Windows system call

- [ ] Learning about windows system call about r3 and r0
- [ ] I want to study these thing use three days

Windows system call (DAY 1

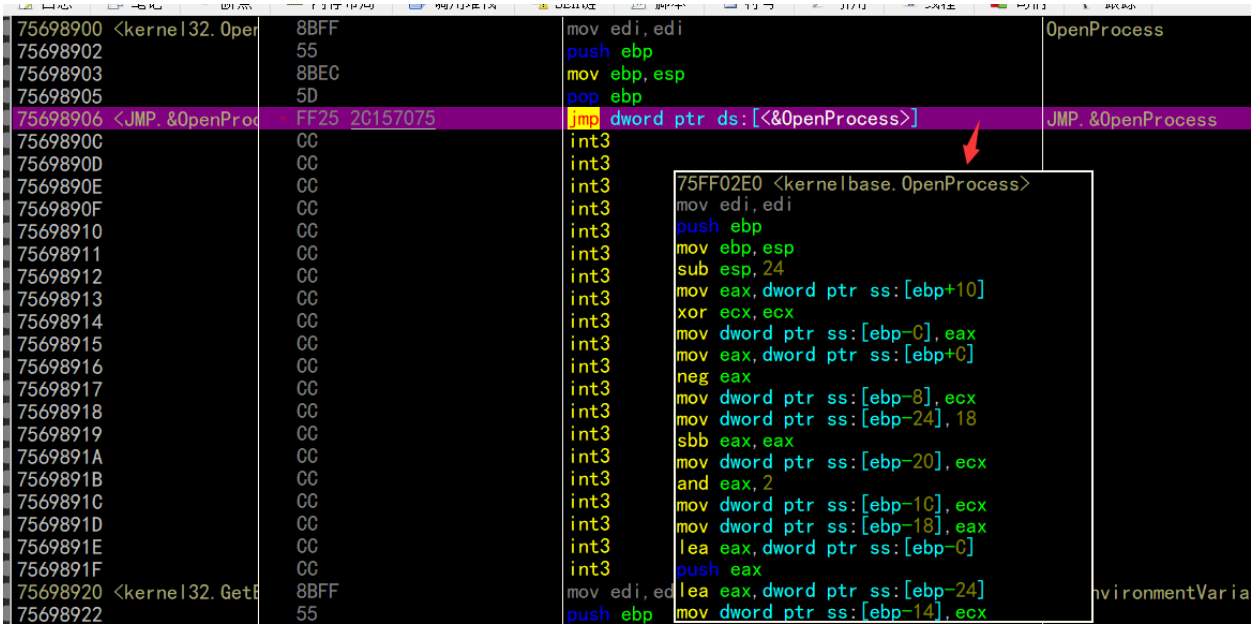
I write code use OpenProcess API to test:

```
#include <stdio.h>
#include <Windows.h>


int main()
{
  _asm int 3;
  OpenProcess(NULL, NULL, NULL);
  return 0;
}
```

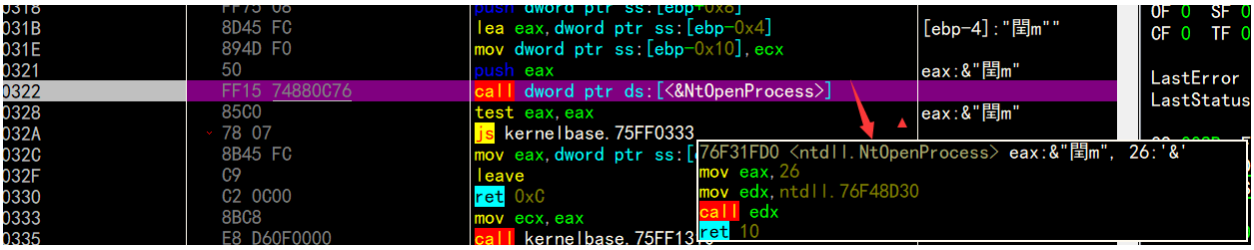
Let's look this Process

- First : OpenProcess → Kernel32.OpenProcess

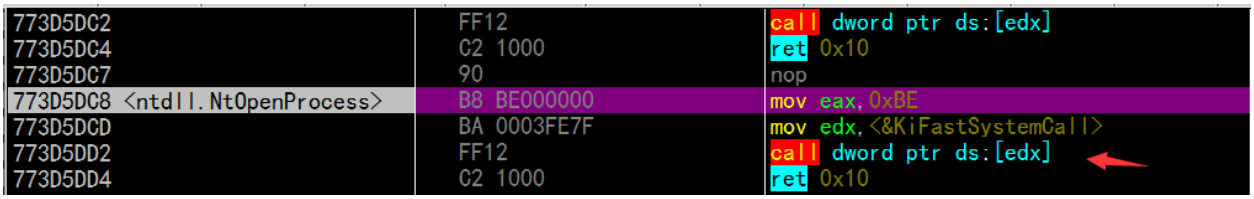
- Second : Kernel32.OpenProcess → Kernelbase.OpenPorcess



- Third : Kernelbase.OpenProcess → Ntdll.NtOpenProcess



- Ntdll.NtOpenProcess → KiFastSystemCall



- Ntdll.KiFastSystemCall → sysenter

So we can know this process : R3.OpenProcess →Kernel32.OpenProcess →Kernelbase.OpenPorcess →Ntdll.NtOpenProcess →KiFastSystemCall

We can find a interesting things that in windows7 all of them must be obey this process.

We can find in windows xp don't have KernelBase.dll but in windows 7 have KernelBase.dll, Why?

Because :

For example : In windows xp OpenProcess API have three parameters , but in windows 7 not satisfied with the previous situation. We should have four parameters. So we need KernelBase.dll to realize the idea. Added a function to maintain compatibility with previous systems. Let's look at top picture, Kernel32.OpenProcess → Kernelbase.OpenPorcess (there have a code jmp kernelbase.xxx to realize the idea)

Analyze use ida:

We should know OpenProcess API

Parameters

`[in] dwDesiredAccess`

The access to the process object. This access right is checked against the security descriptor for the process. This parameter can be one or more of the process access rights.

If the caller has enabled the SeDebugPrivilege privilege, the requested access is granted regardless of the contents of the security descriptor.

`[in] bInheritHandle`

If this value is TRUE, processes created by this process will inherit the handle. Otherwise, the processes do not inherit this handle.

`[in] dwProcessId`

The identifier of the local process to be opened.

If the specified process is the System Idle Process (0x00000000), the function fails and the last error code is `ERROR_INVALID_PARAMETER`. If the specified process is the System process or one of the Client Server Run-Time Subsystem (CSRSS) processes, this function fails and the last error code is `ERROR_ACCESS_DENIED` because their access restrictions prevent user-level code from opening them.

If you are using GetCurrentProcessId as an argument to this function, consider using GetCurrentProcess instead of OpenProcess, for improved performance.

- Kernel32.OpenProcess

```
.text:77E254E7                 mov     edi, edi
.text:77E254E9                 push    ebp
.text:77E254EA                 mov     ebp, esp
.text:77E254EC                 pop     ebp
.text:77E254ED                 jmp     short _OpenProcess@12 ; OpenProcess(x,x,x)
.text:77E254ED _OpenProcessStub@12 endp
.text:77E254ED
.text:77E254ED ; ---------------------------------------------------------------------------
.text:77E254EF                 db 5 dup(90h)
.text:77E254F4
.text:77E254F4 ; =============== S U B R O U T I N E =======================================
.text:77E254F4
.text:77E254F4 ; Attributes: thunk
.text:77E254F4
.text:77E254F4 ; HANDLE __stdcall OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId)
.text:77E254F4 _OpenProcess@12 proc near               ; CODE XREF: OpenProcessStub(x,x,x)+6↑j
.text:77E254F4                                         ; Toolhelp32ReadProcessMemory(x,x,x,x,x)+D↓p
.text:77E254F4
.text:77E254F4 dwDesiredAccess = dword ptr  4
.text:77E254F4 bInheritHandle  = dword ptr  8
.text:77E254F4 dwProcessId     = dword ptr  0Ch
.text:77E254F4
.text:77E254F4                 jmp     ds:__imp__OpenProcess@12 ; OpenProcess(x,x,x)
```

Kernel32.Openprocess

- KernelBase.OpenProcess

```
mov     edi, edi
push    ebp
mov     ebp, esp
sub     esp, 20h
mov     eax, [ebp+dwProcessId] ; Pid参数获取
mov     [ebp+ClientId.UniqueProcess], eax ; ClientID是一个结构体 里面存放了UniqueProcess, UniqueThread, 将Pid获取后存放到UniqueProcess
mov     eax, [ebp+bInheritHandle] ; eax = 继承句柄值
push    esi
xor     esi, esi
neg     eax             ; 值进行取反+1
sbb     eax, eax        ; 算eflag进位减(c 位)
and     eax, 2          ; 因为继承句柄的开关 只有True和flase 所以0的时候还是0 1的时候为ffffffff 所以2为true的开关值
mov     [ebp+ObjectAttributes.Attributes], eax ; 存放到属性值内
lea     eax, [ebp+ClientId]
push    eax             ; ClientId
lea     eax, [ebp+ObjectAttributes]
push    eax             ; ObjectAttributes
push    [ebp+dwDesiredAccess] ; DesiredAccess
lea     eax, [ebp+dwProcessId]
push    eax             ; ProcessHandle
mov     [ebp+ClientId.UniqueThread], esi ; 因为是通过pid来执行的功能所以这些的项都没有什么作用直接给0即可
mov     [ebp+ObjectAttributes.Length], 18h
mov     [ebp+ObjectAttributes.RootDirectory], esi
mov     [ebp+ObjectAttributes.ObjectName], esi
mov     [ebp+ObjectAttributes.SecurityDescriptor], esi
mov     [ebp+ObjectAttributes.SecurityQualityOfService], esi
call    ds:NtOpenProcess ; call ntdll.openprocess
cmp     eax, esi        ; 因为内核里面的 >=0 是成功的判断从内核执行的操作是否是成功的
pop     esi
jl      loc_DCE7554     ; 如果小于0 就执行错误的分支
```

This code is LastError it can get R0 error change R3 error, so this error number don't r0 error number

```
.text:0DCE6BA5 Status          = dword ptr  8
.text:0DCE6BA5
.text:0DCE6BA5                 mov     edi, edi
.text:0DCE6BA7                 push    ebp
.text:0DCE6BA8                 mov     ebp, esp
.text:0DCE6BAA                 push    esi
.text:0DCE6BAB                 push    [ebp+Status]    ; Status
.text:0DCE6BAE                 call    ds:RtlNtStatusToDosError
.text:0DCE6BB4                 mov     esi, eax
.text:0DCE6BB6                 push    esi             ; LastError
.text:0DCE6BB7                 call    ds:RtlSetLastWin32Error
.text:0DCE6BBD                 mov     eax, esi
.text:0DCE6BBF                 pop     esi
.text:0DCE6BC0                 pop     ebp
.text:0DCE6BC1                 retn    4
.text:0DCE6BC1 sub_DCE6BA5     endp
```

- Ntdll.OpenProcess

```
.text:77F05DD8                 public ZwOpenProcessToken
.text:77F05DD8 ZwOpenProcessToken proc near            ; CODE XREF: RtlCreateUserSecurityObject+36↑p
.text:77F05DD8                                         ; RtlAdjustPrivilege+2A↑p ...
.text:77F05DD8                 mov     eax, 0BFh       ; NtOpenProcessToken
.text:77F05DDD                 mov     edx, 7FFE0300h
.text:77F05DE2                 call    dword ptr [edx]
.text:77F05DE4                 retn    0Ch
.text:77F05DE4 ZwOpenProcessToken endp
```

服务号：0xBF
KiFastCallEntry: 0x7FFE0300

We should know a struct (_KUSER_SHARED_DATA)

```
1: kd> dt _KUSER_SHARED_DATA
ntdll!_KUSER_SHARED_DATA
   +0x000 TickCountLowDeprecated : Uint4B
   +0x004 TickCountMultiplier : Uint4B
   +0x008 InterruptTime    : _KSYSTEM_TIME
   +0x014 SystemTime       : _KSYSTEM_TIME
   +0x020 TimeZoneBias     : _KSYSTEM_TIME
   +0x02c ImageNumberLow   : Uint2B
   +0x02e ImageNumberHigh  : Uint2B
   +0x030 NtSystemRoot     : [260] Wchar
   +0x238 MaxStackTraceDepth : Uint4B
   +0x23c CryptoExponent   : Uint4B
   +0x240 TimeZoneId       : Uint4B
   +0x244 LargePageMinimum : Uint4B
   +0x248 Reserved2        : [7] Uint4B
   +0x264 NtProductType    : _NT_PRODUCT_TYPE
   +0x268 ProductTypeIsValid : UChar
   +0x26c NtMajorVersion   : Uint4B
   +0x270 NtMinorVersion   : Uint4B
   +0x274 ProcessorFeatures : [64] UChar
   +0x2b4 Reserved1        : Uint4B
   +0x2b8 Reserved3        : Uint4B
   +0x2bc TimeSlip         : Uint4B
   +0x2c0 AlternativeArchitecture : _ALTERNATIVE_ARCHITECTURE_TYPE
   +0x2c4 AltArchitecturePad : [1] Uint4B
   +0x2c8 SystemExpirationDate : _LARGE_INTEGER
   +0x2d0 SuiteMask        : Uint4B
   +0x2d4 KdDebuggerEnabled : UChar
   +0x2d5 NXSupportPolicy  : UChar
   +0x2d8 ActiveConsoleId  : Uint4B
   +0x2dc DismountCount    : Uint4B
   +0x2e0 ComPlusPackage   : Uint4B
   +0x2e4 LastSystemRITEventTickCount : Uint4B
   +0x2e8 NumberOfPhysicalPages : Uint4B
   +0x2ec SafeBootMode     : UChar
   +0x2ed TscQpcData       : UChar
   +0x2ed TscQpcEnabled    : Pos 0, 1 Bit
   +0x2ed TscQpcSpareFlag  : Pos 1, 1 Bit
   +0x2ed TscQpcShift      : Pos 2, 6 Bits
   +0x2ee TscQpcPad        : [2] UChar
   +0x2f0 SharedDataFlags  : Uint4B
   +0x2f0 DbgErrorPortPresent : Pos 0, 1 Bit
   +0x2f0 DbgElevationEnabled : Pos 1, 1 Bit
   +0x2f0 DbgVirtEnabled   : Pos 2, 1 Bit
   +0x2f0 DbgInstallerDetectEnabled : Pos 3, 1 Bit
   +0x2f0 DbgSystemDllRelocated : Pos 4, 1 Bit
   +0x2f0 DbgDynProcessorEnabled : Pos 5, 1 Bit
   +0x2f0 DbgSEHValidationEnabled : Pos 6, 1 Bit
   +0x2f0 SpareBits        : Pos 7, 25 Bits
   +0x2f4 DataFlagsPad     : [1] Uint4B
   +0x2f8 TestRetInstruction : Uint8B
   +0x300 SystemCall       : Uint4B
   +0x304 SystemCallReturn : Uint4B
   +0x308 SystemCallPad    : [3] Uint8B
   +0x320 TickCount        :  KSYSTEM TIME
```

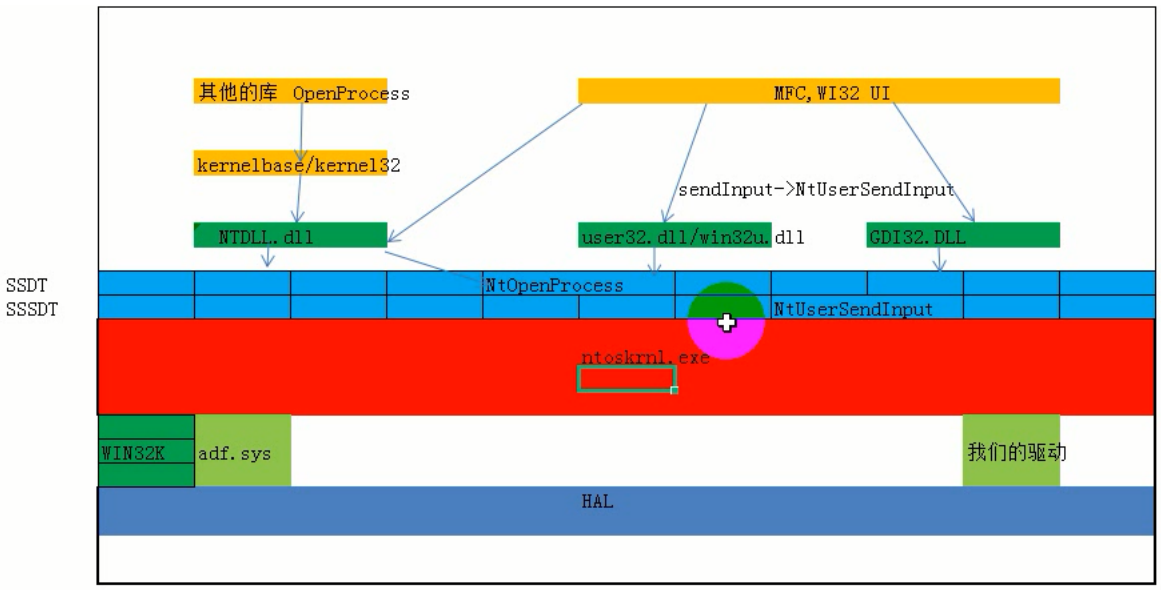
Let's look at 0x7ffe0000, this r3 address → r0 address (0xffdf0000) this struct equivalent to a global variable, it exists in each process(share physics page)

```
1: kd> dt _KUSER_SHARED_DATA 0x7ffe0000
ntdll!_KUSER_SHARED_DATA
   +0x000 TickCountLowDeprecated : 0
   +0x004 TickCountMultiplier : 0xfa00000
   +0x008 InterruptTime     : _KSYSTEM_TIME
   +0x014 SystemTime        : _KSYSTEM_TIME
   +0x020 TimeZoneBias      : _KSYSTEM_TIME
   +0x02c ImageNumberLow    : 0x14c
   +0x02e ImageNumberHigh   : 0x14c
   +0x030 NtSystemRoot      : [260]  "C:\Windows"
   +0x238 MaxStackTraceDepth : 0
   +0x23c CryptoExponent    : 0
   +0x240 TimeZoneId        : 0
   +0x244 LargePageMinimum  : 0x200000
   +0x248 Reserved2         : [7] 0
   +0x264 NtProductType     : 1 ( NtProductWinNt )
   +0x268 ProductTypeIsValid : 0x1 ''
   +0x26c NtMajorVersion    : 6
   +0x270 NtMinorVersion    : 1
   +0x274 ProcessorFeatures : [64]  ""
   +0x2b4 Reserved1         : 0x7ffeffff
   +0x2b8 Reserved3         : 0x80000000
   +0x2bc TimeSlip          : 0
   +0x2c0 AlternativeArchitecture : 0 ( StandardDesign )
   +0x2c4 AltArchitecturePad : [1] 0
   +0x2c8 SystemExpirationDate : _LARGE_INTEGER 0x0
   +0x2d0 SuiteMask         : 0x110
   +0x2d4 KdDebuggerEnabled : 0x3 ''
   +0x2d5 NXSupportPolicy   : 0x2 ''
   +0x2d8 ActiveConsoleId   : 1
   +0x2dc DismountCount     : 0
   +0x2e0 ComPlusPackage    : 0xffffffff
   +0x2e4 LastSystemRITEventTickCount : 0
   +0x2e8 NumberOfPhysicalPages : 0xbff7e
   +0x2ec SafeBootMode      : 0 ''
   +0x2ed TscQpcData        : 0 ''
   +0x2ed TscQpcEnabled     : 0y0
   +0x2ed TscQpcSpareFlag   : 0y0
   +0x2ed TscQpcShift       : 0y000000 (0)
   +0x2ee TscQpcPad         : [2]  ""
   +0x2f0 SharedDataFlags   : 0xe
   +0x2f0 DbgErrorPortPresent : 0y0
   +0x2f0 DbgElevationEnabled : 0y1
   +0x2f0 DbgVirtEnabled    : 0y1
   +0x2f0 DbgInstallerDetectEnabled : 0y1
   +0x2f0 DbgSystemDllRelocated : 0y0
   +0x2f0 DbgDynProcessorEnabled : 0y0
   +0x2f0 DbgSEHValidationEnabled : 0y0
   +0x2f0 SpareBits         : 0y00000000000000000000000000 (0)
   +0x2f4 DataFlagsPad      : [1] 0
   +0x2f8 TestRetInstruction : 0xc3
   +0x300 SystemCall        : 0x76f770f0
   +0x304 SystemCallReturn  : 0x76f770f4
   +0x308 SystemCallPad     : [3] 0
   +0x320 TickCount         : _KSYSTEM_TIME
   +0x320 TickCountQuad     : 0x52e
   +0x320 ReservedTickCountOverlay : [3] 0x52e
   +0x32c TickCountPad      : [1] 0
```

We can find sysenter

```
0: kd> u 0x76f770f0
ntdll!KiFastSystemCall:
76f770f0 8bd4            mov     edx,esp
76f770f2 0f34            sysenter
ntdll!KiFastSystemCallRet:
76f770f4 c3              ret
76f770f5 8da42400000000  lea     esp,[esp]
76f770fc 8d642400        lea     esp,[esp]
ntdll!KiIntSystemCall:
76f77100 8d542408        lea     edx,[esp+8]
76f77104 cd2e            int     2Eh
76f77106 c3              ret
```

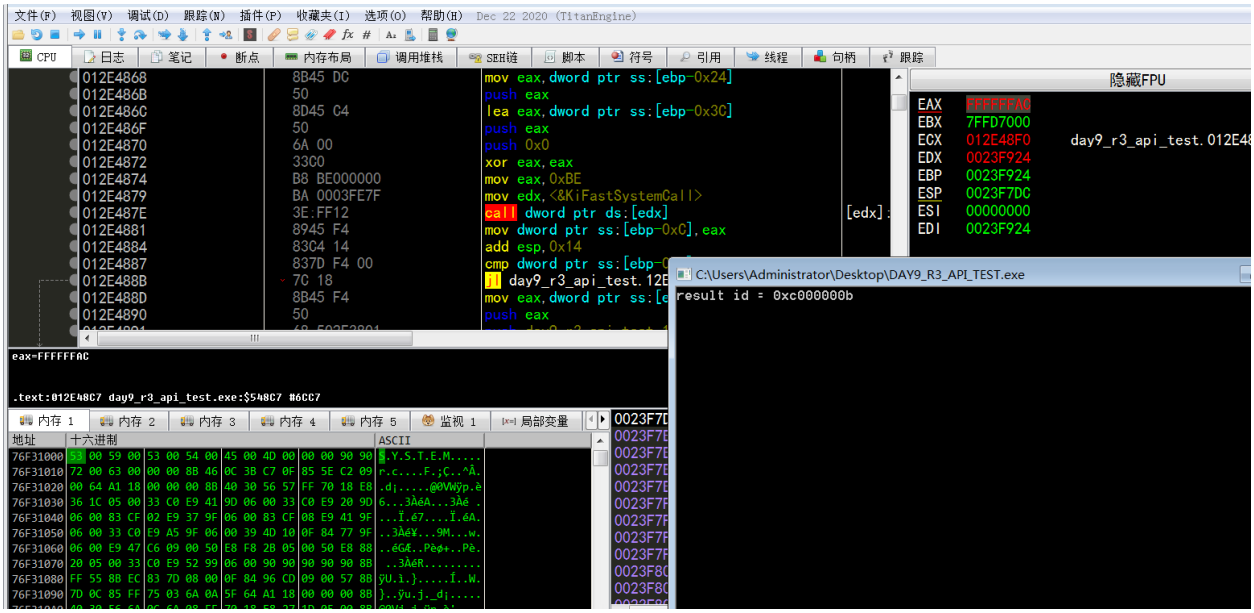
Let’s draw a picture about these



Realize OpenProcess in R3 don't use this API

```
#include <stdio.h>
#include <Windows.h>
#include <winternl.h>
/*
int main()
{
  _asm int 3;
  OpenProcess(NULL, NULL, NULL);
  return 0;
}
*/

//实现不使用OpenProcess API来实现该功能
//

int main()
{
  _asm int 3;
  int result;
  ULONG status;

  DWORD dwDesiredAccess = NULL;
  BOOL bInheritHandle = NULL;
  DWORD dwProcessId = NULL;

  CLIENT_ID ClientId;
  ClientId.UniqueProcess = (HANDLE)dwProcessId;

  OBJECT_ATTRIBUTES ObjectAttributes;
  ObjectAttributes.Attributes = bInheritHandle ? 2 : 0;

  ClientId.UniqueThread = 0;
  ObjectAttributes.Length = 24;
  ObjectAttributes.RootDirectory = 0;
  ObjectAttributes.ObjectName = 0;
  ObjectAttributes.SecurityDescriptor = 0;
  ObjectAttributes.SecurityQualityOfService = 0;

  _asm
  {
    lea eax, ClientId;
    push eax;
    lea eax, ObjectAttributes;
    push eax;
    mov eax, dwDesiredAccess;
    push eax;
```

```
    lea eax, dwProcessId;
    push eax;
    push 0;

    xor eax, eax;
    mov eax, 0xBE;
    mov edx, 0x7FFE0300;
    call dword ptr ds:[edx];
    mov result, eax;
    add esp, 0x14;
  }

  if (result >= 0)
  {
    printf("result id = 0x%x\r\n", result);
    return (HANDLE)dwProcessId;
  }
  else
  {
    printf("result id = 0x%x\r\n", result);
    return 0;
  }
}
```

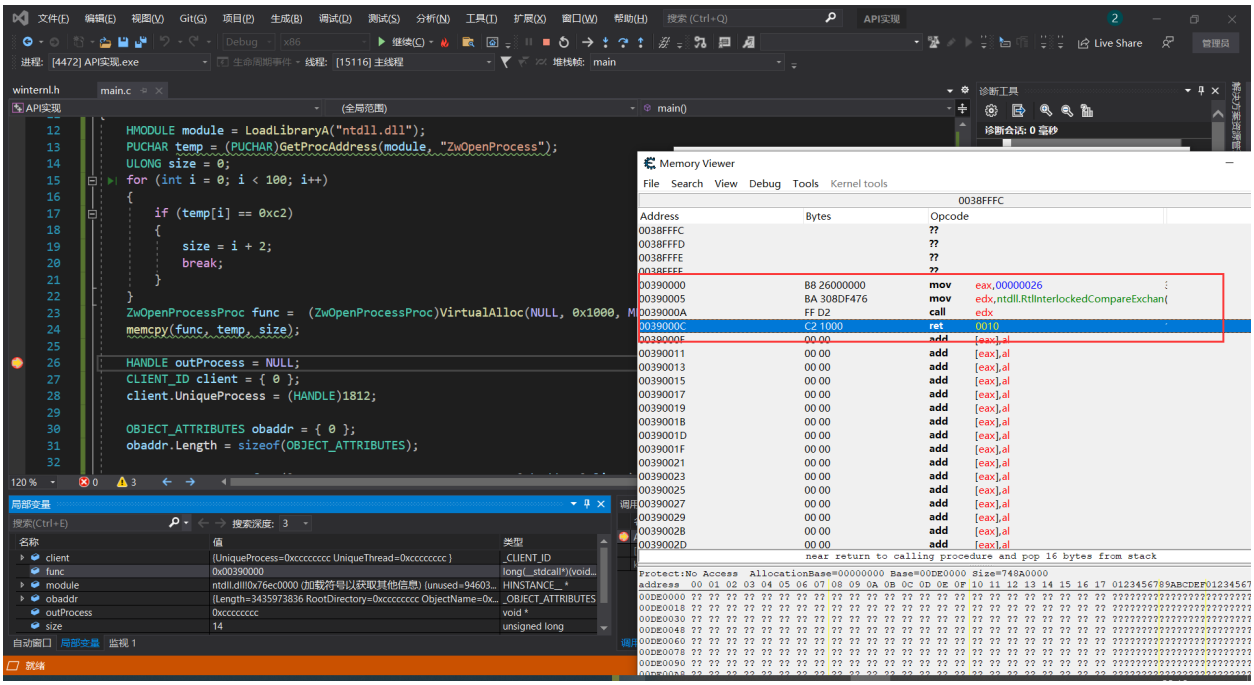


Windows system call (DAY 2 System calls into the kernel

Yesterday Code i rewrite

```
#include <stdio.h>
#include <Windows.h>
#include <winternl.h>



typedef NTSTATUS(WINAPI *ZwOpenProcessProc)
(HANDLE ProcessHandle, ACCESS_MASK DesiredAccess, POBJECT_ATTRIBUTES ObjectAttributes, PCLIENT_ID CLIENT_ID);

int main()
{
  HMODULE module = LoadLibraryA("ntdll.dll");
  PUCHAR temp = (PUCHAR)GetProcAddress(module, "ZwOpenProcess");
  ULONG size = 0;
```

```
  for (int i = 0; i < 100; i++)
  {
    if (temp[i] == 0xc2)
    {
      size = i + 2;
      break;
    }
  }
  ZwOpenProcessProc func =  (ZwOpenProcessProc)VirtualAlloc(NULL, 0x1000, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  memcpy(func, temp, size);

  HANDLE outProcess = NULL;
  CLIENT_ID client = { 0 };
  client.UniqueProcess = (HANDLE)1812;

  OBJECT_ATTRIBUTES obaddr = { 0 };
  obaddr.Length = sizeof(OBJECT_ATTRIBUTES);

  NTSTATUS status = func(&outProcess, PROCESS_ALL_ACCESS, &obaddr, &client);

  return 0;
}
```



Reverse Kifastcallentry

在之前的逆向过程中可以得到信息分别是：

eax : 服务号

edx : 保存的esp地址

在R3调用sysenter指令之后，CPU会做出如下的操作：

1. 将SYSENTER_CS_MSR的值装载到cs寄存器 (rdmsr 174)
2. 将SYSENTER_EIP_MSR的值装载到eip寄存器 (rdmsr 176)
3. 将SYSENTER_CS_MSR的值加8 (Ring0的堆栈段描述符)装载到ss寄存器
4. 将SYSENTER_ESP_MSR的值装载到esp寄存器 (rdmsr 175)

5. 开始执行指定的Ring0代码
6. 如果EFLAGS 寄存器的VM标志被置位，则清除该标志

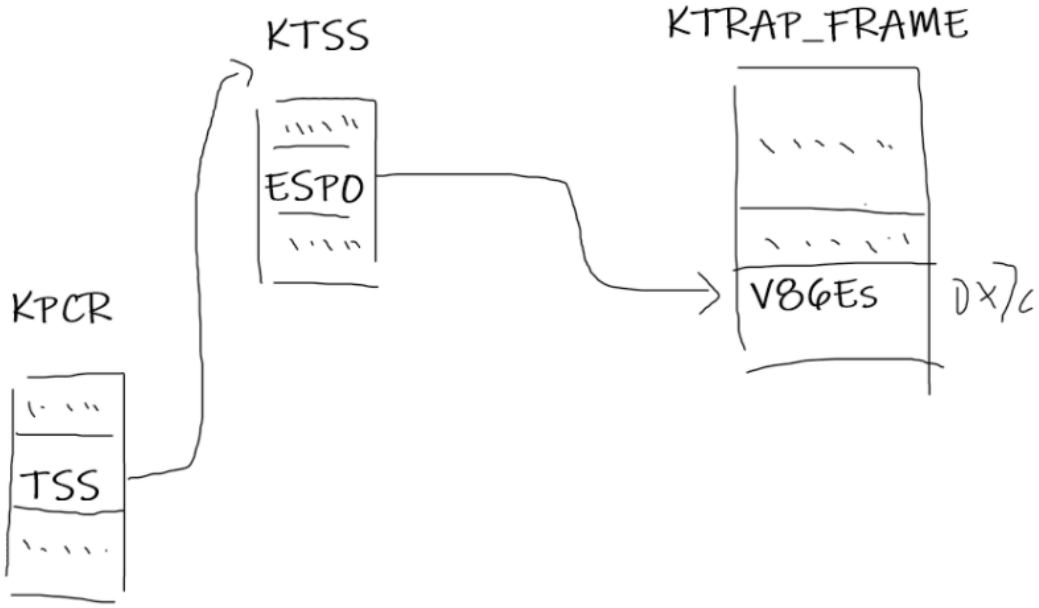
在Ring0代码执行完毕，调用SYSEXIT 指令退回Ring3时，CPU会做出如下操作:

1. 将SYSENTER_CS_MSR 的值加16 (Ring3 的代码段描述符)装载到cs寄存器
2. 将寄存器edx的值装载到eip寄存器
3. 将SYSENTER_CS_MSR的值加24 (Ring3 的堆栈段描述符)装载到ss寄存器
4. 将寄存器ecx的值装载到esp寄存器
5. 将特权级切换到Ring3
6. 继续执行Ring3的代码

```
.text:00435720                mov     ecx, 23h ; '#'
.text:00435725                push    30h ; '0'
.text:00435727                pop     fs              ; 修改fs段寄存器
.text:00435729                mov     ds, ecx         ; 修改ds段寄存器
.text:0043572B                mov     es, ecx         ; es = ds
.text:0043572D                mov     ecx, large fs:_KPCR.TSS ; 获取KPCR_TSS结构地址
.text:00435734                mov     esp, [ecx+_KTSS.Esp0] ; 切换到0换的esp地址 指向了_KTRAP_FRAME中的HardwareSegSs
.text:00435737                push    23h ; '#'       ; HardwareSegSs = 0x23
.text:00435739                push    edx             ; 由于edx在没有来到kifastcallentry的时候做了一个指令叫做mov edx,esp 所以HardwareEsp = r3_ed
.text:0043573A                pushf                   ; EFlags = R3_ELAGS
.text:0043573B
.text:0043573B loc_43573B:                            ; CODE XREF: _KiFastCallEntry2+23↑j
.text:0043573B                push    2
.text:0043573D                add     edx, 8          ; edx + 8由于在之前经历过了Ntdll.NtOpenProcess和KiFastSystemCall 所以根据栈来看这里为获取参数
.text:00435740                popf                    ; eflag = 2
.text:00435741                or      byte ptr [esp+1], 2 ; 启用中断位
.text:00435746                push    1Bh             ; SegCs = 0x1B
.text:00435748                push    dword ptr ds:0FFDF0304h ; Eip = 0xFFDF0304 要返回的Eip 在内核地址0xffdf0000为KUSER_SHARED_DATA的地址
.text:00435748                                        ; 0x304为KUSER_SHARED_DATA.SystemCallReturn
.text:0043574E                push    0               ; ErrCode = 0
.text:00435750                push    ebp             ; 保存 3环ebp ebx esi edi 到trap_frame结构中
.text:00435751                push    ebx
.text:00435752                push    esi
.text:00435753                push    edi
.text:00435754                mov     ebx, large fs:_KPCR.SelfPcr ; ebx = _KPCR结构地址
.text:0043575B                push    3Bh ; ';'       ; SegFs = 0x3B
.text:0043575D                mov     esi, [ebx+_KPCR.PrcbData.CurrentThread] ; 获取当前线程地址
.text:00435763                push    [ebx+_KPCR.___u0.NtTib.ExceptionList] ; 保存旧的ExceptionList
.text:00435765                mov     [ebx+_KPCR.___u0.NtTib.ExceptionList], 0FFFFFFFFh ; 设置新的列表
.text:0043576B                mov     ebp, [esi+_KTHREAD.InitialStack] ; 原始栈底
.text:0043576E                push    1               ; PreviousPreviousMode = 1 代表了是什么模式
.text:00435770                sub     esp, 48h        ; Trap_frame地址 提升到了最开始的地址
.text:00435773                sub     ebp, 29Ch
.text:00435779                mov     [esi+_KTHREAD.PreviousMode], 1 ; 设置了当前线程的新的模式
.text:00435780                cmp     ebp, esp        ; 比较设置的栈顶和栈底的地址位置
.text:00435782                jnz     short loc_43571B
.text:00435784                and     [ebp+_KTRAP_FRAME.Dr7], 0
.text:00435788                test    [esi+_KTHREAD.Header.___u0.__s3.DebugActive], 0DFh ; 看看我们是否需要保存调试寄存器
.text:0043578C                mov     [esi+_KTHREAD.TrapFrame], ebp ; 把Trap_frame结构保存
.text:00435792                jnz     Dr_FastCallDrSave
.text:00435798
.text:00435798 loc_435798:                            ; CODE XREF: Dr_FastCallDrSave+D↑j
.text:00435798                                        ; Dr_FastCallDrSave+79↑j
.text:00435798                mov     ebx, [ebp+_KTRAP_FRAME._Ebp] ; ebx = ebp
.text:0043579B                mov     edi, [ebp+_KTRAP_FRAME._Eip] ; edi = eip
.text:0043579E                mov     [ebp+_KTRAP_FRAME.DbgArgPointer], edx ; 参数指针保存到DbgArgPointer
.text:004357A1                mov     [ebp+_KTRAP_FRAME.DbgArgMark], 0BADB0D00h
.text:004357A8                mov     [ebp+_KTRAP_FRAME.DbgEbp], ebx ; ebgebp = r3_ebp
.text:004357AB                mov     [ebp+_KTRAP_FRAME.DbgEip], edi ; dbgeip = r3_eip
.text:004357AE                sti
.text:004357AF
.text:004357AF loc_4357AF:                            ; CODE XREF: _KiBBTUnexpectedRange+18↑j
.text:004357AF                                        ; _KiSystemService+7F↑j
.text:004357AF                mov     edi, eax        ; eax 服务号传给edi
.text:004357B1                shr     edi, 8          ; (eax >> 8) & 0x10 判断是哪张表
.text:004357B4                and     edi, 10h
.text:004357B7                mov     ecx, edi        ; ecx = edi 等于所要表的偏移
.text:004357B9                add     edi, [esi+_KTHREAD.ServiceTable] ; edi = ServiceTable + 偏移 获取到表的地址
```

```
.text:004357BF                mov     ebx, eax        ; ebx = 服务号
.text:004357C1                and     eax, 0FFFh
.text:004357C6                cmp     eax, [edi+8]    ; 比较所在的位置 是不是超出了表的limit
.text:004357C9                jnb     _KiBBTUnexpectedRange
.text:004357CF                cmp     ecx, 10h
.text:004357D2                jnz     short loc_4357EE ; GUI service
.text:004357D4                mov     ecx, [esi+_KTHREAD.Teb] ; 获取Teb地址赋值给ecx
.text:004357DA                xor     esi, esi
.text:004357DC                or      esi, [ecx+_TEB.GdiBatchCount]
.text:004357E2                jz      short loc_4357EE ; 系统调用次数 +1
.text:004357E4                push    edx
.text:004357E5                push    eax
.text:004357E6                call    ds:_KeGdiFlushUserBatch ; 批量刷新界面
.text:004357EC                pop     eax
.text:004357ED                pop     edx
.text:004357EE
.text:004357EE loc_4357EE:                            ; CODE XREF: _KiFastCallEntry+B2↑j
.text:004357EE                                        ; KiSystemServiceAccessTeb()+6↑j
.text:004357EE                inc     large dword ptr fs:_KPCR.PrcbData.KeSystemCalls ; 系统调用次数 +1
.text:004357F5                mov     esi, edx        ; esi等于参数指针
.text:004357F7                xor     ecx, ecx
.text:004357F9                mov     edx, [edi+0Ch]  ; 获取参数的字节表到edx
.text:004357FC                mov     edi, [edi]      ; ServiceTables的函数表的地址赋值给edi
.text:004357FE                mov     cl, [eax+edx]   ; 获取当前的函数的参数个数给cl
.text:00435801                mov     edx, [edi+eax*4] ; 获取到r0函数的地址
.text:00435804                sub     esp, ecx        ; 提升栈
.text:00435806                shr     ecx, 2          ; ecx = 有多少个DWORDS
.text:00435809                mov     edi, esp        ; edi = esp
.text:0043580B                test    byte ptr [ebp+(_KTRAP_FRAME.EFlags+2)], 2
.text:0043580F                jnz     short loc_435817 ; 判断是不是r3的线性地址
.text:00435811                test    byte ptr [ebp+_KTRAP_FRAME.SegCs], 1
.text:00435815                jz      short _KiSystemServiceCopyArguments@0 ; KiSystemServiceCopyArguments()
.text:00435817
.text:00435817 loc_435817:                            ; CODE XREF: KiSystemServiceAccessTeb()+33↑j
.text:00435817                cmp     esi, ds:_MmUserProbeAddress ; 判断是不是r3的线性地址
.text:0043581D                jnb     loc_435A51
.text:00435823                rep movsd
.text:00435825                test    byte ptr [ebp+_KTRAP_FRAME.SegCs], 1 ; 根据ecx进行参数复制后
.text:00435829                jz      short loc_435841
.text:0043582B                mov     ecx, large fs:_KPCR.PrcbData.CurrentThread
.text:00435832                mov     edi, [esp+0]
.text:00435835                mov     [ecx+_KTHREAD.SystemCallNumber], ebx
.text:0043583B                mov     [ecx+_KTHREAD.FirstArgument], edi
.text:00435841
.text:00435841 loc_435841:                            ; CODE XREF: KiSystemServiceCopyArguments()+6↑j
.text:00435841                mov     ebx, edx
.text:00435843                test    byte ptr ds:dword_52DE48, 40h
.text:0043584A                setnz   [ebp+_KTRAP_FRAME.Logging]
.text:0043584E                jnz     loc_435BD4
.text:00435854
.text:00435854 loc_435854:                            ; CODE XREF: KiSystemServiceCopyArguments()+3B8↓j
.text:00435854                call    ebx             ; ebx = edx(函数地址位置) call 函数
```

总结就是：通过KPCR来获取到 Trap_frame的结构将3环的结构保存到这个结构中，通过3环得到的系统服务号，来判断ServiceTable是哪个，然后将ServiceTable中的参数个数获取到，然后传入参数调用真正的底层函数



Windows system call (DAY 3 System call returns

```
.text:00435856 F6 45 6C 01                          test    byte ptr [ebp+_KTRAP_FRAME.SegCs], 1
.text:0043585A 74 34                                jz      short loc_435890 ; 判断是否为用户模式 (R3还是R0)
.text:0043585C 8B F0                                mov     esi, eax        ; eax 为返回值 保存返回值
.text:0043585E FF 15 68 11 40 00                    call    ds:__imp__KeGetCurrentIrql@0 ; 判断当前Irql的等级
.text:00435864 0A C0                                or      al, al
.text:00435866 0F 85 2F 03 00 00                    jnz     loc_435B9B      ; 如果IRQL 在结束的时候不等于0的时候 蓝屏，因为用户层面只能是0
.text:0043586C 8B C6                                mov     eax, esi
.text:0043586E 64 8B 0D 24 01 00 00                 mov     ecx, large fs:_KPCR.PrcbData.CurrentThread
.text:00435875 F6 81 34 01 00 00 FF                 test    [ecx+_KTHREAD.ApcStateIndex], 0FFh ; 没有恢复挂靠，蓝屏
.text:0043587C 0F 85 37 03 00 00                    jnz     loc_435BB9
.text:00435882 8B 91 84 00 00 00                    mov     edx, dword ptr [ecx+_KTHREAD.___u26.__s0.KernelApcDisable]
.text:00435888 0B D2                                or      edx, edx
.text:0043588A 0F 85 29 03 00 00                    jnz     loc_435BB9
.text:00435890
.text:00435890                          loc_435890:                             ; CODE XREF: KiSystemServicePostCall()+4↑j
.text:00435890 8B E5                                mov     esp, ebp        ; ebp一直都是trap_frame没有变过
.text:00435892 80 7D 12 00                          cmp     [ebp+_KTRAP_FRAME.Logging], 0 ; 看是否是需要打印日志
.text:00435896 0F 85 44 03 00 00                    jnz     loc_435BE0
.text:0043589C
.text:0043589C                          loc_43589C:                             ; CODE XREF: _KiBBTUnexpectedRange+3C↑j
.text:0043589C                                                                  ; _KiBBTUnexpectedRange+47↑j ...
.text:0043589C 64 8B 0D 24 01 00 00                 mov     ecx, large fs:_KPCR.PrcbData.CurrentThread
.text:004358A3 8B 55 3C                             mov     edx, [ebp+_KTRAP_FRAME._Edx] ; R3的ESP，或者是R0的trap_frame
.text:004358A6 89 91 28 01 00 00                    mov     [ecx+_KTHREAD.TrapFrame], edx
.text:004358AC FA                                   cli
.text:004358AD F6 45 72 02                          test    byte ptr [ebp+(_KTRAP_FRAME.EFlags+2)], 2 ; 关闭中断要恢复现场，判断是否是8086模式
.text:004358B1 75 06                                jnz     short loc_4358B9
.text:004358B3 F6 45 6C 01                          test    byte ptr [ebp+_KTRAP_FRAME.SegCs], 1 ; 判断是不是R3
.text:004358B7 74 67                                jz      short loc_435920
.text:004358B9
.text:004358B9                          loc_4358B9:                             ; CODE XREF: _KiServiceExit+5↑j
.text:004358B9                                                                  ; _KiServiceExit+6F↓j
.text:004358B9 64 8B 1D 24 01 00 00                 mov     ebx, large fs:_KPCR.PrcbData.CurrentThread
.text:004358C0 F6 43 02 02                          test    byte ptr [ebx+(_KTRAP_FRAME.DbgEbp+2)], 2
.text:004358C4 74 08                                jz      short loc_4358CE
.text:004358C6 50                                   push    eax
.text:004358C7 53                                   push    ebx
.text:004358C8 E8 56 0E 0A 00                       call    _KiCopyCounters@4 ; 性能统计
```

```
.text:004358CD 58                                      pop     eax
.text:004358CE
.text:004358CE                          loc_4358CE:                             ; CODE XREF: _KiServiceExit+18↑j
.text:004358CE C6 43 3A 00                             mov     [ebx+_KTHREAD.Alerted], 0
.text:004358D2 80 7B 56 00                             cmp     byte ptr [ebx+_KTSS.Reserved5], 0
.text:004358D6 74 48                                   jz      short loc_435920
.text:004358D8 8B DD                                   mov     ebx, ebp
.text:004358DA 89 43 44                                mov     [ebx+44h], eax
.text:004358DD C7 43 50 3B 00 00 00                    mov     [ebx+_KTRAP_FRAME.SegFs], 3Bh ; ';'
.text:004358E4 C7 43 38 23 00 00 00                    mov     [ebx+_KTRAP_FRAME.SegDs], 23h ; '#'
.text:004358EB C7 43 34 23 00 00 00                    mov     [ebx+_KTRAP_FRAME.SegEs], 23h ; '#'
.text:004358F2 C7 43 30 00 00 00 00                    mov     [ebx+_KTRAP_FRAME.SegGs], 0
.text:004358F9 B9 01 00 00 00                          mov     ecx, 1          ; NewIrql
.text:004358FE FF 15 5C 11 40 00                       call    ds:__imp_@KfRaiseIrql@4 ; KfRaiseIrql(x)
.text:00435904 50                                      push    eax
.text:00435905 FB                                      sti
.text:00435906 53                                      push    ebx
.text:00435907 6A 00                                   push    0
.text:00435909 6A 01                                   push    1
.text:0043590B E8 DC 8A 03 00                          call    _KiDeliverApc@12 ; 派发APC
.text:00435910 59                                      pop     ecx             ; NewIrql
.text:00435911 FF 15 58 11 40 00                       call    ds:__imp_@KfLowerIrql@4 ; KfLowerIrql(x)
.text:00435917 8B 43 44                                mov     eax, [ebx+44h]
.text:0043591A FA                                      cli
.text:0043591B EB 9C                                   jmp     short loc_4358B9
.text:0043591B                          ; ---------------------------------------------------------------------------
.text:0043591D 8D 49 00                                align 10h
.text:00435920
.text:00435920                          loc_435920:                             ; CODE XREF: _KiServiceExit+B↑j
.text:00435920                                                                  ; _KiServiceExit+2A↑j
.text:00435920 8B 54 24 4C                             mov     edx, [esp+_KTRAP_FRAME.ExceptionList]
.text:00435924 64 89 15 00 00 00 00                    mov     large fs:_KPCR, edx ; 还原现场
.text:0043592B 8B 4C 24 48                             mov     ecx, [esp+_KTRAP_FRAME.PreviousPreviousMode]
.text:0043592F 64 8B 35 24 01 00 00                    mov     esi, large fs:_KPCR.PrcbData.CurrentThread
.text:00435936 88 8E 3A 01 00 00                       mov     [esi+_KTHREAD.PreviousMode], cl
.text:0043593C F7 44 24 2C FF 23 FF FF                 test    [esp+_KTRAP_FRAME.Dr7], 0FFFF23FFh ; 判断DR寄存器是否有调试，给DR寄存器赋值
.text:00435944 0F 85 7E 00 00 00                       jnz     loc_4359C8
.text:0043594A
.text:0043594A                          loc_43594A:                             ; CODE XREF: _KiServiceExit+12C↓j
.text:0043594A                                                                  ; _KiServiceExit+15B↓j
.text:0043594A F7 44 24 70 00 00 02 00                 test    [esp+_KTRAP_FRAME.EFlags], 20000h ; 判断是不是虚拟8086模式（16位的情况）
.text:00435952 0F 85 34 0A 00 00                       jnz     loc_43638C
.text:00435958 66 F7 44 24 6C F9 FF                    test    word ptr [esp+_KTRAP_FRAME.SegCs], 0FFF9h
.text:0043595F 0F 84 B9 00 00 00                       jz      loc_435A1E
.text:00435965 66 83 7C 24 6C 1B                       cmp     word ptr [esp+_KTRAP_FRAME.SegCs], 1Bh
.text:0043596B 66 0F BA 64 24 6C 00                    bt      word ptr [esp+_KTRAP_FRAME.SegCs], 0
.text:00435972 F5                                      cmc
.text:00435973 0F 87 93 00 00 00                       ja      loc_435A0C      ; 如果CF=1 不跳
.text:00435979 66 83 7D 6C 08                          cmp     word ptr [ebp+_KTRAP_FRAME.SegCs], 8
.text:0043597E 74 05                                   jz      short loc_435985
.text:00435980
.text:00435980                          loc_435980:                             ; CODE XREF: _KiServiceExit+16D↓j
.text:00435980 8D 65 50                                lea     esp, [ebp+_KTRAP_FRAME.SegFs]
.text:00435983 0F A1                                   pop     fs
.text:00435985                                         assume fs:nothing
.text:00435985
.text:00435985                          loc_435985:                             ; CODE XREF: _KiServiceExit+D2↑j
.text:00435985 8D 65 54                                lea     esp, [ebp+_KTRAP_FRAME._Edi]
.text:00435988 5F                                      pop     edi
.text:00435989 5E                                      pop     esi
.text:0043598A 5B                                      pop     ebx
.text:0043598B 5D                                      pop     ebp
.text:0043598C 66 81 7C 24 08 80 00                    cmp     word ptr [esp+8], 80h ; '€' ; 判断cs是不是虚拟8086模式的段
.text:00435993 0F 87 0F 0A 00 00                       ja      loc_4363A8
.text:00435999 83 C4 04                                add     esp, 4          ; eip
.text:0043599C F7 44 24 04 01 00 00 00                 test    dword ptr [esp+4], 1 ; cs
.text:0043599C                          _KiServiceExit  endp ; sp-analysis failed
.text:0043599C
.text:004359A4
.text:004359A4                          _KiSystemCallExitBranch:                ; DATA XREF: KiRestoreFastSyscallReturnState():loc_413C36↑r
.text:004359A4                                                                  ; KiRestoreFastSyscallReturnState()+7D↑w ...
.text:004359A4 75 05                                   jnz     short _KiSystemCallExit
.text:004359A6 5A                                      pop     edx
.text:004359A7 59                                      pop     ecx
.text:004359A8 9D                                      popf
.text:004359A9 FF E2                                   jmp     edx
.text:004359AB                          ; ---------------------------------------------------------------------------
.text:004359AB                          ; START OF FUNCTION CHUNK FOR _KiSystemCallExit2
.text:004359AB
.text:004359AB                          _KiSystemCallExit:                      ; CODE XREF: .text:_KiSystemCallExitBranch↑j
.text:004359AB                                                                  ; _KiSystemCallExit2+8↓j
```

```
.text:004359AB                                                          ; DATA XREF: ...
.text:004359AB CF                                       iret
```

感觉主要学习这个System call的流程主要是也是为了反正hook ssdt的操作，以及怎么找到ssdt表
看看流程是什么样的，学习一下

1. ETW hook
2. 替换MSR 176
3. inline hook
4. 替换ssdt表中的函数地址
5. x86下重构ssdt表，hook想要的线程，因为在x86下，我们的ServiceTable在Thread结构下，但是在64位下的时候ServiceTable是写死的
6. 修改KUSER_SHARED_DATA→CO

Windows system call (DAY 4 SSDT HOOK

主要实现功能：根据自定义的函数，进行选择性函数的ssdt hook

struct.c

```
#pragma once
#include <ntifs.h>
typedef struct _RTL_PROCESS_MODULE_INFORMATION {
    HANDLE Section;                 // Not filled in
    PVOID MappedBase;
    PVOID ImageBase;
    ULONG ImageSize;
    ULONG Flags;
    USHORT LoadOrderIndex;
    USHORT InitOrderIndex;
    USHORT LoadCount;
    USHORT OffsetToFileName;
    UCHAR  FullPathName[256];
} RTL_PROCESS_MODULE_INFORMATION, * PRTL_PROCESS_MODULE_INFORMATION;

typedef struct _RTL_PROCESS_MODULES {
    ULONG NumberOfModules;
    RTL_PROCESS_MODULE_INFORMATION Modules[1];
} RTL_PROCESS_MODULES, * PRTL_PROCESS_MODULES;

typedef struct _ServiceItem
{
    PULONG pServiceTable;
    ULONG pCounterTable;
    ULONG NumberOfServices;
    PCHAR pArgumentTable;
}ServiceItem, * PServiceItem;

typedef struct _ServiceTable
{
    ServiceItem KernelItem;
    ServiceItem uiItem;
}ServiceTable, * PServiceTable;

typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemBasicInformation,
    SystemProcessorInformation,             // obsolete...delete
    SystemPerformanceInformation,
```

```
    SystemTimeOfDayInformation,
    SystemPathInformation,
    SystemProcessInformation,
    SystemCallCountInformation,
    SystemDeviceInformation,
    SystemProcessorPerformanceInformation,
    SystemFlagsInformation,
    SystemCallTimeInformation,
    SystemModuleInformation,
    SystemLocksInformation,
    SystemStackTraceInformation,
    SystemPagedPoolInformation,
    SystemNonPagedPoolInformation,
    SystemHandleInformation,
    SystemObjectInformation,
    SystemPageFileInformation,
    SystemVdmInstemulInformation,
    SystemVdmBopInformation,
    SystemFileCacheInformation,
    SystemPoolTagInformation,
    SystemInterruptInformation,
    SystemDpcBehaviorInformation,
    SystemFullMemoryInformation,
    SystemLoadGdiDriverInformation,
    SystemUnloadGdiDriverInformation,
    SystemTimeAdjustmentInformation,
    SystemSummaryMemoryInformation,
    SystemMirrorMemoryInformation,
    SystemPerformanceTraceInformation,
    SystemObsolete0,
    SystemExceptionInformation,
    SystemCrashDumpStateInformation,
    SystemKernelDebuggerInformation,
    SystemContextSwitchInformation,
    SystemRegistryQuotaInformation,
    SystemExtendServiceTableInformation,
    SystemPrioritySeperation,
    SystemVerifierAddDriverInformation,
    SystemVerifierRemoveDriverInformation,
    SystemProcessorIdleInformation,
    SystemLegacyDriverInformation,
    SystemCurrentTimeZoneInformation,
    SystemLookasideInformation,
    SystemTimeSlipNotification,
    SystemSessionCreate,
    SystemSessionDetach,
    SystemSessionInformation,
    SystemRangeStartInformation,
    SystemVerifierInformation,
    SystemVerifierThunkExtend,
    SystemSessionProcessInformation,
    SystemLoadGdiDriverInSystemSpace,
    SystemNumaProcessorMap,
    SystemPrefetcherInformation,
    SystemExtendedProcessInformation,
    SystemRecommendedSharedDataAlignment,
    SystemComPlusPackage,
    SystemNumaAvailableMemory,
    SystemProcessorPowerInformation,
    SystemEmulationBasicInformation,
    SystemEmulationProcessorInformation,
    SystemExtendedHandleInformation,
    SystemLostDelayedWriteInformation,
    SystemBigPoolInformation,
    SystemSessionPoolTagInformation,
    SystemSessionMappedViewInformation,
    SystemHotpatchInformation,
    SystemObjectSecurityMode,
    SystemWatchdogTimerHandler,
    SystemWatchdogTimerInformation,
    SystemLogicalProcessorInformation,
    SystemWow64SharedInformation,
    SystemRegisterFirmwareTableInformationHandler,
    SystemFirmwareTableInformation,
    SystemModuleInformationEx,
    SystemVerifierTriageInformation,
    SystemSuperfetchInformation,
    SystemMemoryListInformation,
    SystemFileCacheInformationEx,
    MaxSystemInfoClass  // MaxSystemInfoClass should always be the last enum
} SYSTEM_INFORMATION_CLASS;
```

```
NTSTATUS
NTAPI
ZwQuerySystemInformation(
    __in SYSTEM_INFORMATION_CLASS SystemInformationClass,
    __out_bcount_opt(SystemInformationLength) PVOID SystemInformation,
    __in ULONG SystemInformationLength,
    __out_opt PULONG ReturnLength
);
```

main.c:

```
#include <ntifs.h>
#include <ntimage.h>
#include "struct.h"


EXTERN_C PServiceTable KeServiceDescriptorTable;

PULONG pmem = NULL;
int id = NULL;

typedef NTSTATUS (NTAPI *NtSetEventProc)(__in HANDLE EventHandle, __out_opt PLONG PreviousState);

NtSetEventProc OldNtSetEvent = NULL;

NTSTATUS NtSetEvent(__in HANDLE EventHandle, __out_opt PLONG PreviousState)
{
  DbgPrintEx(77, 0, "-----------------------------------------------\r\n");
  OldNtSetEvent(EventHandle, PreviousState);
}

ULONG64 ExportTableFuncByName(char* pData, char* funcName)
{
  PIMAGE_DOS_HEADER pHead = (PIMAGE_DOS_HEADER)pData;
  PIMAGE_NT_HEADERS pNt = (PIMAGE_NT_HEADERS)(pData + pHead->e_lfanew);
  int numberRvaAndSize = pNt->OptionalHeader.NumberOfRvaAndSizes;
  PIMAGE_DATA_DIRECTORY pDir = (PIMAGE_DATA_DIRECTORY)&pNt->OptionalHeader.DataDirectory[0];

  PIMAGE_EXPORT_DIRECTORY pExport = (PIMAGE_EXPORT_DIRECTORY)(pData + pDir->VirtualAddress);

  ULONG64 funcAddr = 0;
  for (int i = 0; i < pExport->NumberOfNames; i++)
  {
    int* funcAddress = pData + pExport->AddressOfFunctions;
    int* names = pData + pExport->AddressOfNames;
    short* fh = pData + pExport->AddressOfNameOrdinals;
    int index = -1;
    char* name = pData + names[i];
    if (strcmp(name, funcName) == 0)
    {
      index = fh[i];
    }
    if (index != -1)
    {
      funcAddr = pData + funcAddress[index];
      break;
    }
  }
  if (!funcAddr)
  {
    KdPrint(("没有找到函数%s\r\n", funcName));

  }
  else
  {
    KdPrint(("找到函数%s addr %p\r\n", funcName, funcAddr));
  }
  return funcAddr;
}

ULONG_PTR QueryModule(char* MoudleName)
{
  if (MoudleName == NULL) return NULL;
  RTL_PROCESS_MODULES rtlMoudles = { 0 };
  PRTL_PROCESS_MODULES SystemMoudles = &rtlMoudles;
```

```
  BOOLEAN isAllocate = FALSE;
  ULONG_PTR MoudleBase = NULL;

  ULONG retLen = 0;
  //获取信息长度返回到retLen
  NTSTATUS status = ZwQuerySystemInformation(SystemModuleInformation, SystemMoudles, sizeof(RTL_PROCESS_MODULES), &retLen);

  if (status == STATUS_INFO_LENGTH_MISMATCH)
  {
    SystemMoudles = ExAllocatePool(PagedPool, retLen + sizeof(RTL_PROCESS_MODULES));
    if (!SystemMoudles) return FALSE;
    memset(SystemMoudles, 0, retLen + sizeof(RTL_PROCESS_MODULES));
    status = ZwQuerySystemInformation(SystemModuleInformation, SystemMoudles, retLen + sizeof(RTL_PROCESS_MODULES), &retLen);

    if (!NT_SUCCESS(status))
    {
      ExFreePool(SystemMoudles);
      return FALSE;
    }

    PUCHAR KernelMoudleName = ExAllocatePool(PagedPool, strlen(MoudleName) + 1);
    memset(KernelMoudleName, 0, strlen(MoudleName) + 1);
    memcpy(KernelMoudleName, MoudleName, strlen(MoudleName));
    //转换为大写
    _strupr(KernelMoudleName);


    for (int i = 0; i < SystemMoudles->NumberOfModules; i++)
    {
      PRTL_PROCESS_MODULE_INFORMATION MoudleInfo = &SystemMoudles->Modules[i];
      PUCHAR PathName = _strupr(MoudleInfo->FullPathName + MoudleInfo->OffsetToFileName);
      if (strstr(KernelMoudleName, PathName))
      {
        MoudleBase = MoudleInfo->ImageBase;
        break;
      }
    }
  }
  return MoudleBase;
}

BOOLEAN SsdtHook(ULONG_PTR NewFuncAddr, char* HookFucName)
{
  PHYSICAL_ADDRESS phyAddr = MmGetPhysicalAddress(KeServiceDescriptorTable->KernelItem.pServiceTable);
  //将给定的物理地址范围映射到未分页的系统空间
  pmem = (PULONG)MmMapIoSpace(phyAddr, PAGE_SIZE, MmCached);
  DbgPrintEx(77, 0, "pmem = %x", pmem);
  //假设要通过进程的PID的话 -> _EPROCESS -> PEB -> PEB_LDR_DATA -> LDR_DATA_TABLE_ENTRY
  //当然可以遍历当前系统所有的吧，不然像ark那些怎么做出来的呢！
  char* NeedModuleName = "ntdll.dll";
  ULONG_PTR base = QueryModule(NeedModuleName);
  PUCHAR func = ExportTableFuncByName((char*)base, HookFucName);
  for (int i = 0; i < 100; i++)
  {
    DbgPrintEx(77, 0, "0x%x\r\n", func[i]);
    if (func[i] == 0xB8)
    {
      id = *((int*)(func + i + 1));
      break;
    }
  }
  if (id == NULL)
  {
    return FALSE;
  }
  OldNtSetEvent = pmem[id];
  pmem[id] = NewFuncAddr;
  return TRUE;
}

VOID
DriverUnload(
  _In_ struct _DRIVER_OBJECT* DriverObject
)
{
  if (id != NULL)
  {
    pmem[id] = OldNtSetEvent;
  }
  MmUnmapIoSpace(pmem, PAGE_SIZE);
  DbgPrintEx(77, 0, "-------------------------OVER-----------------\r\n");
```

```
}


NTSTATUS DriverEntry(PDRIVER_OBJECT pDriver, PUNICODE_STRING pSeg)
{
  pDriver->DriverUnload = DriverUnload;

  DbgBreakPoint();
  DbgPrintEx(77, 0, "pServiceTable = %x, KeServiceDescriptorTableShadow = %x\r\n",
  KeServiceDescriptorTable->KernelItem.pServiceTable, (ULONG)KeServiceDescriptorTable + 0x40);

  // change CR0
  // MDL 映射
  //

  //实现根据名字找到对应函数的索引
  //进行hook
  char* HookFuncName = "ZwSetEvent";
  BOOLEAN is = SsdtHook(NtSetEvent, HookFuncName);
  if (is == NULL)
  {
    DbgPrintEx(77, 0, "---------------FALSE----------------\r\n");
    return FALSE;
  }

  //OldNtSetEvent = pmem[0x143];
  //pmem[0x143] = NtSetEvent;
  return STATUS_SUCCESS;
}
```

测试结果：