



逆向Windows 8: 内核安全特性拾趣

MJ0011

th_decoder@126.com

目的：

逆向Windows 8 Release Preview版本

发现一些新的用于防御或缓和内核漏洞攻击的安全特性

目标：

主要关注ntoskrnl

工具： IDA Pro/Hex-rays/windbg

- 禁止零页内存分配
- 禁止Win32k系统调用
- 安全性故障中断
- 不可执行的非分页池
- 使用Intel® Secure Key 技术
- 使用Intel® SMEP 技术

- 零页内存：Windows上供 16位虚拟机NTVDM使用，确保16位代码正常运行
- 内核漏洞攻击技巧，通过ZwAllocateVirtualMemory等系统调用可以在进程中分配出零页内存
- 触发未初始化对象指针/数据指针引用漏洞或辅助漏洞攻击。
 - 案例：CVE-2010-4398
N-Protect TKRgAc2k.sys kernel 0day(POC2010)
- Window 8 上：禁止进程申请低地址内存(0x0~0x10000)
- EPROCESS->Flags.VdmAllowed

- Windows 8 上，16位虚拟机默认禁用，开启需要管理员权限



- Windows8在所有可能的内存分配位置检查零页分配
 - MiCreatePebOrTeb : 进线程启动, 创建PEB或TEB时
 - MiMapViewOfImageSection->MiIsVaRangeAvailable:
映射镜像内存区时
 - MiMapViewOfDataSection/MiMapViewOfPhysicalSection
映射数据内存区时
 - MmMapLockedPagesSpecifyCache/MmMapLockedPages->
MiMapLockedPagesInUserSpace
映射用户地址时 (内核自身未这样使用, 但其他驱动使用时会限制)
 - NtAllocateVirtualMemory:分配进程内存

禁止Win32k系统调用

- EPROCESS->Flags2.DisallowWin32kSystemCalls
- 实现在：KiFastCallEntry(2)->PsConvertToGuiThread

```
-----  
loc_7BC7F6:                                ; CODE XREF: PsConvertToGuiThread()+23↑j  
      mov     eax, [ebp+CurrentThread]  
      cmp     [eax+KTHREAD.ServiceTable], offset _KeServiceDescriptorTable  
      jz      short @ThreadServiceTableIsSSDT  
  
      mov     eax, STATUS_ALREADY_WIN32  
      jmp     locret_7BC8E3  
  
-----  
@ThreadServiceTableIsSSDT:                 ; CODE XREF: PsConvertToGuiThread()+39↑j  
      mov     eax, [ebp+CurrentThread]  
      mov     eax, [eax+KTHREAD.Process]  
      mov     [ebp+CurrentProcess], eax  
      mov     eax, [ebp+CurrentProcess]  
      mov     [ebp+CurrentProcess_], eax  
      mov     eax, [ebp+CurrentProcess_]  
      mov     eax, [eax+EPROCESS.Flags2]  
      and     eax, 80000000h ; DisallowWin32kSystemCalls : Pos 31, 1 Bit  
      jz      short @AllowConvertToGuiThread  
  
      mov     eax, STATUS_ACCESS_DENIED  
      jmp     locret_7BC8E3  
  
-----  
@AllowConvertToGuiThread:                   ; CODE XREF: PsConvertToGuiThread()+65↑j
```

- 禁用win32k 系统调用的作用
- Win32k.sys : Windows内核漏洞高发 , 调用不受进程权限限制
 - MS11-087 Trojan.win32.Duqu 字体解析漏洞
- 目前应用沙箱的防御策略: Job UI限制 效果不佳
- 禁用win32k系统调用可以有效防御一切win32k.sys相关0day , 无需内核驱动 , 配置简单
- 防御非0day的USER/GDI相关技巧突破沙箱

- PsConvertToGuiThread : GUI线程首次调用win32k system call时切换线程状态
- 应用DisallowWin32kSystemCalls标志后禁止切换，任何对USER32/GDI32相关调用都会失败
- 三种方式获得此标志：
 - 1.IEFO注册表：
 - HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\MitigationOptions (0x10000000)
 - 在进程创建过程NtCreateUserProcess->PspAllocateProcess->PspApplyMitigationOptions应用到标志位上
 - 2.文档化的API:SetProcessMitigationPolicy
 - 实际调用NtSetInformationProcess->ProcessMitigationPolicy设置标志位
 - 3.从父进程继承

- Windows 8 中加入的新的中断: Int 0x29
- Windows8的内核和其他驱动程序在发生安全性故障时使用，将直接引发BSOD
- 最常见的使用地方在双向链表的链接/脱链前，Windows8的OS加载器、内核和驱动程序中，在所有双向链表的使用上都加入了这个处理
- 被称为Safe Linking & Safe Unlinking
 - 安全链接例子:IoRegisterFsRegistrationChangeMountAware
 - 安全脱链例子:IoUnregisterFileSystem
- 防御利用篡改链接后脱链/链接形成任意地址写入

安全脱链机制触发int 0x29中断示例: IoUnregisterFileSystem

```
FILE:00100200
• PAGE:00786288      mov     edi, [ebp+DeviceObject]
• PAGE:0078628B      lea    eax, [edi+DEVICE_OBJECT.Queue.ListEntry.Flink]
• PAGE:0078628E      xor    ebx, ebx
• PAGE:00786290      cmp    [eax+LIST_ENTRY.Flink], ebx
• PAGE:00786292      jz     short loc_7862A7
PAGE:00786292
• PAGE:00786294      mov    edx, [eax+LIST_ENTRY.Flink]
• PAGE:00786296      mov    ecx, [eax+LIST_ENTRY.Blink]
• PAGE:00786299      cmp    [edx+LIST_ENTRY.Blink], eax
• PAGE:0078629C      jnz    short @SecurityFailure
PAGE:0078629C
• PAGE:0078629E      cmp    [ecx+LIST_ENTRY.Flink], eax
• PAGE:007862A0      jnz    short @SecurityFailure
PAGE:007862A0
• PAGE:007862A2      mov    [ecx], edx
• PAGE:007862A4      mov    [edx+4], ecx
PAGE:007862A4
PAGE:007862A7
PAGE:007862A7 loc_7862A7:                                ; CODE XREF: IoUnregisterFileSystem(x)+2B↑j
• PAGE:007862A7      mov    esi, _IopFsNotifyChangeQueueHead
• PAGE:007862AD      jmp    short loc_7862BD
PAGE:007862AD
PAGE:007862AF
PAGE:007862AF
PAGE:007862AF @SecurityFailure:                          ; CODE XREF: IoUnregisterFileSystem(x)+35↑j
PAGE:007862AF                                          ; IoUnregisterFileSystem(x)+39↑j
• PAGE:007862AF      push   3
• PAGE:007862B1      pop    ecx
• PAGE:007862B2      int    29h                                ; KiRaiseSecurityFailure
PAGE:007862B2
```

- KiRaiseSecurityCheckFailure :
 - Int 0x29触发后调用到中断Handler
 - 简单处理后调用KiFastFailDispatch->KiBugCheck执行BSOD
- Bugcheck 代码 : 0x139 尚未文档化
 - 参数:ecx 错误ID
- 目前已知的错误ID:
 - 0x2: 内核驱动报告Security Cookie异常
 - 0x3: Safe unlinking / Safe linking异常
 - 0x6: 内核驱动Security Cookie未被初始化未高质量Cookie
 - 0x9: RtlQueryRegistryValuesEx注册表非可信(CVE-2010-4398)

- 在Windows8之前，内核和内核驱动使用ExAllocatePoolXXX只能分配出可执行的内核非分页内存
- 可执行的内核非分页内存可以用于内核溢出漏洞ROP
- Windows8开始，引入了新的PoolType:
 - NonPagedPoolNx
 - NonPagedPoolNxCacheAligned
 - NonPagedPoolSessionNx
- 通过ExAllocatePool(NonPagedPoolNx)分配出的内核内存不再可执行，在此类型内存池内执行代码将直接引发BSOD
- Windows8内核和驱动程序都已经使用此类型内存池替换所有原来使用的NonPagedPool类型

- 内核默认使用不可执行的非分页池的一个例子:
- IoAllocateDriverObjectExtension

```
__stdcall IoAllocateDriverObjectExtension(x, x, x, x) proc near
var_1          = byte ptr -1
DriverObject   = dword ptr  8
ClientIdentificationAddress= dword ptr  0Ch
DriverObjectExtensionSize= dword ptr  10h
DriverObjectExtension= dword ptr  14h

; FUNCTION CHUNK AT .text:004D4D37 SIZE 00000006 BYTES
; FUNCTION CHUNK AT .text:00566040 SIZE 00000015 BYTES
; FUNCTION CHUNK AT .text:0056605A SIZE 00000012 BYTES

mov     edi, edi
push   ebp
mov     ebp, esp
push   ecx
mov     eax, [ebp+DriverObjectExtensionSize]
push   edi
mov     edi, [ebp+DriverObjectExtension]
and     dword ptr [edi], 0
mov     [ebp+var_1], 0
cmp     eax, 0FFFFFFF7h
ja     loc_4D4D29

push   ebx
push   esi
push   'virD'          ; Tag
lea    ebx, [eax+8]
push   ebx            ; NumberOfBytes
push   NonPagedPoolNx ; PoolType
call   ExAllocatePoolWithTag(x,x,x)
```

- Intel® Secure Key技术 代号公牛山
- 在Intel 第三代Core处理器：Ivy Bridge中加入，今年4月正式发布
 - 提供硬件实现的底层数字化随机数生成器 (DRNG) 支持
 - 提供基于硬件的高性能、高质量的熵和随机数生成器
- 引入新的指令：RDRAND
- Windows8 内核开始使用该指令产生随机数，其中重要的使用者就是Security Cookie/ ASLR的生成过程
- 相关内核函数：ExGenRandom

- 过去针对内核随机数的攻击：Security Cookie预测 / ASLR暴力攻击
- Windows 8以前，Windows内核Security Cookie/ASLR使用系统时钟KeTickCount或RDTSC作为随机数源
- 结合获取模块加载时间，Security Cookie可被轻易预测，成功率达到46%以上 (j00ru)
- *J00ru: "Windows Kernel-mode GS Cookies subverted"*
- *H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh: "On the effectiveness of address-space randomization,"*
- Windows 8 内核自身使用Intel Secure Key技术生成的Security Cookie，并强制应用到所有加载的驱动程序中

- Windows 8在加载内核驱动时调用MiProcessLoadConfigForDriver，产生随机数并定位LoadConfig节内旧的Security Cookie，强制进行替换

```
PAGE:006F4053 ; __stdcall MiProcessLoadConfigForDriver(x)
PAGE:006F4053 _MiProcessLoadConfigForDriver@4 proc near
PAGE:006F4053 ; CODE XREF: MmLoadSystemImage(x,x,x,x,x,x)+265↑p
PAGE:006F4053 ; MiReloadBootLoadedDrivers(x)+360↓p
PAGE:006F4053     xor     eax, eax
PAGE:006F4055     call   _ExGenRandom@4 ; ExGenRandom(x)
PAGE:006F405A     push  eax
PAGE:006F405B     push  dword ptr [esi+20h]
PAGE:006F405E     mov   eax, [esi+18h]
PAGE:006F4061     call  _LdrInitSecurityCookie@16 ; LdrInitSecurityCookie(x,x,x,x)
PAGE:006F4066     retn
PAGE:006F4066 _MiProcessLoadConfigForDriver@4 endp
PAGE:006F4066
PAGE:006F4066
```

- 新的Win8驱动检查自身的Security Cookie是否被替换

```
INIT:00319643 ___security_init_cookie proc near ; CODE XREF: GsDriverEntry(x,x)+5↑p
INIT:00319643     xor     eax, eax
INIT:00319645     push  eax
INIT:00319646     push  eax
INIT:00319647     push  eax
INIT:00319648     push  eax
INIT:00319649     call  _ForceSEHExceptionHandler@16 ; ForceSEHExceptionHandler(x,x,x,x)
INIT:0031964E     mov   eax, ___security_cookie
INIT:00319653     test  eax, eax
INIT:00319655     jz    short loc_319666
INIT:00319657     cmp   eax, 0BB40E64Eh
INIT:0031965C     jz    short loc_319666
INIT:0031965E     not  eax
INIT:00319660     mov   ___security_cookie_complement, eax
INIT:00319665     retn
INIT:00319666 ; -----
INIT:00319666 loc_319666: ; CODE XREF: ___security_init_cookie+12↑j
INIT:00319666 ; ___security_init_cookie+19↑j
INIT:00319666     push  6
INIT:00319668     pop  ecx
INIT:00319669     int  29h ; Win8: RtlFailFast(ecx)
```

- Windows 7 内核/第三方驱动的Security Cookie的生成方式：
HalQueryRealTimeClock(from CMOS) ^ rdtsc
- Windows 8 内核Security Cookie的生成方式：
ExGenRandom-> ExpSecurityCookieRandomData ^ rdtsc
- Windows运行时内核ntoskrnl本身并不直接调用RDRAND指令
- ExGenRandom使用的随机数熵源来自OS加载器Winload.exe在启动过程中调用RDRAND指令的结果
 - Winload! OslpGatherRdrandEntropy
- 事实上OS加载器用五种方法试图获得综合的高质量随机数熵源:
- 外部熵(来自注册表)、TPM熵、实时时间熵、ACPI熵和RDRAND熵

- IDA Pro 6.3 已经加入了对RDRAND指令的解码支持
- 可以看到 winload 在启动过程中初始化SecureKey

```
.text:0040B8B5  
.text:0040B8B5 loc_40B8B5: ; CODE XREF: 0s1pGatherRdrandEntropy(x,x)+66↓j  
.text:0040B8B5 ; 0s1pGatherRdrandEntropy(x,x)+71↓j  
.text:0040B8B5 rdrand    edx  
.text:0040B8B8 jnb      short loc_40B8B5  
.text:0040B8BA mov      [ecx+eax*4], edx  
.text:0040B8BD inc      eax  
.text:0040B8BE cmp      eax, 1800h  
.text:0040B8C3 jb       short loc_40B8B5  
.text:0040B8C5 lea     esi, [edi+24h]  
.text:0040B8C8 push    6000h  
.text:0040B8CD mov     edx, ecx  
.text:0040B8CF call    @SymCryptSha512@12 ; SymCryptSha512(x,x,x)  
.text:0040B8D4 mov     esi, [ebp+arg_4]  
.text:0040B8D7 mov     edx, 6000h  
.text:0040B8DC mov     ecx, esi  
.text:0040B8DE call    @SymCryptWipeAasm@8 ; SymCryptWipeAasm(x,x)  
.text:0040B8E3 push    esi  
.text:0040B8E4 call    _BIMmFreeHeap@4 ; BIMmFreeHeap(x)  
.text:0040B8E9 xor     esi, esi  
.text:0040B8EB loc_40B8EB: ; CODE XREF: 0s1pGatherRdrandEntropy(x,x)+21↑j  
.text:0040B8EB ; 0s1pGatherRdrandEntropy(x,x)+3E↑j  
.text:0040B8EB mov     [edi+10h], ebx  
.text:0040B8EE mov     [edi+14h], esi  
.text:0040B8F1 pop     edi  
.text:0040B8F2 pop     esi  
.text:0040B8F3 pop     ebx  
.text:0040B8F4 pop     ebp  
.text:0040B8F5 retn   8  
.text:0040B8F5 _0s1pGatherRdrandEntropy@8 endp  
.text:0040B8F5
```

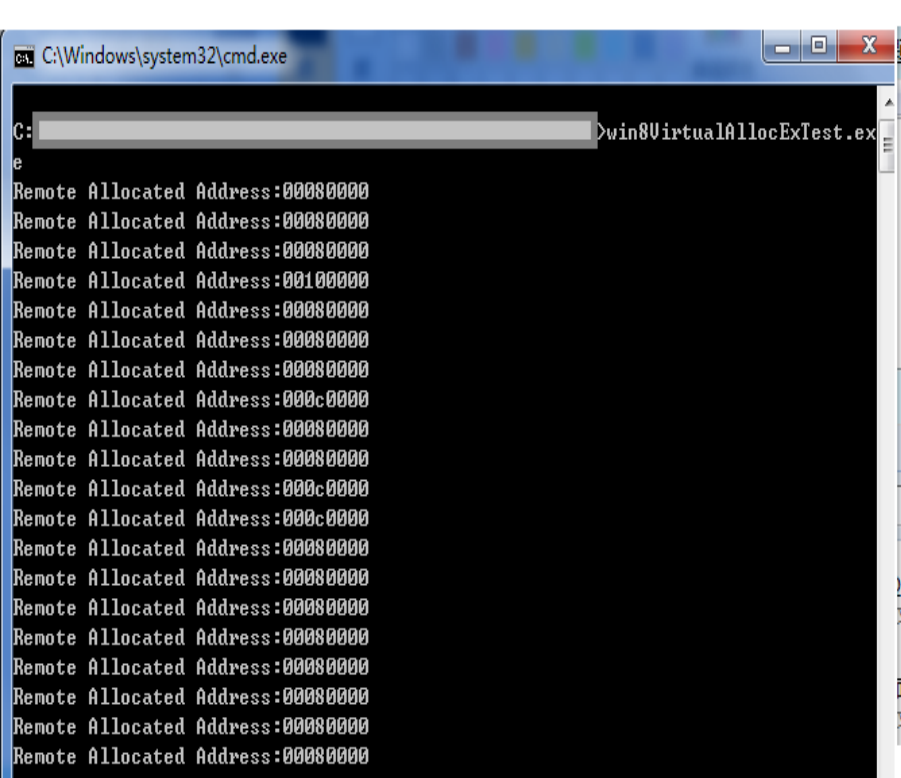
- ExGenRandom在内核其他的方面的使用：
 - 内存池配额Cookie
 - 内存池地址分配随机化
 - PEB/TEB地址随机化
 - 内核模块随机化
 - 线程栈地址空间和堆地址随机化
- 传递给用户态使用
 - Shared User Data->Cookie(ring3 Ldr*编解码)
 - 用户态地址分配随机化
 - 内存映射和镜像映射地址分配随机化

- *Guillaume: Bypassing ASLR and DEP on Adobe Reader X*
- Adobe Reader X 和Chrome的沙箱使用 VirtualAllocEx分配并放置 System Call Stub跳转代码
- Win7及以前的操作系统上VirtualAllocEx分配的地址未随机化，每次启动时有85%以上的几率落在固定地址
- 利用SystemCallStub跳转代码分配内配内存，绕过DEP+ASLR
- Windows8：进程启动时MmInitializeProcessAddressSpace调用 ExGenRandom的生成地址随机数种子
- NtAllocateVirtualMemory时使用MiSelectAddress通过已经生成的随机数熵来跳转随机地址进行分配

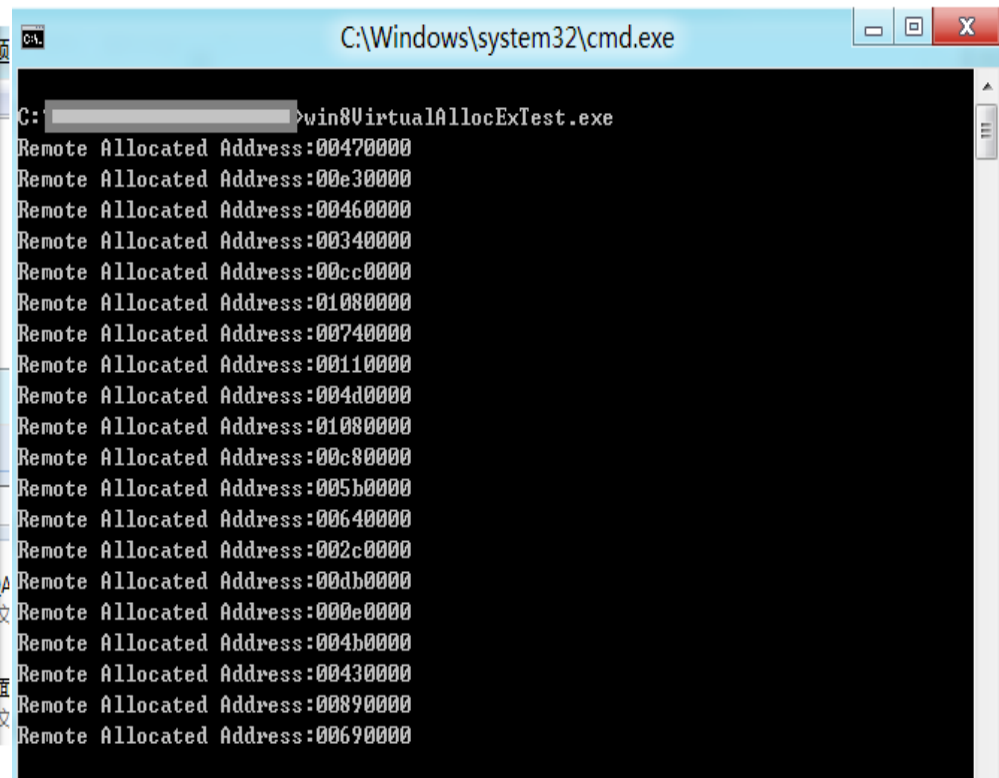
- Windows7和Windows8分配用户地址随机化的对比实验
- 分别启动20次 计算器(calc.exe)来分配远程内存

Windows7远程内存分配结果：有少量随机，但多数集中在固定的0x80000

Windows8远程内存分配结果：非常随机



```
C:\Windows\system32\cmd.exe
C: [redacted] >win8VirtualAllocExTest.exe
Remote Allocated Address:00000000
Remote Allocated Address:00000000
Remote Allocated Address:00000000
Remote Allocated Address:00100000
Remote Allocated Address:00080000
Remote Allocated Address:00000000
Remote Allocated Address:00000000
Remote Allocated Address:000c0000
Remote Allocated Address:00000000
Remote Allocated Address:00000000
Remote Allocated Address:000c0000
Remote Allocated Address:00000000
Remote Allocated Address:000c0000
Remote Allocated Address:00000000
Remote Allocated Address:00000000
Remote Allocated Address:00000000
Remote Allocated Address:00000000
Remote Allocated Address:00000000
Remote Allocated Address:00000000
Remote Allocated Address:00000000
```



```
C:\Windows\system32\cmd.exe
C: [redacted] >win8VirtualAllocExTest.exe
Remote Allocated Address:00470000
Remote Allocated Address:00e30000
Remote Allocated Address:00460000
Remote Allocated Address:00340000
Remote Allocated Address:00cc0000
Remote Allocated Address:01080000
Remote Allocated Address:00740000
Remote Allocated Address:00110000
Remote Allocated Address:004d0000
Remote Allocated Address:01080000
Remote Allocated Address:00c80000
Remote Allocated Address:005b0000
Remote Allocated Address:00640000
Remote Allocated Address:002c0000
Remote Allocated Address:00db0000
Remote Allocated Address:000e0000
Remote Allocated Address:004b0000
Remote Allocated Address:00430000
Remote Allocated Address:00890000
Remote Allocated Address:00690000
```

- SMEP : Supervisor-Mode Execution Prevention
- 内核模式执行保护机制，同样来自Intel第三代Core处理器
- Intel CPU提供的新保护机制，可以禁止从Ring0(Supervisor Mode)执行标记为Ring3(User Mode)地址空间的代码
- 背景：绝大多数内核漏洞攻击都使用一些技巧使内核代码跳转到预先放置在用户地址空间的 ShellCode
- 经典的技巧：
- 替换HalDispatchTable-> HalQuerySystemInformation
- 为什么将ShellCode放在Ring3：payload问题和地址随机化

- SMEP开启后：
 - Ring0下(CPL<3)的代码在存取指令时会检查页面的U/S标记，如果跳转到一个用户模式内存，会触发页面异常，Windows8下将处理此类型异常并触发BSOD
- 将cr4.SMEP(bit 20) 置1将开启此模式
- Windows 8 内核默认开启了SMEP，在内核初始化的第1阶段：
- Phase1Initialization-> Phase1InitializationDiscard -> KiInitMachineDependent

```
INIT:00928B50
INIT:00928B50
INIT:00928B50 0F 20 E0
INIT:00928B53 0D 00 00 10 00
INIT:00928B58 0F 22 E0
INIT:00928B5B E9 FB 4C FD FF
INIT:00928B5B
INIT:00928B5B
INIT:00928B60

@EnableSMEP:
mov     eax, cr4
or      eax, 100000h
mov     cr4, eax
jmp     loc_90085B

; CODE XREF: KiInitMachineDependent()+23E1j

; END OF FUNCTION CHUNK FOR _KiInitMachineDependent@0
```


- MI_CHECK_KERNEL_NOEXECUTE_FAULT
- Windows8在页面异常中断处理KiTrap0E 中用于处理两种不可执行的页面异常：非执行页面和SMEP
- 检测到若是支持SMEP的CPU上发生用户态页面的指令存取异常，则触发BSOD

```
1 int __userpurge MI_CHECK_KERNEL_NOEXECUTE_FAULT<eax>(int FaultStatus<eax>, ULONG_PTR ReasonId<edx>, ULONG_PTR VirtualAddress)
2 {
3     if ( FaultStatus & MmPaeErrMask )
4     {
5         if ( KeFeatureBits & KF_SMEP && MmPte & 4 ) // SMEP Enabled and PTE Owner = User
6         {
7             ReasonId |= REASON_ID_SMEP;
8         }
9         else
10        {
11            FaultStatus = HIDWORD(MmPte) & HIDWORD(MmPaeMask);
12            if ( !(MmPte & MmPaeMask) )
13                return FaultStatus;
14        }
15        KeBugCheckEx(ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY, VirtualAddress, MmPte, TrapInformation, ReasonId); // BSOD
16    }
17    return FaultStatus;
18 }
```

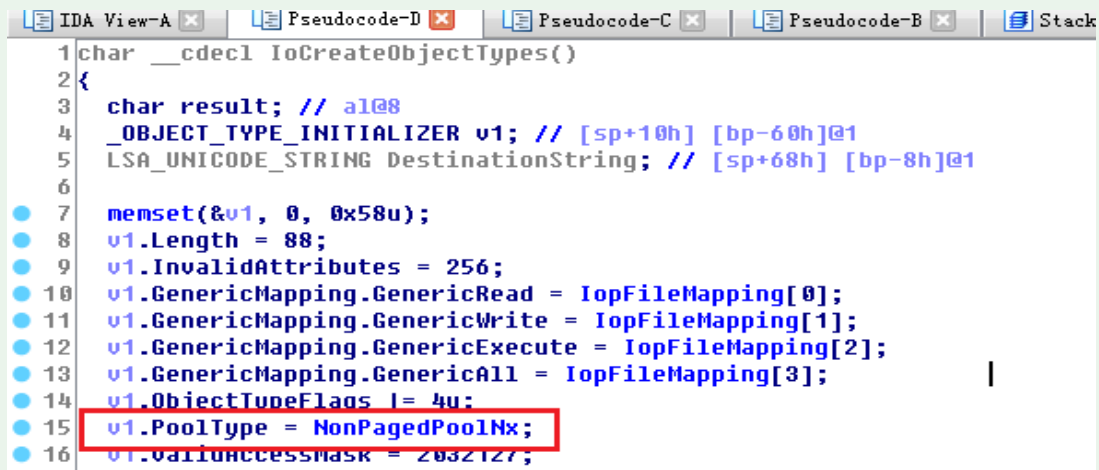
- 绕过SMEP的一个主意：通过确定的对象地址进行，向对象内存中放入一些代码
- NtQuerySystemInformation->SystemHandleInformation(Ex)

```
typedef struct _SYSTEM_HANDLE_TABLE_ENTRY_INFO {
    USHORT UniqueProcessId;
    USHORT CreatorBackTraceIndex;
    UCHAR ObjectTypeIndex;
    UCHAR HandleAttributes;
    USHORT HandleValue;
    PVOID Object;
    ULONG GrantedAccess;
} SYSTEM_HANDLE_TABLE_ENTRY_INFO, *PSYSTEM_HANDLE_TABLE_ENTRY_INFO;
```

- 可利用的对象：FileObject？

```
typedef struct _FILE_OBJECT {
    CSHORT Type;
    CSHORT Size;
    PDEVICE_OBJECT DeviceObject;
    PVOID Vpb;
    PVOID FsContext;
    PVOID FsContext2;
    PSECTION_OBJECT_POINTERS SectionObjectPointers;
    PVOID PrivateCacheMap;
    NTSTATUS FinalStatus;
    struct _FILE_OBJECT *RelatedFileObject;
    BOOLEAN LockOperation;
    BOOLEAN DeletePending;
    BOOLEAN ReadAccess;
    BOOLEAN WriteAccess;
    BOOLEAN DeleteAccess;
    BOOLEAN SharedRead;
    BOOLEAN SharedWrite;
    BOOLEAN SharedDelete;
    ULONG Flags;
    UNICODE_STRING FileName;
    LARGE_INTEGER CurrentByteOffset;
    ULONG Waiters;
    ULONG Busy;
    PVOID LastLock;
    KEVENT Lock;
    KEVENT Event;
    PIO_COMPLETION_CONTEXT CompletionContext;
} ?end _FILE_OBJECT ? FILE_OBJECT;
typedef struct _FILE_OBJECT *PFILE_OBJECT; // ntdis
```

- Windows8不可行：SMEP + NonPagedPoolNx
- 所有的对象内存都已经不可执行
- 对象内存的PoolType是由系统初始化时调用ObCreateObjectType时指定(IoCreateObjectTypes)
- Windows8上已经指定FileObject的PoolType为NonPagedPoolNx



```
IDA View-A x Pseudocode-D x Pseudocode-C x Pseudocode-B x Stack
1 char __cdecl IoCreateObjectTypes()
2 {
3     char result; // a1@8
4     _OBJECT_TYPE_INITIALIZER v1; // [sp+10h] [bp-60h]@1
5     LSA_UNICODE_STRING DestinationString; // [sp+68h] [bp-8h]@1
6
7     memset(&v1, 0, 0x58u);
8     v1.Length = 88;
9     v1.InvalidAttributes = 256;
10    v1.GenericMapping.GenericRead = IopFileMapping[0];
11    v1.GenericMapping.GenericWrite = IopFileMapping[1];
12    v1.GenericMapping.GenericExecute = IopFileMapping[2];
13    v1.GenericMapping.GenericAll = IopFileMapping[3];
14    v1.ObjectTypeFlags |= 4u;
15    v1.PoolType = NonPagedPoolNx;
16    v1.ObjectAccessMask = 2032127;
```

– Windows8 RP SMEP 已知攻击方式防御状况

| 攻击方式 | Windows 8防御方式 |
|--------------------------------|-----------------------------|
| SystemHandleInformation(Ex) | 内核对象非分页内存Nx |
| SystemLockInformation | Safe Linking/Unlinking |
| SystemModuleInformation | 模块数据区无保护/代码区写保护 |
| SystemExtendProcessInformation | 无保护 |
| GDT/IDT | 无保护 |
| 0xFFDF0000 (User Shared Data) | MiProtectKernelRegions 设置Nx |
| 0xFFC00000~0xFFFFFFFF(KPCR) | KPCR不固定地址/FFDFF000清空 |
| Win32k Shared Section | USER/内核对象分页内存Nx |

- Intel. [Intel® Digital Random Number Generator Software Implementation Guide](#)
- Intel. [Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 3A](#)
- J00ru . [Exploiting the otherwise non-exploitable:Windows Kernel-mode GS Cookies subverted](#)
- H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. [On the Effectiveness of Address-Space Randomization](#)
- Guillaume. [Bypassing ASLR and DEP on Adobe Reader X](#)

- 感谢:
- CHROOT Security Group
 - 360Safe MDT/HIPS Team