*УДК 004.451.87*

# БЕЗОПАСНОСТЬ СИСТЕМНЫХ ОБЪЕКТОВ В ОПЕРАЦИОННЫХ СИСТЕМАХ WINDOWS

О.П. СОЛОВЕЙ, А.В. БУДНИК

*Белорусский государственный университет информатики и радиоэлектроники*
*П. Бровки,6, Минск, 220013, Беларусь*

*Поступила в редакцию 18 сентября 2009*

Рассмотрены и проанализированы принципы обеспечения безопасности ядра в современных операционных системах. Указаны потенциальные уязвимости системы безопасности. Предложена программная реализация для доступа и изменения списка активных процессов на основе драйвера режима ядра.

*Ключевые слова:* уровни защиты, процесс, ядро, безопасность системного адресного пространства, драйвер режима ядра.

## Введение

В последнее время все чаще появляются публикации на тему безопасности современных операционных систем. Это связано с тем, что безопасность на сегодняшний день становится важнейшим потребительским качеством как системных средств, так и приложений.

Как известно, в архитектуре процессора Intel x86 определено четыре уровня привилегий, или колец (rings), предназначенных для защиты кода и данных системы от случайной или умышленной перезаписи с помощью кода, имеющего меньший уровень привелегий. Код может исполняться на одном из четырех уровней (колец) защиты. Наиболее привилегированным является нулевое кольцо, наименее привилегированным — третье. Windows использует всего два уровня привелегий: нулевой (или кольцо 0) для режима ядра и уровень привилегий 3 (или кольцо 3) для пользовательского режима. Это связано с тем, что на некоторых из ранее поддерживавшихся аппаратных платформ (например, Compaq Alpha и Silicon Graphics MIPS) реализовано лишь два уровня привилегий [1]. В нулевом кольце доступны привилегированные команды, порты ввода–вывода, и вся память. В других кольцах могут быть установлены другие правила: запрет некоторых команд, запрет ввода–вывода и.т.д. Между уровнями защиты можно переключаться только через специальные шлюзы, определенные в системных таблицах процессора (GDT, LDT, IDT). Доступ к памяти в защищенном режиме происходит только через селекторы находящиеся в этих таблицах, а у каждого селектора есть уровень привилегий, необходимый для его использования [2]. Подобная система позволяет изолировать код, выполняющийся на непривилегированных уровнях защиты, и полностью контролировать его исполнение с помощью кода нулевого кольца.

## Теоретический анализ

Любой Windows-процесс имеет свою выделенную изолированную область памяти. Код операционной системы и драйверы устройств, работающие в режиме ядра, делят единое виртуальное адресное пространство. Каждая страница в виртуальной памяти помечается тэгом, определяющим, в каком режиме должен работать процессор для чтения и/или записи данной страницы [3]. Страницы в системном пространстве доступны лишь в режиме ядра, а все страницы в пользовательском адресном пространстве — в пользовательском режиме. Страницы, предна-

значенные только для чтения (например, содержащие лишь исполняемый код), ни в каком режиме для записи недоступны.

Windows не предусматривает защиту системной памяти от компонентов, работающих в режиме ядра. Иначе говоря, код операционной системы и драйверов устройств в режиме ядра получает полный доступ к системной памяти и может обходить средства защиты Windows при обращении к любым объектам [4]. Поскольку основная часть кода Windows выполняется в режиме ядра, крайне важно, чтобы компоненты, работающие в этом режиме, были тщательно продуманы и протестированы.

Осуществляя манипуляции с процессами в ядре системы, следует четко представлять их внутреннюю организацию. Для описания процессов в системе используется структура EPROCESS. Данная структура имеет отличия в разных версиях Windows, поэтому целесообразно рассмотреть только основные ее части.

| EPROCESS | | |
|---|---|---|
| 0x000 | Pcb | Блок процесса ядра |
| … | | |
| 0x09X | UniqueProcessId | Уникальный идентификатор процесса |
| … | | |
| 0x0A0 | ActiveProcessLinks | Список активных процессов |
| … | | |
| 0x168 | ThreadListHead | Список потоков |
| … | | |
| 0x178 | ActiveThreads | Активные потоки |
| … | | |

Рис. 1. Структура EPROCESS Windows Server 2008 SP1

Совокупность процессов в операционной системе представляется двунаправленным кольцом. Следует обратить внимание на реализацию связей в этом кольце. Структура ActiveProcessLinks типа LIST_ENTRY является элементом кольца и содержит указатели FLink и Blink. Указатели ActiveProcessLinks указывают не на начало структуры EPROCESS, а на элементы списка, поэтому для получения указателя на следующую/предыдущую структуру EPROCESS необходимо от текущего указателя отнять смещение ActiveProcessLinks в структуре EPROCESS. В таблице представлены смещения ActiveProcessLinks и UniqueProcessId современных ОС.

**Некоторые значения смещений в структуре EPROCESS**

| Offsets | Windows 2000 | Windows XP | Windows Server 2003 SP2 R2 | Windows Vista | Windows Server 2008 SP1 |
|---|---|---|---|---|---|
| UniqueProcessId | 0x09C | 0x084 | 0x094 | 0x09C | 0x09C |
| ActiveProcessLinks | 0x0A0 | 0x088 | 0x098 | 0x0A0 | 0x0A0 |

```
typedef struct _LIST_ENTRY {
   struct _LIST_ENTRY *Flink; указывает на следующий элемент списка
   struct _LIST_ENTRY *Blink; указывает на предыдущий элемент списка
} LIST_ENTRY, *PLIST_ENTRY;
```

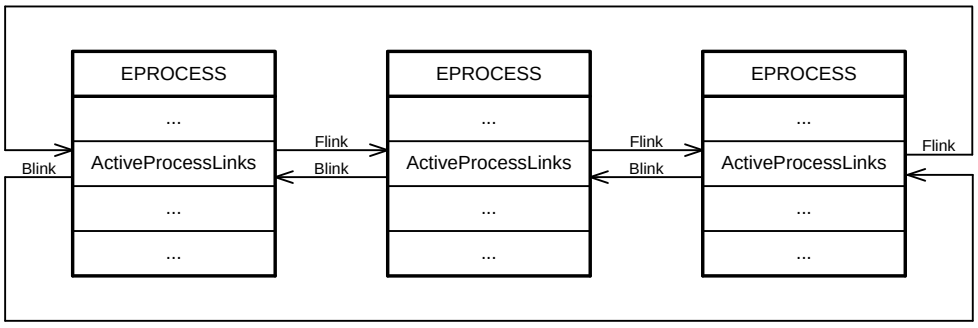


Рис. 2. Реализация связей в списке процессов

## Практическая реализация

Для написания драйвера режима ядра необходим комплект Microsoft Windows Driver Kit (WDK), предыдущие версии комплекта именовались Driver Development Kit. WDK — набор из средств разработки, заголовочных файлов, библиотек, утилит, который позволяет программистам создавать драйверы для устройств по определенной технологии или для определенной платформы. Хотя создание драйвера возможно и без использования WDK, в нем содержатся средства, упрощающие разработку драйвера, обеспечивающие совместимость драйвера с операционной системой, а также установку и тестирование драйвера.

Рассмотрим реализацию драйвера, выполняющего скрытие процесса по его PID.

Простейший драйвер включает в себя две функции:

```
VOID DriverUnload(IN PDRIVER_OBJECT driverObject) {…}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT driverObject, IN PUNICODE_STRING registryPath) {…}
```

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject,
IN PUNICODE_STRING RegistryPath)
{
   PVOID PageHandle;
   NTSTATUS st;
   PCWSTR dDeviceName                    = L"\\Device\\Ring0Port";
   PCWSTR dSymbolicLinkName  = L"\\DosDevices\\Ring0Port";
   PEPROCESS SysProc;

   PageHandle = MmLockPagableCodeSection(DriverEntry);
   MmLockPagableSectionByHandle(PageHandle);

   RtlInitUnicodeString(&DeviceName, dDeviceName);
   RtlInitUnicodeString(&SymbolicLinkName, dSymbolicLinkName);

   st = IoCreateDevice(DriverObject, 256, &DeviceName, FILE_DEVICE_NULL,
METHOD_NEITHER, FALSE, &deviceObject);

   if (st == STATUS_SUCCESS)
      st = IoCreateSymbolicLink(&SymbolicLinkName, &DeviceName);

   st = GetConsts();

   SysProc = PsGetCurrentProcess();
   PsActiveProcessHead = *(PVOID *)((PUCHAR)SysProc + ActivePsListOffset + 4);

   DriverObject–>MajorFunction[IRP_MJ_CREATE] =
   DriverObject–>MajorFunction[IRP_MJ_DEVICE_CONTROL] = DriverDispatcher;
   DriverObject–>MajorFunction[IRP_MJ_WRITE] = DriverDispatcher;
   DriverObject–>MajorFunction[IRP_MJ_CLOSE ] = DriverCreateClose;
   DriverObject–>DriverUnload = DriverUnload;

   return st;
}
```

Функция DriverEntry вызывается при загрузке драйвера в систему и содержит все действия по его инициализации. В качестве параметров ей передается указатель на объект драйвера и путь в реестре, по которому он будет прописан при установке. При использовании IoCreateDevice создается объект устройства, с которым будет работать реализуемый драйвер. Для использования имени в приложениях для открытия устройства при помощи IoCreateSymbolicLink создается символическая связь между устройством и его именем. Функция GetConsts инициализирует используемые переменные (смещения в структуре EPROCESS) в зависимости от версии ядра системы. Определения структур не предоставляются в открытом виде, при этом компания–разработчик предоставляет файлы символов (symbol files) для изучении внутренних структур данных ядра. Таким образом, при помощи отладчика ядра Microsoft

Windows Debugging Tools и файлов можно получить необходимую информацию. Следует отметить, что данные переменные являются зависимыми от версии ядра операционной системы (см. таблицу).

Головой двусвязного кольца процессов является ActiveProcessLinks процесса System. Это единственный обязательный процесс в системе, который существует постоянно после загрузки системы (имеет идентификатор 8 в Windows 2000, 4 — в Windows XP, Windows Server 2003, Windows Vista и Windows Server 2008). DriverEntry выполняется в контексте процесса System. Указатель на структуру текущего процесса можно получить при помощи функции PsGetCurrentProcess. Используя смещение ActiveProcessLinks в структуре EPROCESS, не составляет труда вычислить необходимый адрес.

При инициализации драйвера необходимо занести в объект драйвера адреса функций, которые будут вызываться системой в ответ на определенные действия: получение дескриптора драйвера функцией CreateFile, запрос к драйверу на выполнение требуемого действия функцией DeviceIoControl и закрытие драйвера функцией CloseHandle. Для хранения адресов основных функций в объекте драйвера предусмотрен массив указателей на функции типа PDRIVERDISPATCH MajorFunction.

Исходя из вышеизложенного можно заключить, что для сокрытия процесса, необходимо получить указатель на его врутреннюю структуру ActiveProcessLinks и перенаправить указатели Flink и Blink вобход интересуемого процесса. В результате при перечислении процессов системой данный процесс отображен не будет (рис. 3). Необходимо отметить, что планировщик Windows не нуждается в информации о процессах, он всего лишь распределяет процессорное время между всеми существующими потоками независимо от их принадлежности какому-либо процессу.

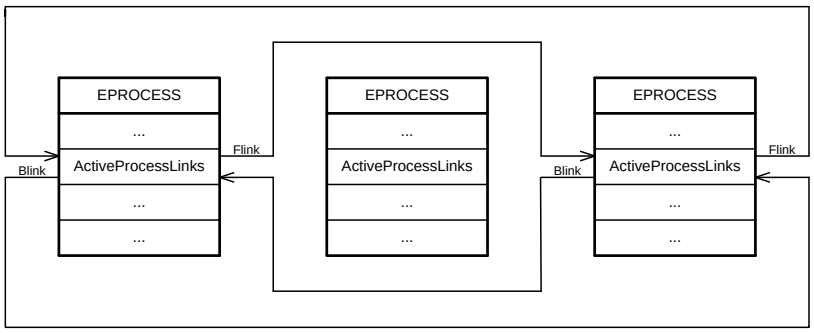


Рис. 3. Механизм сокрытия процессов.
Перенаправление связей в списке активных процессов

Для восстановления исходного состояния списка целесообразно сохранять указатели на структуры ActiveProcessLinks скрываемых процессов. Ниже приведен пример реализации, в котором список processQueue содержит указатели на ActiveProcessLinks скрытых процессов.

```
typedef struct _ProcessQueue
{
   PVOID NextItem;
   ULONG PId;
   PLIST_ENTRY pActiveProcessLinks;
} TProcessQueue, *PProcessQueue;

PProcessQueue processQueue = NULL;

VOID HideProcess(IN ULONG pId)
{
   PLIST_ENTRY pActiveProcessLinks = NULL;

   if(Exists(processQueue, pId) == FALSE){
      pActiveProcessLinks = GetActiveProcessLinks(PId);
   if(pActiveProcessLinks != NULL)
   {
      Add(&processQueue, pId, pActiveProcessLinks);

      pActiveProcessLinks–>Flink–>Blink = pActiveProcessLinks–>Blink;
      pActiveProcessLinks–>Blink–>Flink = pActiveProcessLinks–>Flink;
   }
}
```

## Заключение

Таким образом, современные операционные системы даже в силу своих конструктивных особенностей не лишены уязвимостей. В статье показано, что на основе драйвера режима ядра можно получить доступ к внутренним структурам ядра и манипулировать ими. Приведенный пример иллюстрирует лишь некоторые возможности драйвера режима ядра. Драйверы такого типа могут использоваться и в злоумышленных целях. Следовательно, необходимо быть осторожным при загрузке драйвера устройства от стороннего поставщика: перейдя в режим ядра, он получит полный доступ ко всем данным операционной системы. Одним из возможных вариантов обеспечения безопасности системы от такого рода уязвимости может служить установка хука (функции–ловушки) на функцию загрузки драйвера, которая будет уведомлять пользователя о ее вызове.

# WINDOWS SYSTEM OBJECTS SECURITY

O.P. SOLOVEY, A.V. BUDNIK

## Abstract

Principles of security in modern operating systems kernels are considered and analyzed. Potential vulnerability of security system is specified. Software implementation for access and modifying of the list of active processes of the kernel mode driver is offered.

## Литература

1. *Руссинович М., Соломон Д.* Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP и Windows 2000. СПб., 2008.
2. *Hoglund G., Butler J.* Rootkits: Subverting the Windows Kernel. Addison-Wesley Professional. 2005.
3. *Рихтер Дж.* Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. СПб., 2001.
4. *Schreiber S.* Undocumented Windows 2000 Secrets: A Programmer's Cookbook. Addison-Wesley Professional. 2001.