Информационные процессы, Том 11, № 3, 2011, стр. 413–421

Сокрытие информации о процессах в ОС Windows

Д. П. Бредихин

Южный федеральный университет, Ростов-на-Дону, Россия

21.06.2011

Аннотация—В работе представлено исследование возможностей глубокого скрытия процессов в операционных системах семейства Windows NT 5, целью которого является определение минимума данных в ядре ОС, необходимых для корректной работы процесса. Рассматривается реализация эффективной методики поиска скрытых процессов на основе анализа этих данных.

1. ВВЕДЕНИЕ

Технологии скрытия объектов в операционной системе всегда являлись предметом интереса как специалистов по информационной безопасности, так и создателей различного вредоносного программного обеспечения. Разработка средств, использующих в своей работе т.н. руткит-технологии, требует иногда достаточно глубокого вмешательства во внутренние механизмы операционной системы. В последние годы сфера применения технологий, на основе которых строятся руткиты, то есть программные средства, обеспечивающие скрытое присутствие атакующего на целевой машине, значительно расширилась. Большая часть серьезных вредоносных и шпионских программ скрывают свое присутствие и деятельность, что делает проблему анализа методов скрытия внутренних объектов операционной системы очень актуальной. В настоящее время существует большое количество как концептуальных утилит, демонстрирующих различные технологии внедрения в ядро операционной системы, так и решений в области обнаружения скрытых объектов [1–4]. Теоретически, полностью скрыть некоторый объект в ядре ОС нельзя, так как в этом случае любая работа с ним станет невозможной. Однако каждый объект имеет немало ссылок и записей о себе в различных системных структурах данных. В настоящее время исследователи работают над созданием концепций скрытия объектов, оставляющих как можно меньше следов существования этих самых объектов [5]. Такие концепции основаны на перехвате и модификации системных функций и структур данных ядра.

Наибольшую опасность представляют скрытые процессы, так как они по сути являются неизвестным кодом, неизвестно кем и с неизвестными целями запущенным на машине. В настоящей работе предлагается исследование возможностей глубокого скрытия процессов в операционных системах семейства Windows NT 5 (далее Windows) с целью определения минимума данных в ядре ОС, необходимых для корректной работы процесса. На основе анализа именно этих данных возможно создание эффективной методики поиска скрытых процессов.

2. ОБЗОР БАЗОВЫХ МЕТОДИК

2.1. Перехват функций

Не составляет труда скрыть процесс от базовых средств системного мониторинга, таких как Диспетчер задач или Process Explorer [6]. Указанные утилиты получают информацию о работающих процессах через API-функции режима пользователя, и поэтому для достижения

цели стоит только перехватить эти функции, модифицировав при этом возвращаемые ими данные. Существует целый рад способов, позволяющих осуществить перехват. Это и модификация таблицы импорта нужных приложений, и непосредственное изменение кода самих функций (т.н. inline-перехват) [7, с. 76–85]. Однако вышеуказанные методы, являясь достаточно простыми в реализации, также отличаются и простотой обнаружения и нейтрализации. К тому же, любое обращение к системным структурам на более низком уровне приведет к обнаружению скрытого процесса.

Гораздо больший интерес представляют концепции, основанные на внедрении в ядро операционной системы [7, с. 85–87]. Таким образом, код захвата оказывается на одном уровне с низкоуровневыми модулями специальных средств мониторинга. Ядро расположено в верхней части виртуальных адресов оперативной памяти. В компьютерах архитектуры Intel x86 оно обычно располагается, начиная с адреса 0x80000000 (0xC0000000 при загрузке с ключом `/3GB`). Как правило, процессы не имеют доступа к памяти ядра. Обсуждение способов обхода такой защиты [7, с. 87–89] лежит за рамками данной работы. Для доступа к памяти ядра используется драйвер устройства.

Исполнительная система Windows работает в режиме ядра и предоставляет службы поддержки для всех подсистем окружения. Адреса этих служб перечислены в структуре ядра, называемой таблицей диспетчеризации системных служб (System Service Dispatch Table, SSDT). Имея номер службы, можно получить ее адрес [8, с. 128–133]. В Windows используется функция `ZwQuerySystemInformation`, предоставляющая различную информацию о системе, в том числе и список запущенных процессов. Многие средства мониторинга, работающие в пользовательском режиме, задействуют эту функцию. Можно подменить эту функцию в SSDT и получить возможность фильтровать возвращаемую ею информацию.

Процесс, скрытый таким образом, остается невидимым для многих средств мониторинга, работающих в режиме пользователя. Аналогичным образом можно произвести захват таблицы дескрипторов прерываний (IDT) или таблицы IRP-функций драйвера. Однако сам факт такого захвата легко обнаруживается многими защитными утилитами [9], да и просмотр низкоуровневых структур ядра выявляет скрытие процесса. К тому же, современные технологии защиты памяти ядра могут затруднить осуществление перехвата. Поэтому более подробно останавливаться на этой концепции не будем.

2.2. Манипулирование объектами ядра

Рассмотрим технологию, названную DKOM — Direct Kernel Object Manipulation, непосредственное манипулирование объектами ядра [10]. В расшифровке данного термина под словом «объект» понимается некоторая структура ядра. Технология имеет как достоинства, так и недостатки. С одной стороны, ее применение трудно обнаружить, особенно, если манипуляции также затрагивают структуры диспетчера объектов. С другой стороны, технология достаточно хрупкая и детали ее реализации различны для различных версий, а порой даже и сборок ядра. От разработчика требуется точное знание многих внутренних структур ядра и системных механизмов, часто недокументированных, а также получение адресов многих неэкспортируемых системных функций или переменных. Технологию DKOM можно использовать, например, для скрытия процессов, драйверов устройств, портов, а также для манипуляций с привилегиями процессов и потоков.

Операционная система Windows хранит учетную информацию в оперативной памяти в виде структур, или объектов. Windows получает список запущенных процессов обходом кольцевого двусвязного списка `PsActiveProcesses`, указанного в структуре `EPROCESS` каждого процесса. Каждая структура `EPROCESS` содержит в себе структуру `LIST_ENTRY` с полями `Flink` и `Blink`, указывающими соответственно на следующий и предыдущий элементы списка. Фор-

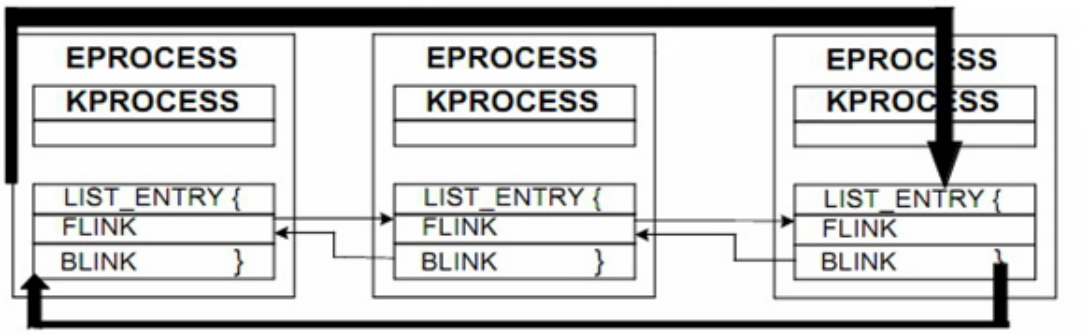

Рис. 1. Исключение системного объекта из списка

мат структуры `EPROCESS` различен для различных версий системы. Поиск структуры в памяти осуществляется вызовом функции `PsGetCurrentProcess`, возвращающей указатель на структуру `EPROCESS` текущего процесса. Идентификацию целевого процесса можно проводить по его имени или PID, для чего необходимо знать смещения соответствующих полей в структуре `EPROCESS`.

После получения адреса структуры `EPROCESS` целевого процесса, чтобы скрыть его, необходимо изменить поле `Flink` предыдущего процесса и поле `Blink` следующего процесса, чтобы целевой процесс был исключен из списка. Полю `Blink` следующего процесса присваивается значение поля `Blink` целевого процесса, а полю `Flink` предыдущего процесса присваивается значение поля `Flink` целевого процесса. Значения полей `Blink` и `Flink` целевого процесса меняются так, чтобы они указывали сами на себя. Последнее необходимо для того, чтобы стабилизировать систему в момент завершения целевого процесса. При уничтожении процесса функцией `PspExitProcess` обновляется двусвязный список. Если за время скрытого существования целевого процесса один из его «соседей» по списку завершится, поля `Flink` и `Blink` целевого процесса не обновятся, поэтому при его завершении они могут уже не указывать на существующий процесс и даже на выделенную память. Также следует помнить, что если вышеуказанные манипуляции совершаются на многопроцессорной системе, то пока происходит переход от одного элемента списка к другому, может быть создан новый процесс или уничтожен один из существующих. В этом случае для безопасного прохода списка процессов необходимо повысить IRQL всех процессоров на период выполнения манипуляций со списками до `DISPATCH_LEVEL`. Поток, выполняющийся на уровне `DISPATCH_LEVEL`, не может быть вытеснен.

При рассмотрении исключения записи о процессе из списка `PsActiveProcesses` может возникнуть вопрос, не лишится ли скрытый процесс квантов процессорного времени. Ответ прост, данные манипуляции не затрагивают системный планировщик, так как в Windows объектами планирования являются не процессы, а потоки, описываемые в ядре структурами `ETHREAD`.

Также следует отметить, что по вышеописанной технологии производится модификация, вообще говоря, любых внутренних списков ядра, в дальнейшем это не упоминается, но подразумевается.

3. КОНЦЕПЦИЯ ГЛУБОКОГО СКРЫТИЯ ПРОЦЕССОВ

Как известно, для разработки эффективной концепции защиты необходимо построить модель наиболее «сильной» атаки, выяснить, насколько глубоко возможно внедриться в систему. В рамках настоящей статьи рассматриваются возможные методики исключения информации

о работающих процессах из различных подсистем ядра ОС Windows XP, а также предлагается возможная концепция глубокого скрытия процесса в системе. Определяется минимальный объем информации о процессе, необходимый для его корректной работы, а затем определяется наиболее эффективное семейство методов поиска скрытых процессов.

3.1. Противодействие базовым методикам

Несмотря на кажущуюся сложность детектирования скрытых с использованием технологии DKOM процессов, список `PsActiveProcesses` является далеко не единственным местом, откуда можно почерпнуть информацию о работающих в системе процессах. Даже некоторые методы обнаружения, работающие в режиме пользователя, позволяют обнаружить скрытый с помощью DKOM процесс.

Каждый работающий в системе процесс имеет целый ряд косвенных признаков своего существования, по которым его можно обнаружить. Это могут быть созданные им дескрипторы, окна, некоторые другие системные объекты. Для разработки метода глубоко скрытия процесса в системе необходимо учесть все побочные проявления работы процесса, что сделать не очень легко. Одна из концепций поиска скрытых процессов заключается в построении двух списков процессов: «высокоуровневого», полученного с помощью функций API, и «низкоуровневого», полученного анализом косвенных признаков Работы процессов, и сравнении этих списков.

Даже если запись о процессе исключена из списка `PsActiveProcess`, перечислив, например, открытые дескрипторы через `ZwQuerySystemInformation`, мы можем построить полный список процессов. Также, дескрипторы работающих процессов имеются в таблице дескрипторов процесса csrss.exe, являющегося их супервизором. Получить список процессов, имеющих окна, можно по результатам анализа зарегистрированных в системе окон. Применение сразу нескольких из вышеуказанных методик достаточно эффективно в плане обнаружения скрытых процессов. Однако основным недостатком всех вышеописанных путей построения списка процессов является то, что при желании можно скорректировать соответствующие списки структур в ядре без особого ущерба для работы системы и целевого процесса.

3.2. Диспетчер объектов

Как было сказано ранее, работающий процесс оставляет в системе немало следов своей работы. С каждым процессом связано немало системных объектов, таких как потоки, дескрипторы и т.п. С этими объектами связаны соответствующие структуры ядра. С помощью модификации списков некоторых из этих структур с применением технологии DKOM можно сократить количество информации, которое средства мониторинга смогут получить о процессе.

Более подробно следует остановиться на таблице дескрипторов, являющейся частью диспетчера объектов. Диспетчер объектов - компонент исполнительной системы, отвечающий за создание, удаление, защиту и отслеживание объектов. Реализованная в Windows модель объектов позволяет получить согласованный и безопасный доступ к различным внутренним сервисам исполнительной системы. Различают два типа объектов: объекты исполнительной системы и объекты ядра, нас будут интересовать только первые.

Кратко рассмотрим структуру объектов исполнительной системы (далее объектов) и работу диспетчера объектов с таблицами дескрипторов (описателей, хэндлов). Каждый объект состоит из заголовка и тела. Заголовками объектов управляет диспетчер объектов, а телами — владеющие компоненты исполнительной системы. Заголовок объекта содержит указатели на список процессов, которые открыли объект, а также на специальный объект — объект типа, содержащий информацию, специфичную для конкретного типа объектов. Подробное описание

принципов работы диспетчера объектов Windows можно получить в официальных изданиях Microsoft [8, с. 133–145].

Когда процесс создает или открывает объект по имени, он получает дескриптор (описатель, handle), который дает ему доступ к объекту. Дескрипторы служат косвенными указателями на объекты, позволяющими пользовательским программам избегать прямого взаимодействия с системными структурами данных. Дескриптор объекта представляет собой индекс в таблице дескрипторов, принадлежащей процессу (структура `HANDLE_TABLE`). Указатель на нее содержится в структуре блока процесса исполнительной системы (`EPROCESS`). Таблица содержит указатели на все объекты, дескрипторы которых открыты процессом.

Для удобства перечисления дескрипторов, все таблицы дескрипторов объединены в двусвязный список `HandleTableList`, указатель на который содержится в структуре `HANDLE_TABLE`. Наряду с этим, вышеуказанная структура содержит и указатель на владеющий ей процесс (`QuotaProcess`). Пройдя по списку таблиц дескрипторов, мы можем построить список процессов. Следовательно, для предотвращения обнаружения процесса, исключенного из списка `PsActiveProcesses`, необходимо исключить таблицу дескрипторов целевого процесса из списка `HandleTableList`, что реализуется аналогично исключению структуры процесса из списка `PsActiveProcesses`.

Таблицы дескрипторов реализованы по трехуровневой схеме аналогично таблицам страничной адресации памяти [8, с. 145–152]. Таким образом, на каждый процесс поддерживается более 16 миллионов дескрипторов. Следует помнить, что форматы таблиц дескрипторов могут отличаться в различных версиях ОС, в настоящей работе рассматривается Windows XP. При создании процесса в Windows XP диспетчер объектов формирует только таблицу нижнего уровня, остальные создаются при необходимости. Эта таблица вмещает столько элементов, сколько помещается на страницу памяти минус один элемент. Каждый элемент таблицы дескрипторов состоит из структуры `HANDLE_TABLE_ENTRY` с двумя 32-битными указателями: указателем на объект и маской доступа. Таким образом, таблица на любом уровне может вмещать до 511 записей.

Механизм хранения объектов в памяти основан на учете числа открытых дескрипторов, а также числа ссылок на объект [8, с. 152–154]. Поэтому при удалении описателя на объект необходимо также уменьшить значение поля `TotalNumberOfObjects` в структуре объекта типа, содержащего информацию о числе активных объектов данного типа.

Чтобы скрыть процесс от обнаружения с использованием анализа таблиц дескрипторов, необходимо удалить ссылки на скрываемый процесс из таблиц. Предварительно следует определить число уровней таблиц дескрипторов и провести трансляцию таблицы для получения доступа непосредственно к записям об объектах. Поле `TableCode` структуры `HANDLE_TABLE` в двух младших битах содержит данные о числе уровней таблицы (от одного до трех), остальные биты представляют собой указатель на таблицу первого уровня. Непосредственно исключение элемента из таблицы реализуется путем обнуления значения поля `Object` (указатель на объект) в структуре `HANDLE_TABLE_ENTRY` — элементе таблицы дескрипторов, — соответствующей целевому процессу. Для получения правильного адреса объекта необходимо сбросить младший бит в поле указателя на объект, являющийся флагом блокировки объекта.

Как было отмечено ранее, процесс ccrss.exe исполняет роль супервизора для процессов пользовательского режима. Для продолжения работы все Win32 процессы должны информировать CSRSS о своём создании на стадии инициализации. Таким образом, таблица дескрипторов этого процесса содержит записи о работающих в системе пользовательских процессах. Существуют утилиты для обнаружения скрытых процессов, работа которых основана на анализе таблицы дескрипторов процесса csrss.exe [11]. Отсюда следует, что, исключение записей о целевом процессе из таблицы дескрипторов процесса csrss.exe уменьшит шансы быть обнаруженным.

Также в целях дополнительной маскировки можно провести замену PID и имени образа скрываемого процесса. Это уменьшит шансы обнаружения сигнатурными методами. Аналогичным образом можно произвести манипуляции и со списками CSR_PROCESS и CSR_THREAD, также служащими источником информации о запущенных процессах. Побочным негативным эффектом применения вышеописанного метода является то, что скрытый процесс не сможет порождать дочерние процессы.

3.3. Таблица PspCidTable

Многие специальные утилиты для обнаружения скрытых процессов используют методы, основанные на сканировании PspCidTable — таблицы дескрипторов процессов и потоков [12]. Одним из таких методов является т.н. PID Bruteforce, то есть вызов OpenProcess для всех возможных значений идентификатора процесса. Даже после исключения процесса из списка PsActiveProcesses ничто не мешает его открытию с помощью OpenProcess. Это говорит о том, что при открытии процесса его поиск ведется по списку, отличному от PsActiveProcesses. При переборе PID обнаруживается еще одна особенность: по нескольким различным PID могут быть открыты процессы с одинаковым исполняемым образом, что наводит на мысль о том, что используемая таблица процессов может представлять собой HANDLE_TABLE. Рассмотрим данный вопрос более подробно.

Функция OpenProcess является оберткой над NtOpenProcess. NtOpenProcess реализована в ядре, в ntoskrnl.exe.

NtOpenProcess выполняет три основных действия:

1) Вызовом PsLookupProcessByProcessId проверяет, существует ли процесс.
2) Пытает получить дескриптор (handle) процесса вызовом ObOpenObjectByPointer.
3) Если дескриптор был получен успешно, он возвращается вызывающей подпрограмме.

Рассмотрим начало исполняемого кода функции PsLookupProcessByProcessId [13]:

```
push    ebp
mov     ebp, esp
push    ebx
push    esi
mov     eax, large fs:124h
push    [ebp+arg_0]
mov     esi, eax
dec     dword ptr [esi+0D4h]
push    _PspCidTable
call    _ExMapHandleToPointer
```

В теле функции NtOpenProcess происходит вызов функции PsLookupProcessByProcessId, которая обращается к PspCidTable — таблице дескрипторов процессов и потоков. Это подтверждает наличие второй таблицы процессов, организованной как HANDLE_TABLE. Как мы видим, дескриптор и указатель на таблицу дескрипторов передаются функции ExMapHandleToPointer, возвращающей указатель на соответствующий элемент таблицы — HANDLE_TABLE_ENTRY. Таким образом, функция ExMapHandleToPointer ищет соответствующий идентификатор процесса в PspCidTable.

Так как PspCidTable предназначена для слежения за всеми процессами и потоками, было бы логично, если бы методика обнаружения скрытых процессов включала сканирование

PspCidTable для их поиска. Отсюда возникает новая задача: для более глубокого скрытия процесса необходимо удалить записи о нем из данной таблицы.

Каждый дескриптор процесса занимает свое место в PspCidTable, а PspCidTable является указателем на структуру HANDLE_TABLE. Дальнейшая работа осуществляется аналогично исключению дескриптора процесса из таблицы дескрипторов, описанному в предыдущей части статьи.

Однако каким образом программа может динамически определять адрес PspCidTable, который, как и адреса функций для работы с данной таблицей, не экспортируется ядром? Обозначенная проблема порождает задачу нахождения адресов неэкспортируемых системных переменных. Явное указание адресов в коде не является лучшим решением. Возможно встраивание в программу дизассемблера для динамического поиска нужных адресов. Однако Edgar Barbosa предложил иной метод поиска неэкспортируемых переменных [14]. Поле KdVersionBlock в структуре Области Управления Процессом (Process Control Region) указывает на структуру KDDEBUGGER_DATA32, содержащую указатели на многие часто используемые неэкспортируемые переменные.

Также следует отметить еще одну проблему, возникающую при удалении записи о скрываемом процессе из PspCidTable. Когда объект удаляется из таблицы, соответствующее поле заполняется нулями, что означает, что процесс не существует. Когда процесс, не имеющий записи в PspCidTable, завершает работу, система, ссылаясь на соответствующий элемент таблицы, пытается разыменовать нулевой объект, что вызывает крах системы. Для разрешения данной проблемы вызовом PsSetCreateProcessNotifyRoutine регистрируется callback-функция, вызывающаяся всякий раз при завершении процесса и восстанавливающая запись о скрытом процессе в PspCidTable.

4. ОБНАРУЖЕНИЕ СКРЫТЫХ ПРОЦЕССОВ

Ранее уже упоминалось, что полностью скрыть присутствие процесса в системе невозможно, так как в случае, например, исключения информации о потоках процесса из структур данных планировщика процесс лишится квантов процессорного времени. В предыдущей части статьи было продемонстрировано, что в операционных системах семейства Windows NT 5 возможна модификация немалого количества служебной информации о запущенном процессе, по которой возможен его поиск. При этом процесс корректно работает. Единственными данными, необходимыми для работы процесса в Windows XP, являются записи о потоках процесса в списках планировщика. Именно на основе анализа этих данных возможно построение списка процессов, с наибольшей вероятностью содержащего все работающие в системе процессы.

Несколько лет назад была предложена концепция [15], позволяющая удалить записи о потоках скрываемого процесса из списков планировщика. Ее суть заключается в создании собственного планировщика, который будет планировать только скрываемые потоки, и подвязке его работы с существующим системным планировщиком. Однако в силу особенностей реализации данная концепция по сути лишь глубоко маскирует скрываемые потоки. При жесткой политике безопасности поток созданного «специального» планировщика может быть обнаружен и классифицирован как подозрительный. Ко всему прочему, реализация связана с глубоким вмешательством в код ядра и поиском множества неэкспортируемых данных. Поэтому, даже учитывая вышесказанное, наибольшую эффективность имеют методики поиска скрытых процессов основанные на анализе внутренних структур планировщика.

В Windows XP планировщик имеет два списка потоков: KiWaitListHead, содержащий потоки, ожидающие наступления события, и KiDispatcherReadyListHead, содержащий потоки, готовые к исполнению. Пройдя по спискам потоков, можно получить указатели на соответствующие структуры ETHREAD, а по ним определить владеющие процессы. При реализации

данной методики возникает необходимость динамически, сканированием кода функций по шаблону, определять адреса указателей на списки потоков. Адрес `KiWaitListHead` содержится в функции `KeDelayExecutionThread`, а адрес `KiDispatcherReadyListHead` — в функции `KiDispatchInterrupt`.

Также одним из самых эффективных методов обнаружения скрытых процессов является перехват функции `SwapContext`, выполняющей переключение потоков. Функция является неэкспортируемой, ее адрес можно получить, анализируя код функции `KiDispatchInterrupt`. Единственная возможность перехвата - замена первых байт кода функции. Параметры функции передаются в регистрах и содержат указатели на переключающиеся потоки. Анализ данных параметров в комплексе с работой со списками планировщика лежит в основе одного из самых эффективных способов поиска скрытых процессов и потоков.

5. ЗАКЛЮЧЕНИЕ

В рамках настоящей работы был предложен и программно реализован метод, демонстрирующий комплексный подход к скрытию процессов. Были реализованы следующие методики: исключение записи о целевом процессе из списка `PsActiveProcesses`, исключение таблицы дескрипторов целевого процесса из списка таблиц дескрипторов, корректировка структур диспетчера объектов, исключение записи о целевом процессе и его потоках из таблицы дескрипторов процессов и потоков `PspCidTable`, исключение записей о целевом процессе и его потоках из таблицы дескрипторов процесса csrss.exe, корректировка списков `CSR_PROCESS` и `CSR_THREAD`, маскировка структур данных в памяти, корректировка списков окон в системе. Скрытый таким образом процесс не обнаруживается многими специализированными утилитами. В ядре системы остаются только записи о потоках скрытого процесса в двух списках планировщика. Это минимальные данные, необходимые для работы процесса, их корректировка приводит к прекращению работы скрытого процесса.

Против разработанной концепции были реализованы и применены различные методики поиска скрытых процессов. Результаты тестирования подтверждают предположение о наиболее эффективной методике обнаружения скрытых процессов. Только лишь с помощью анализа структур планировщика удалось обнаружить потоки скрытого по вышеописанной методике процесса. Итоги сравнительного тестирования программного обеспечения для поиска скрытых процессов подтвердили результаты анализа методик обнаружения. Эффективными оказались только средства, использующие данные планировщика.

СПИСОК ЛИТЕРАТУРЫ

1. Tuluka Kernel Inspector, `http://www.tuluka.org/tlk/Tuluka_v1.0.394.77.zip`
2. FU Rootkit, `http://www.f-secure.com/v-descs/fu.shtm`
3. IceSword, `http://mail.ustc.edu.cn/~jfpan/download/IceSword122en.zip`
4. Hacker Defender, `http://www.hacker-soft.net/tools/Defense/hxdef100r.zip`
5. Sparks S. Butler J. „Shadow Walker“Raising the bar for rootkit detection, `http://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf`
6. HideToolz, `http://www.wasm.ru/tools/4/HideToolz.zip`
7. Хоглунд Г, Батлер Д. *Руткиты: внедрение в ядро Windows.* М.: Питер, 2007
8. Руссинович М. Соломон Д. *Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP и Winsows 2000.* М.: Питер, 2006
9. Rootkit Unhooker, `http://forum.sysinternals.com/uploads/20071210_182632_rku37300509.rar`

10. Butler J. DKOM (Direct Kernel Object Manipulation)
http://www.blackhat.com/presentations/bh-europe-04/bh-eu-04-butler.pdf

11. CsrWalker Rootkit Detector,
http://forum.sysinternals.com/csrwalker-processes-detection-from-user-mode_topic15457.html

12. Nanavati M. Hidden processes detection using PspCidTable.
http://helios.miel-labs.com/downloads/process_scan.pdf

13. Silberman P. FUTo. *Uninformed*, 2006, vol. 3, article 7, pp. 5 - 6,
http://uninformed.org/index.cgi?v=3&a=7&t=pdf

14. Finding Kernel Global Variables in Windows,
http://moyix.blogspot.com/2008/04/finding-kernel-global-variables-in.html

15. Bypassing Klister 0.4 With No Hooks or Running a Controlled Thread Scheduler,
http://ht-group.net/33