密CmpAppendDllSection函数。也就变成了执行pg代码前，先用一种算法解密一次CmpAppendDllSection，然后执行CmpAppendDllSection时，CmpAppendDllSection再解密自身和剩下的context内容。我找到了解密算法，如下：

```
01.
02.  .text:0000000140151130                                          CallPatchGuard_1 proc near
           ; DATA XREF: .rdata:0000000140258954↓o
03.  .text:0000000140151130 40 53                                                 push    rbx
04.  .text:0000000140151132 55                                                    push    rbp
05.  .text:0000000140151133 48 83 EC 28                                           sub     rsp, 28h
06.  .text:0000000140151137 48 8B EA                                              mov     rbp, rdx
07.  .text:000000014015113A 88 4D 44                                              mov     [rbp+44h],
     cl
08.  .text:000000014015113D 84 C9                                                 test    cl, cl
09.  .text:000000014015113F 0F 84 07 02 00 00                                     jz
      loc_14015134C
10.  .text:0000000140151145 E9 2F 01 00 00                                        jmp
        loc_140151279
11.  .text:000000014015114A                                          ; ---------------------------------
     ---------------------------------------
12.  .text:000000014015114A
13.  .text:000000014015114A                                          loc_14015114A:
           ; CODE XREF: CallPatchGuard_1+1FB↓j
14.  .text:000000014015114A 45 33 DB                                              xor     r11d, r11d
15.  .text:000000014015114D 44 89 5D 40                                           mov     [rbp+40h],
     r11d
16.  .text:0000000140151151 48 8B 5D 28                                           mov     rbx,
     [rbp+28h]
```

```
.text:0000000140151155
.text:0000000140151155                                                  loc_140151155:
      ; CODE XREF: CallPatchGuard_1+D5↑j
.text:0000000140151155 4D 8B 01                                         mov     r8, [r9]
.text:0000000140151158 4C 89 85 18 01 00 00                             mov     [rbp+118h],
r8
.text:000000014015115F 49 8B D0                                         mov     rdx, r8
.text:0000000140151162 48 8B 05 77 E3 22 00                             mov     rax,
cs:KiWaitNever
.text:0000000140151169 48 33 D0                                         xor     rdx, rax
.text:000000014015116C 8B C8                                            mov     ecx, eax
.text:000000014015116E 48 D3 C2                                         rol     rdx, cl
.text:0000000140151171 48 33 D3                                         xor     rdx, rbx
.text:0000000140151174 48 0F CA                                         bswap   rdx
.text:0000000140151177 48 33 15 E2 E4 22 00                             xor     rdx,
cs:KiWaitAlways
.text:000000014015117E 49 89 11                                         mov     [r9], rdx
.text:0000000140151181 41 8B C3                                         mov     eax, r11d
.text:0000000140151184 49 0F AF C2                                      imul    rax, r10
.text:0000000140151188 48 03 C2                                         add     rax, rdx
.text:000000014015118B 49 89 01                                         mov     [r9], rax
.text:000000014015118E 41 8B C8                                         mov     ecx, r8d
.text:0000000140151191 F7 D1                                            not     ecx
.text:0000000140151193 83 E1 3F                                         and     ecx, 3Fh
.text:0000000140151196 B8 C8 00 00 00                                   mov     eax, 0C8h
.text:000000014015119B 41 2B C3                                         sub     eax, r11d
.text:000000014015119E 41 0F AF C3                                      imul    eax, r11d
.text:00000001401511A2 48 D3 C8                                         ror     rax, cl
.text:00000001401511A5 48 33 D8                                         xor     rbx, rax
.text:00000001401511A8 48 89 5D 28                                      mov     [rbp+28h],
rbx
.text:00000001401511AC 41 83 E0 3F                                      and     r8d, 3Fh
.text:00000001401511B0 41 8A C8                                         mov     cl, r8b
.text:00000001401511B3 48 D3 C3                                         rol     rbx, cl
.text:00000001401511B6 48 89 5D 28                                      mov     [rbp+28h],
rbx
.text:00000001401511BA 49 03 DA                                         add     rbx, r10
.text:00000001401511BD 48 89 5D 28                                      mov     [rbp+28h],
rbx
.text:00000001401511C1 45 33 C0                                         xor     r8d, r8d
.text:00000001401511C4 44 89 45 48                                      mov     [rbp+48h],
r8d
.text:00000001401511C8
.text:00000001401511C8                                                  loc_1401511C8:
      ; CODE XREF: CallPatchGuard_1+C0↑j
.text:00000001401511C8 41 0F B6 01                                      movzx   eax, byte
ptr [r9]
.text:00000001401511CC 83 E0 0F                                         and     eax, 0Fh
.text:00000001401511CF 0F B6 54 05 30                                   movzx   edx, byte
ptr [rbp+rax+30h]
.text:00000001401511D4 49 83 21 F0                                      and     qword ptr
[r9], 0FFFFFFFFFFFFFFF0h
.text:00000001401511D8 49 0B 11                                         or      rdx, [r9]
.text:00000001401511DB 49 89 11                                         mov     [r9], rdx
.text:00000001401511DE 48 C1 CA 04                                      ror     rdx, 4
.text:00000001401511E2 49 89 11                                         mov     [r9], rdx
.text:00000001401511E5 41 FF C0                                         inc     r8d
.text:00000001401511E8 44 89 45 48                                      mov     [rbp+48h],
r8d
.text:00000001401511EC 41 83 F8 10                                      cmp     r8d, 10h
.text:00000001401511F0 72 D6                                            jb      short
loc_1401511C8
```

```
65.  .text:00000001401511F2 49 83 C1 08                                  add     r9, 8
66.  .text:00000001401511F6 4C 89 4D 50                                  mov     [rbp+50h],
     r9
67.  .text:00000001401511FA 41 FF C3                                     inc     r11d
68.  .text:00000001401511FD 44 89 5D 40                                  mov     [rbp+40h],
     r11d
69.  .text:0000000140151201 41 83 FB 19                                  cmp     r11d, 19h
70.  .text:0000000140151205 0F 82 4A FF FF FF                            jb
      loc_140151155
71.  .text:000000014015120B 48 B9 F5 6F 1B AD 5F 93 44 62                mov     rcx,
     6244935FAD1B6FF5h
72.  .text:0000000140151215 49 8B 02                                     mov     rax, [r10]
73.  .text:0000000140151218 48 33 C1                                     xor     rax, rcx
74.  .text:000000014015121B 48 89 45 28                                  mov     [rbp+28h],
     rax
75.  .text:000000014015121F 48 8B 45 28                                  mov     rax,
     [rbp+28h]
76.  .text:0000000140151223 48 B9 DB 27 2A BC 17 A2 15 6A                mov     rcx,
     6A15A217BC2A27DBh
77.  .text:000000014015122D 48 33 C1                                     xor     rax, rcx
78.  .text:0000000140151230 48 89 45 28                                  mov     [rbp+28h],
     rax
79.  .text:0000000140151234 41 C6 02 2E                                  mov     byte ptr
     [r10], 2Eh
80.  .text:0000000140151238 41 C6 42 01 48                               mov     byte ptr
     [r10+1], 48h
81.  .text:000000014015123D 41 C6 42 02 31                               mov     byte ptr
     [r10+2], 31h
82.  .text:0000000140151242 41 C6 42 03 11                               mov     byte ptr
     [r10+3], 11h
83.  .text:0000000140151247 45 33 C9                                     xor     r9d, r9d
84.  .text:000000014015124A 45 33 C0                                     xor     r8d, r8d
85.  .text:000000014015124D 48 8B 55 28                                  mov     rdx,
     [rbp+28h]
86.  .text:0000000140151251 49 8B CA                                     mov     rcx, r10
87.  .text:0000000140151254 41 FF D2                                     call    r10
88.  .text:0000000140151257 C7 85 58 01 00 00 01 00 00 00                mov     dword ptr
     [rbp+158h], 1
89.  .text:0000000140151261 83 45 20 02                                  add     dword ptr
     [rbp+20h], 2
90.  .text:0000000140151265 48 8D 15 25 08 EB FF                         lea     rdx,
     loc_140001A91
91.  .text:000000014015126C 48 8B 8D 00 01 00 00                         mov     rcx,
     [rbp+100h]
92.  .text:0000000140151273 E8 98 14 FE FF                               call
     _local_unwind
93.  .text:0000000140151278 90                                           nop
94.  .text:0000000140151279
95.  .text:0000000140151279                                  loc_140151279:
           ; CODE XREF: CallPatchGuard_1+15↑j
96.  .text:0000000140151279 8B 45 20                                     mov     eax,
     [rbp+20h]
97.  .text:000000014015127C 83 F8 02                                     cmp     eax, 2
98.  .text:000000014015127F 0F 85 AB 00 00 00                            jnz
        loc_140151330
99.  .text:0000000140151285 48 8B 8D BA 00 00 00                         mov     rcx,
     [rbp+0BAh]
100. .text:000000014015128C 48 89 8D 20 01 00 00                         mov     [rbp+120h],
     rcx
101. .text:0000000140151293 4C 8B 85 B2 00 00 00                         mov     r8,
     [rbp+0B2h]
```

```
.text:000000014015129A 48 8B 85 BA 00 00 00                                  mov     rax,
[rbp+0BAh]
.text:00000001401512A1 48 89 85 D0 00 00 00                                  mov     [rbp+0D0h],
rax
.text:00000001401512A8 48 8B 55 6A                                           mov     rdx,
[rbp+6Ah]
.text:00000001401512AC 49 D3 C8                                              ror     r8, cl
.text:00000001401512AF 8B C8                                                 mov     ecx, eax
.text:00000001401512B1 48 D3 C2                                              rol     rdx, cl
.text:00000001401512B4 4C 8B 52 40                                           mov     r10,
[rdx+40h]
.text:00000001401512B8 4C 89 55 28                                           mov     [rbp+28h],
r10
.text:00000001401512BC 4D 33 D0                                              xor     r10, r8
.text:00000001401512BF 48 B8 00 00 00 00 00 80 FF FF                         mov     rax,
0FFFF800000000000h
.text:00000001401512C9 4C 0B D0                                              or      r10, rax
.text:00000001401512CC 4C 89 95 F8 00 00 00                                  mov     [rbp+0F8h],
r10
.text:00000001401512D3 4D 8B CA                                              mov     r9, r10
.text:00000001401512D6 4C 89 55 50                                           mov     [rbp+50h],
r10
.text:00000001401512DA 41 8B CA                                              mov     ecx, r10d
.text:00000001401512DD 83 E1 3F                                              and     ecx, 3Fh
.text:00000001401512E0 49 8B C2                                              mov     rax, r10
.text:00000001401512E3 48 D3 C8                                              ror     rax, cl
.text:00000001401512E6 48 89 45 28                                           mov     [rbp+28h],
rax
.text:00000001401512EA C7 45 30 09 0A 0C 01                                  mov     dword ptr
[rbp+30h], 10C0A09h
.text:00000001401512F1 C7 45 34 0F 00 05 0E                                  mov     dword ptr
[rbp+34h], 0E05000Fh
.text:00000001401512F8 C7 45 38 04 03 07 0D                                  mov     dword ptr
[rbp+38h], 0D070304h
.text:00000001401512FF C7 45 3C 08 06 02 0B                                  mov     dword ptr
[rbp+3Ch], 0B020608h
.text:0000000140151306 33 D2                                                 xor     edx, edx
.text:0000000140151308 89 55 40                                              mov     [rbp+40h],
edx
.text:000000014015130B 8B C2                                                 mov     eax, edx
.text:000000014015130D 4C 8D 45 30                                           lea     r8,
[rbp+30h]
.text:0000000140151311 4C 03 C0                                              add     r8, rax
.text:0000000140151314
.text:0000000140151314                                                loc_140151314:
      ; CODE XREF: CallPatchGuard_1+1F9↓j
.text:0000000140151314 41 8A 08                                              mov     cl, [r8]
.text:0000000140151317 83 F1 09                                              xor     ecx, 9
.text:000000014015131A 88 4C 15 30                                           mov
   [rbp+rdx+30h], cl
.text:000000014015131E FF C2                                                 inc     edx
.text:0000000140151320 89 55 40                                              mov     [rbp+40h],
edx
.text:0000000140151323 49 FF C0                                              inc     r8
.text:0000000140151326 83 FA 10                                              cmp     edx, 10h
.text:0000000140151329 72 E9                                                 jb      short
loc_140151314
.text:000000014015132B E9 1A FE FF FF                                        jmp
   loc_14015114A
.text:0000000140151330                                                ; ---------------------------------
-----------------------------------------
.text:0000000140151330
```

```
.text:0000000140151330                                          loc_140151330:
      ; CODE XREF: CallPatchGuard_1+14F↑j
.text:0000000140151330 48 8B 85 BA 00 00 00                             mov     rax,
[rbp+0BAh]
.text:0000000140151337 48 89 85 10 01 00 00                             mov     [rbp+110h],
rax
.text:000000014015133E 48 8B 95 B2 00 00 00                             mov     rdx,
[rbp+0B2h]
.text:0000000140151345 8B C8                                            mov     ecx, eax
.text:0000000140151347 48 D3 CA                                         ror     rdx, cl
.text:000000014015134A 8B 02                                            mov     eax, [rdx]
.text:000000014015134C
.text:000000014015134C                                          loc_14015134C:
      ; CODE XREF: CallPatchGuard_1+F↑j
.text:000000014015134C 48 83 C4 28                                      add     rsp, 28h
.text:0000000140151350 5D                                               pop     rbp
.text:0000000140151351 5B                                               pop     rbx
.text:0000000140151352 C3                                               retn
.text:0000000140151352                                          CallPatchGuard_1 endp
```

复制代码

自己实现了出来，代码如下：

```
void DecodeCmpAppendDllSection(ULONG_PTR *pfnCmpAppendDllSection, ULONG_PTR KiWaitNever, ULONG_PTR
KiWaitAlways)
{
auto DynamicFactor = ROR((ULONG_PTR)pfnCmpAppendDllSection,(ULONG)pfnCmpAppendDllSection & 0x3f,64);
for (auto i = 0u;i < 0x19/*0x19 * 0x8 = 0xC8*/;i++)
{
  auto Code = (ULONG)pfnCmpAppendDllSection[i];
  pfnCmpAppendDllSection[i] = BSWAP_64(ROL(pfnCmpAppendDllSection[i] ^ KiWaitNever,(UCHAR)KiWaitNever,64)
^ DynamicFactor) ^ KiWaitAlways;
  pfnCmpAppendDllSection[i] += (i * (ULONG_PTR)pfnCmpAppendDllSection);
  DynamicFactor ^= ROR(((0xc8 - i) * i), ((~Code) & 0x3f), 64);
  DynamicFactor = ROL(DynamicFactor, Code & 0x3f, 64) + (ULONG_PTR)pfnCmpAppendDllSection;
  for (auto n = 0u;n < 16;n++)
  {
   UCHAR Key[16] = { 0x00 ,0x03 ,0x05 ,0x08 ,0x06 ,0x09 ,0x0c ,0x07 ,0x0d ,0x0a ,0x0e ,0x04 ,0x01 ,0x0f
,0x0b ,0x02 };
    pfnCmpAppendDllSection[i] = ROR(Key[pfnCmpAppendDllSection[i] & 0xf] | (pfnCmpAppendDllSection[i] &
0xfffffffffffffff0),4,64);
  }
}
}
```

复制代码

我的思路是找到加密算法，然后在遍历内存的时候，为了效率我只先解密出0x10字节大小的内容，然后套用以前windows7解密方式再次解密出context，修改完成后也要重新加密两次回去。因为修改完最后要加密CmpAppendDllSection函数回去，那么就要找加密算法了，一番查找后，在FsRtlMdlReadCompleteDevEx函数里面找到里加密算法，示例如下：

```


INITKDBG:000000014022AFB1                                       loc_14022AFB1:
         ; CODE XREF: FsRtlMdlReadCompleteDevEx+AFA1↑j
INITKDBG:000000014022AFB1 B8 03 00 00 00                                mov     eax, 3
        ; 加密
INITKDBG:000000014022AFB6 88 9D 1B 01 00 00                             mov
   [rbp+11Bh], bl
```

```
06.  INITKDBG:0000000014022AFBC 88 85 20 01 00 00                                    mov
       [rbp+120h], al
07.  INITKDBG:0000000014022AFC2 41 8B C9                                             mov     ecx, r9d
08.  INITKDBG:0000000014022AFC5 B8 0C 00 00 00                                       mov     eax, 0Ch
09.  INITKDBG:0000000014022AFCA 44 88 A5 18 01 00 00                                 mov
       [rbp+118h], r12b
10.  INITKDBG:0000000014022AFD1 88 85 19 01 00 00                                    mov
       [rbp+119h], al
11.  INITKDBG:0000000014022AFD7 4D 8B D1                                             mov     r10, r9
12.  INITKDBG:0000000014022AFDA B8 0F 00 00 00                                       mov     eax, 0Fh
13.  INITKDBG:0000000014022AFDF 44 88 AD 1E 01 00 00                                 mov
       [rbp+11Eh], r13b
14.  INITKDBG:0000000014022AFE6 4C 8B 6D 00                                          mov     r13,
     [rbp+0]
15.  INITKDBG:0000000014022AFEA 41 8B DC                                             mov     ebx,
     r12d
16.  INITKDBG:0000000014022AFED 4D 8B FC                                             mov     r15, r12
17.  INITKDBG:0000000014022AFF0 C6 85 1D 01 00 00 02                                 mov     byte ptr
     [rbp+11Dh], 2
18.  INITKDBG:0000000014022AFF7 44 8B A5 C0 00 00 00                                 mov     r12d,
     [rbp+0C0h]
19.  INITKDBG:0000000014022AFFE 4D 8B D9                                             mov     r11, r9
20.  INITKDBG:0000000014022B001 8D 70 01                                             lea     esi,
     [rax+1]
21.  INITKDBG:0000000014022B004 C6 85 21 01 00 00 05                                 mov     byte ptr
     [rbp+121h], 5
22.  INITKDBG:0000000014022B00B C6 85 24 01 00 00 06                                 mov     byte ptr
     [rbp+124h], 6
23.  INITKDBG:0000000014022B012 C6 85 1F 01 00 00 07                                 mov     byte ptr
     [rbp+11Fh], 7
24.  INITKDBG:0000000014022B019 C6 85 25 01 00 00 08                                 mov     byte ptr
     [rbp+125h], 8
25.  INITKDBG:0000000014022B020 C6 85 22 01 00 00 09                                 mov     byte ptr
     [rbp+122h], 9
26.  INITKDBG:0000000014022B027 C6 85 26 01 00 00 0A                                 mov     byte ptr
     [rbp+126h], 0Ah
27.  INITKDBG:0000000014022B02E C6 85 1C 01 00 00 0B                                 mov     byte ptr
     [rbp+11Ch], 0Bh
28.  INITKDBG:0000000014022B035 C6 85 27 01 00 00 0D                                 mov     byte ptr
     [rbp+127h], 0Dh
29.  INITKDBG:0000000014022B03C C6 85 23 01 00 00 0E                                 mov     byte ptr
     [rbp+123h], 0Eh
30.  INITKDBG:0000000014022B043 88 85 1A 01 00 00                                    mov
       [rbp+11Ah], al
31.  INITKDBG:0000000014022B049 49 D3 CA                                             ror     r10, cl
32.  INITKDBG:0000000014022B04C
33.  INITKDBG:0000000014022B04C                                          loc_14022B04C:
             ; CODE XREF: FsRtlMdlReadCompleteDevEx+B0D2↓j
34.  INITKDBG:0000000014022B04C 49 8B 13                                             mov     rdx,
     [r11]
35.  INITKDBG:0000000014022B04F BF 01 00 00 00                                       mov     edi, 1
36.  INITKDBG:0000000014022B054 4C 8B C6                                             mov     r8, rsi
37.  INITKDBG:0000000014022B057 8D 77 0E                                             lea     esi,
     [rdi+0Eh]
38.  INITKDBG:0000000014022B05A
39.  INITKDBG:0000000014022B05A                                          loc_14022B05A:
             ; CODE XREF: FsRtlMdlReadCompleteDevEx+B07A↓j
40.  INITKDBG:0000000014022B05A 41 0F B6 03                                          movzx   eax,
     byte ptr [r11]
41.  INITKDBG:0000000014022B05E 48 83 E2 F0                                          and     rdx,
     0FFFFFFFFFFFFFFF0h
42.  INITKDBG:0000000014022B062 48 23 C6                                             and     rax, rsi
```

```
INITKDBG:0000000014022B065 0F B6 8C 05 18 01 00 00                    movzx   ecx,
byte ptr [rbp+rax+118h]
INITKDBG:0000000014022B06D 48 0B D1                                   or      rdx, rcx
INITKDBG:0000000014022B070 48 C1 CA 04                                ror     rdx, 4
INITKDBG:0000000014022B074 49 89 13                                   mov     [r11],
rdx
INITKDBG:0000000014022B077 4C 2B C7                                   sub     r8, rdi
INITKDBG:0000000014022B07A 75 DE                                      jnz     short
loc_14022B05A
INITKDBG:0000000014022B07C 48 8B 7D 40                                mov     rdi,
[rbp+40h]  ; KiWaitNever
INITKDBG:0000000014022B080 49 2B D7                                   sub     rdx, r15
INITKDBG:0000000014022B083 49 89 13                                   mov     [r11],
rdx
INITKDBG:0000000014022B086 45 85 E4                                   test    r12d,
r12d
INITKDBG:0000000014022B089 75 13                                      jnz     short
loc_14022B09E
INITKDBG:0000000014022B08B 49 33 D5                                   xor     rdx,
r13        ; KiWaitAlways
INITKDBG:0000000014022B08E 8B CF                                      mov     ecx, edi
INITKDBG:0000000014022B090 48 0F CA                                   bswap   rdx
INITKDBG:0000000014022B093 49 33 D2                                   xor     rdx, r10
INITKDBG:0000000014022B096 48 D3 CA                                   ror     rdx, cl
INITKDBG:0000000014022B099 48 33 D7                                   xor     rdx, rdi
INITKDBG:0000000014022B09C EB 03                                      jmp     short
loc_14022B0A1
INITKDBG:0000000014022B09E                                    ; ------------------------------
-------------------------------------------
INITKDBG:0000000014022B09E
INITKDBG:0000000014022B09E                                    loc_14022B09E:
         ; CODE XREF: FsRtlMdlReadCompleteDevEx+B089↑j
INITKDBG:0000000014022B09E 49 33 D2                                   xor     rdx, r10
INITKDBG:0000000014022B0A1
INITKDBG:0000000014022B0A1                                    loc_14022B0A1:
         ; CODE XREF: FsRtlMdlReadCompleteDevEx+B09C↑j
INITKDBG:0000000014022B0A1 49 89 13                                   mov     [r11],
rdx
INITKDBG:0000000014022B0A4 8B CA                                      mov     ecx, edx
INITKDBG:0000000014022B0A6 BA C8 00 00 00                             mov     edx,
0C8h
INITKDBG:0000000014022B0AB F7 D1                                      not     ecx
INITKDBG:0000000014022B0AD 2B D3                                      sub     edx, ebx
INITKDBG:0000000014022B0AF 4D 03 F9                                   add     r15, r9
INITKDBG:0000000014022B0B2 0F AF D3                                   imul    edx, ebx
INITKDBG:0000000014022B0B5 BE 10 00 00 00                             mov     esi, 10h
INITKDBG:0000000014022B0BA FF C3                                      inc     ebx
INITKDBG:0000000014022B0BC 48 D3 CA                                   ror     rdx, cl
INITKDBG:0000000014022B0BF 41 8B 0B                                   mov     ecx,
[r11]
INITKDBG:0000000014022B0C2 4C 33 D2                                   xor     r10, rdx
INITKDBG:0000000014022B0C5 49 D3 C2                                   rol     r10, cl
INITKDBG:0000000014022B0C8 49 83 C3 08                                add     r11, 8
INITKDBG:0000000014022B0CC 4D 03 D1                                   add     r10, r9
INITKDBG:0000000014022B0CF 83 FB 19                                   cmp     ebx, 19h
INITKDBG:0000000014022B0D2 0F 82 74 FF FF FF                          jb
 loc_14022B04C
INITKDBG:0000000014022B0D8 48 8B 75 38                                mov     rsi,
[rbp+38h]
INITKDBG:0000000014022B0DC 48 C7 C7 00 74 79 B8                       mov     rdi,
0FFFFFFFFB8797400h
```

```
86.  INITKDBG:0000000014022B0E3 4C 8B 75 18                                    mov     r14,
     [rbp+18h]
87.  INITKDBG:0000000014022B0E7 BB 01 00 00 00                                 mov     ebx, 1
88.  INITKDBG:0000000014022B0EC
89.  INITKDBG:0000000014022B0EC                                          loc_14022B0EC:
                ; CODE XREF: FsRtlMdlReadCompleteDevEx+AF15↑j
90.
     复制代码
```

等我写出加密算法后发现有点儿不对了，果然一实验真不对，不好的预感来了，去FsRtlMdlReadCompleteDevEx一翻，加密算法有很多种，大同小异。然后再去翻解密算法，同样有十几种，之前看到以为都是一样的，但是细心一看都有些区别，崩溃...
那么想通过双解密后修改然后在双加密的方式来绕过windows10的pg就不好办了，我总不能去研究十几种加解密算法吧，而且哪种加密对应哪种解密还要慢慢实验，最郁闷的是解密的时候我要去试探十几种算法对效率也产生了很大困扰。于是此路不通了~
于是我再找办法，既然解密CmpAppendDllSection如此麻烦，那我就不解密CmpAppendDllSection函数。而是解密后面的内容。CmpAppendDllSection+0xc8以后的内容都是原来的加解密方法，那么能不能跳过CmpAppendDllSection函数而解密后面的内容呢？答案是能！但是有个问题是解密后面的内容每8字节用到的key不一样，key的动态计算方法在上面代码给出了，是下面这部分代码：

```
01.
02.  auto TempSize = FollowContextSize;    //context的尺寸
03.  auto FollowContextKey = ContextKey;
04.  //解密剩下的部分
05.  do {
06.    pTempMem[(0xC0 / sizeof(ULONG_PTR)) + TempSize] ^= FollowContextKey;
07.    auto RorBit = static_cast<UCHAR>(TempSize);
08.    FollowContextKey = ROR(FollowContextKey, RorBit, 64);
09.  } while (--TempSize);
10.
     复制代码
```

想解密出CmpAppendDllSection之后的代码必须要有一个关键东西，就是context的尺寸，因为以这个尺寸为计算动态key的因子。这就麻烦了，context的大小我没办法得到，因为都解密不出来怎么得到呢。当然我可以用其他办法解密出来context的大小，但是程序中就要写死了，写死的话就不行了，因为context大小是随机的！那有没有不用context大小就能解密出剩下的部分呢，我想了半天虽然前后两次解密有联系，但是这种联系是不牢固的，大脑不够用不想去想了...
上面已经有两种办法不好使了，我比较追求好用的办法，于是继续研究。下面贴出windows10 pg执行的流程：

```
fffff801`23ad7bb8 fffff801`21f78cc4 : 00000000`00000000 00000000`00000000 00000000`00000000 fffff801`23ad8338 : 0xffffe000`b0cae074
fffff801`23ad7bc0 fffff801`21f55f6f : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt! ?? ::FNODOBFM::`string'+0x34c4
fffff801`23ad7bf0 fffff801`21f6cc5d : 00000000`00000000 fffff801`23ad7d90 fffff801`221a3b00 fffff801`23ad7d90 : nt!_C_specific_handler+0x9f
fffff801`23ad7c60 fffff801`21e8e150 : 00000000`00000000 fffff801`21e1a000 fffff801`220dfa00 00000000`00000000 : nt!RtlpExecuteHandlerForException+0xd
fffff801`23ad7c90 fffff801`21e8ad78 : fffff801`23ad8bb8 fffff801`23ad88d0 fffff801`23ad8bb8 00000000`00000001 : nt!RtlDispatchException+0x4e8
fffff801`23ad83a0 fffff801`21f71682 : 00000000`00000002 ffffe000`b1c8a310 00000188`a8c0e914 00000000`00000000 : nt!KiDispatchException+0x144
fffff801`23ad8a80 fffff801`21f6fb7e : fffff801`23ad8cc9 00000000`00000000 ffffe000`b3168010 00000000`00000001 : nt!KiExceptionDispatch+0xc2
fffff801`23ad8c60 fffff801`21f789cc : 00000000`00000011 ffffe000`b1dd8030 ffffe000`b1dd8030 ffffe000`b2485260 : nt!KiGeneralProtectionFault+0xfe
fffff801`23ad8df0 fffff801`21f5605e : ffffe000`b1dd8030 ffffe000`b1f0a4c0 00000000`00000001 ffffe000`00000000 : nt! ?? ::FNODOBFM::`string'+0x31cc
fffff801`23ad8e20 fffff801`21f6ccdd : fffff801`23adb000 fffff801`23ad8fc0 00000000`00000000 fffff801`23ad5000 : nt!_C_specific_handler+0x18e
fffff801`23ad8e90 fffff801`21e8ea8d : fffff801`23adb000 fffff801`23ad9c20 fffff801`23ad5000 00000000`00000000 : nt!RtlpExecuteHandlerForUnwind+0xd
fffff801`23ad8ec0 fffff801`21f55fb2 : fffff801`220fd2c4 fffff801`00000001 fffff801`23ada860 00000000`00000000 : nt!RtlUnwindEx+0x4dd
fffff801`23ad95a0 fffff801`21f6cc5d : 00000000`00000000 fffff801`23ad9740 fffff801`2219e600 fffff801`9bfb9300 : nt!_C_specific_handler+0xe2
fffff801`23ad9610 fffff801`21e8e150 : 00000000`00000000 fffff801`221a3c00 fffff801`23ada568 fffff801`21e963d6 : nt!RtlpExecuteHandlerForException+0xd
fffff801`23ad9640 fffff801`21e8ad78 : fffff801`23ada568 fffff801`23ada280 fffff801`23ada568 fffe000`00000001 : nt!RtlDispatchException+0x4e8
fffff801`23ad9d50 fffff801`21f71682 : fffff801`00000801 00000000`00001810 00000001`00000001 fffff801`00000000 : nt!KiDispatchException+0x144
fffff801`23ada430 fffff801`21f6fb7e : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : nt!KiExceptionDispatch+0xc2
fffff801`23ada610 fffff801`21f6e68d : fffff801`23ada954 fffff801`21ebafaf fffff801`23ada7d0 fffff801`23ada950 : nt!KiGeneralProtectionFault+0xfe
fffff801`23ada7a0 fffff801`21f6e5cd : 00000000`00025b9b fffff801`21ea3a70 00000004`c4f38f91 fffff801`2267bff1 : nt!KiCustomRecurseRoutine9+0xd
fffff801`23ada7d0 fffff801`21f6e50d : 00000000`00000000 fffff801`2217d180 ffffffff`ffffffd1 fffff801`22680d7f : nt!KiCustomRecurseRoutine8+0xd
fffff801`23ada800 fffff801`21f6e542 : fffff801`23ada848 00000000`00000019 00000001`96646e9d fffff801`2266d372 : nt!KiCustomRecurseRoutine7+0xd
fffff801`23ada830 fffff801`21f22b83 : 00000000`00000000 fffff801`2217d180 00000000`00000000 fffff801`21e26d5e : nt!KiCustomAccessRoutine7+0x22
fffff801`23ada860 fffff801`21e6925a : ffffe000`b045d5be fffff801`23adab10 00000000`00000001 00000000`00000002 : nt!ExpTimeRefreshDpcRoutine+0x83
fffff801`23adaa10 fffff801`21f6997a : fffff801`2217d180 fffff801`2217d180 fffff801`221f3740 ffffe000`b2c9e400 : nt!KiRetireDpcList+0x5a
fffff801`23adac60 00000000`00000000 : fffff801`23adb000 fffff801`23ad5000 00000000`00000000 00000000`00000000 : nt!KiIdleLoop+0x9a
```


上图中，0xffffe000`b0cae074就是CmpAppendDllSection地址。我试了很多次，因为我这个windows10版本比较底，发现它都是dpc例程触发的一个异常，然后异常中执行pg解密代码再运行，ExQueueWorkItem没触发过pg(难道我打开方式不对？？！)。所以针对我的这个版本，就好处理了。我们可以在KiExceptionDispatch函数这里拦截下，发现是KiCustomRecurseRoutine(0~9)函数内触发的异常，就跳过这个异常不让异常分发下去，这样pg就没有执行机会了。当然也没必要说一定拦截KiExceptionDispatch，具体拦截什么可以看图中调用流程，只要先于pg代码执行之前就行。
上面的三种办法，就第三种目前来说比较靠谱点儿，但是能不能兼容所有版本问题也很大，要是自己玩玩想更加了解pg运行原理，可以拿这种方法开刀。

那么难道就没比较稳定的方法吗？答案是有的，我在弄第一种办法之前就想过一个办法，实验后发现很稳定的过掉了pg(狂喜 )。不过为了解密出它有很高的荣誉感所以才去研究第一种办法，结果泪奔  。为了讨论才诞生出第二种第

三种办法，同样泪奔...

上面是对windows10 pg的讨论，小伙伴们有什么办法也可以提出来交流一下。

windows10, patchguard

本主题由 mengwuji 于 3 天前 设置高亮

我一定是见鬼了！

万剑归宗

升级 22%

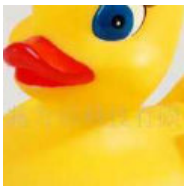
1	0	11
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 沙发

梦老大给力，友情点赞

394999482



升级 8%

0	0	4
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 板凳

有志者事竟成

hzqst

发表于 3 天前 | 只看该作者 地板

讲道理如果全是dpc触发异常的话直接撸idt自己处理一下异常应该可以

升级 2.33%

25	2	207
主题	精华	积分

发消息

点评 回复 支持 反对 举报

璀璨刀光

升级 26.67%

2	0	90
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 5#

支持支持，谢谢梦大分享

点评 回复 支持 反对 举报

mengwuji

升级 100%

220	66	1478
主题	精华	积分

发消息

楼主 | 发表于 3 天前 | 只看该作者 6#

> hzqst 发表于 2017-11-9 09:40
> 讲道理如果全是dpc触发异常的话直接撸idt自己处理一下异常应该可以

我这个版本全部dpc触发的，但是windows7都没这么简单windows10不会这么菜，后来听说是版本的问题，高点儿版本修复了这个问题。

我一定是见鬼了！

点评 回复 支持 反对 举报

空白即是正义

升级 14.67%

18	0	244
主题	精华	积分

发表于 3 天前 | 只看该作者 7#

厉害了word梦

发消息

点评 回复 支持 反对 举报

Chameleon

升级 21.33%

8	0	82
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 8#

支持梦老大，PG还没研究过 

点评 回复 支持 反对 举报

killvxk

升级 8.33%

1	0	225
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 9#

本帖最后由 killvxk 于 2017-11-9 12:16 编辑

过PG目前看来hook SwapContext怼Workitem还算靠谱，其实就是swapcontext的时候识别线程代码是否FsRtlMdlReadCompleteDevEx...
WorkItem是在CmpAppendDllSection设置的。
PG流程：DpcTimer或者SEH或者其他方式触发CmpAppendDllSection，解密后先执行PgSelfVerifyRoutine验证PGContext自身（这里蓝屏的话也是0x109），
验证成功后，执行ExQueueWorkItem塞入一个WorkItem，WorkItem里执行真正的PG部分（验证ntos完整性等等都在这里）。

点评 回复 支持 反对 举报

mengwuji

升级 100%

220	66	1478
主题	精华	积分

发消息

楼主 | 发表于 3 天前 | 只看该作者 10#

> killvxk 发表于 2017-11-9 12:10
> 过PG目前看来hook SwapContext怼Workitem还算靠谱，其实就是swapcontext的时候识别线程代码是否FsRtlMdlRea ...

hook SwapContext成功了吗，如何识别pg代码呢，没执行前是密文没法识别，执行后再去识别也来不及了...
WorkItem不只是在CmpAppendDllSection设置的，其他地方也有设置。
触发流程目前我这个版本上只有dpc触发异常，异常handler触发pg；我要下载个高版本的看看有多少种触发方式。

我一定是见鬼了！

点评 回复 支持 反对 举报

killvxk

发表于 3 天前 | 只看该作者 11#

本帖最后由 killvxk 于 2017-11-9 13:40 编辑

升级 8.33%

1	0	225
主题	精华	积分

发消息

> mengwuji 发表于 2017-11-9 13:06
> hook SwapContext成功了吗，如何识别pg代码呢，没执行前是密文没法识别，执行后再去识别也来不及了...
> W ...

执行后识别来的及，识别的是FsRtlMdlReadCompleteDevEx。特征码
FA 0F 01 8D并且Rip不在模块。

处理很简单，直接修改Rip到自己的代码，然后自己的代码强行jmp去线程RoutineStart：

```
PCHAR CurrentThread = (PCHAR)PsGetCurrentThread();
PVOID StartRoutine = *(PVOID **)(CurrentThread + g_ThreadContextRoutineOffset);

PCHAR StackPage = (PCHAR)IoGetInitialStack();
*(ULONG64 *)StackPage = (((ULONG_PTR)StackPage + 0x1000) & 0x0FFFFFFFFFFFF000);//stack起始的MagicCode，

AdjustStackCallPointer(
                        (ULONG_PTR)StackPointer - 0x8,
                        StartRoutine,
                        NULL);
```

点评 回复 支持 反对 举报

mengwuji

升级 100%

220	66	1478
主题	精华	积分

发消息

楼主 | 发表于 3 天前 | 只看该作者 12#

> killvxk 发表于 2017-11-9 13:25
> 执行后识别来的及，识别的是FsRtlMdlReadCompleteDevEx。特征码
> FA 0F 01 8D并且Rip不在模块。

嗯，感觉有点儿不安全....

我一定是见鬼了！

点评 回复 支持 反对 举报

hzqst

升级 2.33%

25	2	207
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 13#

> killvxk 发表于 2017-11-9 13:25
> 执行后识别来的及，识别的是FsRtlMdlReadCompleteDevEx。特征码
> FA 0F 01 8D并且Rip不在模块。

又是BackTo1942大法么。。。

点评 回复 支持 反对 举报

killvxk

升级 8.33%

1	0	225
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 14#

> hzqst 发表于 2017-11-9 14:05
> 又是BackTo1942大法么。。。

没back，workitem不用back了，直接去执行线程该执行的东西就行了

点评 回复 支持 反对 举报

killvxk

升级 8.33%

1	0	225
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 15#

> mengwuji 发表于 2017-11-9 13:54
> 嗯，感觉有点儿不安全....

不知道安不安全，我没怎么测，写出来用了一次，后面都没管。

点评 回复 支持 反对 举报

hzqst

升级 2.33%

25	2	207
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 16#

> killvxk 发表于 2017-11-9 14:15
> 没back，workitem不用back了，直接去执行线程该执行的东西就行了

那直接把rip改到一个无限死循环里不是更好

点评 回复 支持 反对 举报

killvxk

升级 8.33%

1	0	225
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 17#

> hzqst 发表于 2017-11-9 14:30
> 那直接把rip改到一个无限死循环里不是更好

workitem的thread可能是有用的

点评 回复 支持 反对 举报

错误代码

升级 20%

3	0	80
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 18#

怼PG

点评 回复 支持 反对 举报

mengwuji

升级 100%

220	66	1478
主题	精华	积分

发消息

楼主 | 发表于 3 天前 | 只看该作者 19#

下午睡觉灵光一现，想到了解密出context的对策，马上实验没想到真的可行，详见http://www.mengwuji.net/thread-7609-1-1.html

我一定是见鬼了！

点评 回复 支持 反对 举报

YOUBADBAD

升级 97.33%

43	0	196
主题	精华	积分

发消息

发表于 3 天前 | 只看该作者 20#

某游戏之前解密一些自己文件的时候就用的这种

点评 回复 支持 反对 举报

下一页 »

发帖 回复

返回列表 1 2 1 / 2 页 下一页

| 上传 高级模式

发表回复 将此回复同步到 | 回帖后跳转到最后一页

本版积分规则