x64内核

- GDT、IDT等一项改为16字节
- fs寄存器改为gs寄存器
- SSDT的函数地址数组元素需要右移4位+数组首地址得API地址
- 存在DSE、KPP
- ...

DSE驱动强制签名

关闭驱动强制签名：

- win7
  - `bcdedit.exe -set loadoptions DDISABLE_INTEGRITY_CHECKS`
- win10
  - `bcdedit.exe /set nointegritychecks on`

KPP保护防止驱动恶意修改

由于KPP保护存在，不能对系统代码进行修改，而windows又提供了其他的方法可进行hook

PsSetXxxNotifyRoutine系列API

windows提供`PsSetXxxNotifyRoutine`系列的API用来hook，例如：

Hook CreateProcess需要使用`PsSetCreateProcessNotifyRoutine`或者`PsSetCreateProcessNotifyRoutineEx`来注册回调函数：

- `PsSetCreateProcessNotifyRoutine`
  - 只能监控
- `PsSetCreateProcessNotifyRoutineEx`
  - 可支持拦截，但必须要求强制签名
  - 在驱动编译时，加链接选项`/INTEGRITYCHECK`，此选项将在PE的可选头的DLL属性中勾选**Code Integrity Image**标志

```
NTSTATUS PsSetCreateProcessNotifyRoutine(
  PCREATE_PROCESS_NOTIFY_ROUTINE NotifyRoutine,     // 回调
  BOOLEAN                        Remove             // 删除(TRUE)，增加(FALSE)
);

NTSTATUS PsSetCreateProcessNotifyRoutineEx(
  PCREATE_PROCESS_NOTIFY_ROUTINE_EX NotifyRoutine,  // 回调
  BOOLEAN                           Remove          // 删除(TRUE)，增加(FALSE)
);
```

`PCREATE_PROCESS_NOTIFY_ROUTINE` 或 `PCREATE_PROCESS_NOTIFY_ROUTINE_EX` 回调声明：

```
PCREATE_PROCESS_NOTIFY_ROUTINE PcreateProcessNotifyRoutine;
void PcreateProcessNotifyRoutine(
  HANDLE ParentId,      // 父进程id
  HANDLE ProcessId,     // 进程id
  BOOLEAN Create        // 指示是否创建了进程(TRUE)或删除了进程(FALSE)
)
{...}


PCREATE_PROCESS_NOTIFY_ROUTINE_EX PcreateProcessNotifyRoutineEx;
void PcreateProcessNotifyRoutineEx(
  PEPROCESS Process,                        // 父进程id
  HANDLE ProcessId,                         // 进程id
  PPS_CREATE_NOTIFY_INFO CreateInfo         // 指向PS_CREATE_NOTIFY_INFO结构的指
针，该结构包含关于新进程的信息
                                            // 如果该参数为空，则指定的进程退出
)
{...}
```

示例

通过hook来禁止打开计算器

```
// hook回调
VOID CreateProcessNotify(PEPROCESS  Process, HANDLE  ProcessId,
PPS_CREATE_NOTIFY_INFO  CreateInfo)
{
    if (CreateInfo != NULL) {
        UNICODE_STRING ustrImageName;
        RtlInitUnicodeString(&ustrImageName, L"\\??
\\C:\\Windows\\system32\\calc.exe");

        if (RtlCompareUnicodeString(&ustrImageName, CreateInfo->ImageFileName,
FALSE) == 0) {
            CreateInfo->CreationStatus = STATUS_UNSUCCESSFUL;
        }
    } else {
        KdPrint(("[51asm] CreateProcessNotify Exit ProcessId:%d\n", ProcessId));
    }

}

// 驱动入口
NTSTATUS DriverEntry(
    __in struct _DRIVER_OBJECT  *DriverObject,
    __in PUNICODE_STRING  RegistryPath
    )
{
    ...
    PsSetCreateProcessNotifyRoutineEx(CreateProcessNotify, FALSE);  // 注册hook回
调函数
    ...
}
```

内部原理

根据WRK的源码：

在注册回调函数的内部会将回调函数放入一个全局数组中（数组大小每个版本不一样，win7支持64项，且每个API都有一个数组）

```
//
// Allocate a new callback block.
//
CallBack = ExAllocateCallBack ((PEX_CALLBACK_FUNCTION) NotifyRoutine, NULL);
if (CallBack == NULL) {
    return STATUS_INSUFFICIENT_RESOURCES;
}

for (i = 0; i < PSP_MAX_CREATE_PROCESS_NOTIFY; i++) {
    //
    // Try and swap a null entry for the new block.
    //
    if (ExCompareExchangeCallBack (&PspCreateProcessNotifyRoutine[i],
                                   CallBack,
                                   NULL)) {
        InterlockedIncrement ((PLONG) &PspCreateProcessNotifyRoutineCount);
        return STATUS_SUCCESS;
    }
}
//
// No slots left. Free the block and return.
//
ExFreeCallBack (CallBack);
return STATUS_INVALID_PARAMETER;
```

在创建进程时，会调用此数组中的回调

```
if (OldActiveThreads == 0) {
    PERFINFO_PROCESS_CREATE (Process);

    if (PspCreateProcessNotifyRoutineCount != 0) {
        ULONG i;
        PEX_CALLBACK_ROUTINE_BLOCK CallBack;
        PCREATE_PROCESS_NOTIFY_ROUTINE Rtn;

        for (i=0; i<PSP_MAX_CREATE_PROCESS_NOTIFY; i++) {
            CallBack = ExReferenceCallBackBlock
(&PspCreateProcessNotifyRoutine[i]);
            if (CallBack != NULL) {
                Rtn = (PCREATE_PROCESS_NOTIFY_ROUTINE) ExGetCallBackBlockRoutine
(CallBack);
                Rtn (Process->InheritedFromUniqueProcessId,
                     Process->UniqueProcessId,
                     TRUE);
                ExDereferenceCallBackBlock (&PspCreateProcessNotifyRoutine[i],
```

```
                                                CallBack);
            }
        }
    }
}
```

注意：这个数组不是直接存放回调函数的地址，是有处理的（解密函数：`ExCompareExchangeCallBack`）

1. win7 32位

   ```
   数组元素 & 0xFFFFFFF8 = 结构体指针
   结构体指针 + 4 = 函数指针
   ```

2. win7 64位

   ```
   数组元素 & 0xFFFFFFFFFFFFFFF0 = 结构体指针
   结构体指针 + 8 = 函数指针
   ```

3. win10 64位 1909

   与win7一致

分析PsSetCreateProcessNotifyRoutineEx寻找相应系统回调函数地址

关于win7

目标平台：win7 64位，内核文件：ntoskrnl.exe

`PsSetCreateProcessNotifyRoutineEx`内部调用了`PsSetCreateProcessNotifyRoutineEx`，传入第3个参数`r8b = 1`

在`PsSetCreateProcessNotifyRoutineEx`开始，判断`remove`参数

```
PAGE:00000001404B91F0 PspSetCreateProcessNotifyRoutine proc near
PAGE:00000001404B91F0                                         ; CODE XREF: PsSetCreateProcessNotifyRoutine+3↓j
PAGE:00000001404B91F0                                         ; PsSetCreateProcessNotifyRoutineEx+3↓j
PAGE:00000001404B91F0                                         ; DATA XREF: ...
PAGE:00000001404B91F0
PAGE:00000001404B91F0 arg_0           = qword ptr  8
PAGE:00000001404B91F0 arg_8           = qword ptr  10h
PAGE:00000001404B91F0 arg_10          = qword ptr  18h
PAGE:00000001404B91F0
PAGE:00000001404B91F0                 mov     [rsp+arg_0], rbx
PAGE:00000001404B91F5                 mov     [rsp+arg_8], rbp
PAGE:00000001404B91FA                 mov     [rsp+arg_10], rsi
PAGE:00000001404B91FF                 push    rdi
PAGE:00000001404B9200                 push    r12
PAGE:00000001404B9202                 push    r13
PAGE:00000001404B9204                 push    r14
PAGE:00000001404B9206                 push    r15
PAGE:00000001404B9208                 sub     rsp, 20h
PAGE:00000001404B920C                 xor     r12d, r12d      ; r12 = 0
PAGE:00000001404B920F                 mov     bpl, r8b        ; bpl是个标志，PsSetCreateProcessNotifyRoutineEx传入为1
PAGE:00000001404B9212                 mov     r13, rcx        ; r13 = 回调函数地址
PAGE:00000001404B9215                 lea     ebx, [r12+1]    ; rbx = 1
PAGE:00000001404B921A                 cmp     dl, r12b        ; 判断remove标志，0为增加
PAGE:00000001404B921D                 jz      loc_1404B9331   ; 增加，跳转
PAGE:00000001404B9223                 mov     rdi, gs:188h
PAGE:00000001404B922C                 or      eax, 0FFFFFFFFh
PAGE:00000001404B922F                 add     [rdi+1C4h], ax
PAGE:00000001404B9236                 lea     r14, PspCreateProcessNotifyRoutine ; r14 = 数组首地址
PAGE:00000001404B923D
```

这里将`remove`视为0，跳转，之后检查回调，分配回调的结构内存

```
PAGE:00000001404B9331 loc_1404B9331:                ; CODE XREF: PspSetCreateProcessNotifyRoutine+2D↑j
PAGE:00000001404B9331             cmp     bpl, r12b
PAGE:00000001404B9334             jz      short loc_1404B934F
PAGE:00000001404B9336             call    MmVerifyCallbackFunction ; 检查回调
PAGE:00000001404B933B             cmp     eax, r12d
PAGE:00000001404B933E             jnz     short loc_1404B934A
PAGE:00000001404B9340             mov     eax, 0C0000022h
PAGE:00000001404B9345             jmp     loc_1404B93DC
PAGE:00000001404B934A ; ---------------------------------------------------------------------------
PAGE:00000001404B934A
PAGE:00000001404B934A loc_1404B934A:                ; CODE XREF: PspSetCreateProcessNotifyRoutine+14E↑j
PAGE:00000001404B934A             mov     rdx, rbx
PAGE:00000001404B934D             jmp     short loc_1404B9352 ; 回调函数地址
PAGE:00000001404B934F ; ---------------------------------------------------------------------------
PAGE:00000001404B934F
PAGE:00000001404B934F loc_1404B934F:                ; CODE XREF: PspSetCreateProcessNotifyRoutine+144↑j
PAGE:00000001404B934F             mov     rdx, r12
PAGE:00000001404B9352
PAGE:00000001404B9352 loc_1404B9352:                ; CODE XREF: PspSetCreateProcessNotifyRoutine+15D↑j
PAGE:00000001404B9352             mov     rcx, r13        ; 回调函数地址
PAGE:00000001404B9355             call    ExAllocateCallBack ; 分配回调的结构内存
PAGE:00000001404B935A             mov     rsi, rax        ; rsi = 结构地址
```

在分配内存中可得知回调结构的组成，并返回此结构的地址

```
PAGE:0000000140473B90 ExAllocateCallBack proc near          ; CODE XREF: PsSetLoadImageNotifyRoutine+C↓p
PAGE:0000000140473B90                                        ; PsSetCreateThreadNotifyRoutine+C↓p ...
PAGE:0000000140473B90
PAGE:0000000140473B90 arg_0           = qword ptr  8
PAGE:0000000140473B90
PAGE:0000000140473B90                 mov     [rsp+arg_0], rbx
PAGE:0000000140473B95                 push    rdi
PAGE:0000000140473B96                 sub     rsp, 20h
PAGE:0000000140473B9A                 mov     rbx, rdx
PAGE:0000000140473B9D                 mov     edx, 18h        ; NumberOfBytes
PAGE:0000000140473BA2                 mov     rdi, rcx
PAGE:0000000140473BA5                 lea     ecx, [rdx-17h]  ; PoolType
PAGE:0000000140473BA8                 mov     r8d, 62726243h  ; Tag
PAGE:0000000140473BAE                 call    ExAllocatePoolWithTag ; 分配内存
PAGE:0000000140473BB3                 test    rax, rax
PAGE:0000000140473BB6                 jz      short loc_140473BC4
PAGE:0000000140473BB8                 and     qword ptr [rax], 0 ; RoutineStruct.unknown1 = 0,   8 bytes
PAGE:0000000140473BBC                 mov     [rax+8], rdi    ; RoutineStruct.unknown2 = rdi, 8 bytes, 回调地址
PAGE:0000000140473BC0                 mov     [rax+10h], rbx  ; RoutineStruct.unknown3 = rbx, 8 bytes
PAGE:0000000140473BC4
PAGE:0000000140473BC4 loc_140473BC4:                          ; CODE XREF: ExAllocateCallBack+26↑j
PAGE:0000000140473BC4                 mov     rbx, [rsp+28h+arg_0]
PAGE:0000000140473BC9                 add     rsp, 20h
PAGE:0000000140473BCD                 pop     rdi
PAGE:0000000140473BCE                 retn
PAGE:0000000140473BCE ; ---------------------------------------------------------------------------
PAGE:0000000140473BCF                 align 20h
PAGE:0000000140473BCF ExAllocateCallBack endp
PAGE:0000000140473BCF
```

参考ReactOs源码，此结构为

```
typedef struct _EX_CALLBACK_ROUTINE_BLOCK {
    EX_RUNDOWN_REF RundownProtect;
    PEX_CALLBACK_FUNCTION Function;
    PVOID Context;
} EX_CALLBACK_ROUTINE_BLOCK, *PEX_CALLBACK_ROUTINE_BLOCK;
```

之后执行交换回调，完成解密工作

```
PAGE:00000001404B9369
PAGE:00000001404B9369 loc_1404B9369:                ; CODE XREF: PspSetCreateProcessNotifyRoutine+170↑j
PAGE:00000001404B9369             mov     edi, r12d
PAGE:00000001404B936C             lea     r14, PspCreateProcessNotifyRoutine ; r14 = 数组首地址
PAGE:00000001404B9373
PAGE:00000001404B9373 loc_1404B9373:                ; CODE XREF: PspSetCreateProcessNotifyRoutine+19E↓j
PAGE:00000001404B9373             mov     eax, edi        ; eax = index
PAGE:00000001404B9375             xor     r8d, r8d
PAGE:00000001404B9378             mov     rdx, rsi        ; rdx = 结构地址
PAGE:00000001404B937B             lea     rcx, [r14+rax*8] ; 数组寻址，第i个元素的地址
PAGE:00000001404B937F             call    ExCompareExchangeCallBack ; 比较交换回调函数
```

在此函数内，动态跟踪，便可找到解密点，如下：

```
PAGE:000000014040A1AB
PAGE:000000014040A1AB loc_14040A1AB:                ; CODE XREF: ExCompareExchangeCallBack+4A↑j
PAGE:000000014040A1AB                               ; ExCompareExchangeCallBack+64↑j
PAGE:000000014040A1AB             mov     rsi, rbx        ; rsi = 数组第i个元素
PAGE:000000014040A1AE             and     rsi, 0FFFFFFFFFFFFFFF0h ; 数组第i个元素 & 0FFFFFFFFFFFFFFF0h，此时动态调试时，就已经得到了回调函数地址
PAGE:000000014040A1B2             cmp     rsi, r11
PAGE:000000014040A1B5             jnz     short loc_14040A230
```

规则为：**数组元素 & 0xFFFFFFFFFFFFFFF0 -> 回调结构地址，回调结构的第二个成员就是回调函数地址**

```
0: kd> u CreateProcessNotify
KMDFDriver1!CreateProcessNotify [q:\tmp\workspace\kmdf_driver1\k
fffff880`03c72350 4c89442418      mov     qword ptr [rsp+18h],r8
fffff880`03c72355 4889542410      mov     qword ptr [rsp+10h],rd
fffff880`03c7235a 48894c2408      mov     qword ptr [rsp+8],rcx
fffff880`03c7235f 4883ec38        sub     rsp,38h
fffff880`03c72363 48837c245000    cmp     qword ptr [rsp+50h],0
fffff880`03c72369 743b            je      KMDFDriver1!CreateProc
fffff880`03c7236b 488d152e010000  lea     rdx,[KMDFDriver1! ?? :
fffff880`03c72372 488d4c2420      lea     rcx,[rsp+20h]
```

```
0: kd> dq fffff800`0407ffa0                                          数组
fffff800`0407ffa0  fffff8a0`0000884f fffff8a0`003033cf
fffff800`0407ffb0  fffff8a0`003033ff fffff8a0`0035117f
fffff800`0407ffc0  fffff8a0`003f0aef fffff8a0`080a25ff
fffff800`0407ffd0  fffff8a0`0178fedf fffff8a0`09e20a4f
fffff800`0407ffe0  00000000`00000000 00000000`00000000
fffff800`0407fff0  00000000`00000000 00000000`00000000
fffff800`04080000  00000000`00000000 00000000`00000000
fffff800`04080010  00000000`00000000 00000000`00000000
0: kd> dq fffff8a0`09e20a4f&FFFFFFFFFFFFFFF0                         回调
fffff8a0`09e20a40  00000000`00000020 fffff880`03c72350
fffff8a0`09e20a50  00000000`00000001 00005053`412e0004
fffff8a0`09e20a60  7346744e`03040303 00000000`00000000
fffff8a0`09e20a70  fffff8a0`081eda10 00000000`00000000
fffff8a0`09e20a80  00000000`00000000 00000000`00000000
fffff8a0`09e20a90  00010000`000067c8 fffff8a0`0340e010
fffff8a0`09e20aa0  73634946`030c0304 00000000`00000000
fffff8a0`09e20ab0  00000000`00000000 fffffa80`192535a0
```

关于win10

同win7一样

```
text:0000000140181C5F                jbe     short loc_140181CA8
text:0000000140181C61
text:0000000140181C61 loc_140181C61:                          ; CODE XREF: ExCompareExchangeCallBack+A7↓j
text:0000000140181C61                mov     rsi, rbx
text:0000000140181C64                mov     rdx, 0FFFFFFFFFFFFFFF0h ; 解密点
text:0000000140181C6B                and     rsi, rdx
text:0000000140181C6E                cmp     rsi, rbp
text:0000000140181C71                jz      short loc_140181CCA
```