УДК 004.056

СПОСОБ КОНТРОЛЯ ПОЛЬЗОВАТЕЛЬСКИХ ПОТОКОВ ДАННЫХ ДЛЯ USB-НАКОПИТЕЛЕЙ

Варавка А.В., Демеш Н.С., Цололо С.А.
ГВУЗ «Донецкий национальный технический университет»
antosha90@yandex.ru

В работе выполнен анализ существующих методов контроля и данных с помощью API-функций. Предложен способ контроля пользовательских потоков данных с внешних устройств в рамках реализации клиент-серверной системы, нацеленной на обеспечение информационной безопасности предприятия.

Введение

В рамках работы над системой обеспечения информационной безопасности в компьютерной сети предприятия [1, 2], прежде всего, необходимо реализовать надежный и эффективный способ перехвата пользовательских потоков данных для внешних устройств. В частном случае, предлагаемый в данной работе способ применяется авторами для контроля USB-накопителей, однако в общем случае он является универсальны и может с успехом применяться для любых потоков данных.

Современные операционные системы на основе ядра NT (2000, XP, 2003, Vista, 7) используют два режима привилегий при работе процессора – защищенный режим (режим ядра) и безопасный режим (пользовательский режим). В режиме ядра имеется доступ к вводу-выводу через порты памяти с помощью привилегированных команд. Драйверы, осуществляющие прямое управление устройствами, исполняются в защищенном режиме. Из пользовательского режима обращение к функциям ядра и драйверов возможно только через специальные шлюзы, предусмотренные разработчиками.

Ядро содержит функции Native API, которые могут быть вызваны из пользовательского режима. Программа пользовательского режима вызывает какую-либо функцию, и она с помощью шлюзов передает управление ядру. Ядро получает параметры функции и выполняет код, результаты исполнения передаются обратно. Например, стандартная WinAPI функция CreateFile вызывает недокументированную функцию ZwCreateFile из библиотеки ntdll.dll. Эта функция вызывает одноимённый сервис ядра через специальный шлюз. Процесс создания, копирования или изменения данных в большинстве случаев осуществляется выполнением стандартной функции CreateFile. Таким образом, при передаче управления функции CreateFile, вызывается функция ZwCreateFile. Подменив эту функцию, предоставляется возможность перехвата передачи пользовательских данных. В случае перехвата функции WinAPI имеется возможность анализа и контроля пользовательских потоков данных, в том числе и с внешних устройств, подключенных к шине USB.

Основные способы перехвата API-функций

Суть перехвата системной функции WinAPI состоит в подмене заданного адреса в памяти процесса заданного кода в функции [3]. В результате при вызове управление передается не системной функции, а функции, которая ее подменяет (так называемая «мнимая» функция). Работа мнимой функции заключается в выполнении заданных операций и возможном последующем вызове оригинальной функции. Такой перехват функций является эффективным средством отслеживания, изменения или блокировки действий программы.

Большинство функций, содержащихся в библиотеке ntdll.dll, являются функциями ядра, а одноимённые функции, доступные в пользовательском режиме представляют собой заглушки, которые передают управление функциям ядра через шлюзы. Функции, которые находятся в библиотеке ntdll.dll называют NT Native API [4].

При вызове Native API функции из библиотеки ntdll.dll в регистр eax записывается номер системной функции ядра. Затем вызывается прерывание 2Eh и происходит переход текущего потока в режим ядра [4]. Ядро содержит обработчик прерываний, в том числе и для прерывания 2Eh. Обработчиком является функция KiSystemService (KiST). Эта функция копирует передаваемые системному сервису параметры в память ядра, и производит вызов NativeAPI функции ядра согласно содержимому ServiceDescriptorTable. Данная таблица находится в памяти ядра, и представляет собой структуру, которая содержит 4 таблицы системных сервисов (SST). Первая из этих таблиц описывает сервисы ядра (ntoskrnl.exe), вторая – графической подсистемы (win32k.sys), остальные две зарезервированы на будущее и в данный момент не используются. ServiceDescriptorTable (SDT) содержит таблицы с указателями на функции. Функция пользовательского режима при вызове записывает индекс элемента в этой таблице.

Таким образом, для перехвата Native API функции в режиме ядра необходимо просто заменить её адрес в SDT. При этом для использования определенной функции, необходимо знать индексы элементов в таблице SDT Номер одной и той же функции в SDT зависит от операционной системы. Необходимо также учитывать, что перехват системных функций значительно снижает быстродействие системы, так как при каждом системном вызове выполняется дополнительный код перехватчика.

На данный момент известно несколько методов перехвата API-функций:

1. Перехват функции с помощью правки адресов в KiST. Это наиболее распространенный метод, однако, подобный вид перехвата очень заметен и существуют утилиты, позволяющие восстановить KiST.
2. Перехват модификацией машинного кода ядра. Перехват данного типа сложнее обнаружить и нейтрализовать, поэтому можно предполагать, что данный метод постепенно придет на смену модификации KiST. Одной из особенностей метода является возможность перехвата функций, которые не вызываются через KiST или не экспортируются ядром.
3. Перехват вектора 2Eh и sysenter.
4. Установка драйвера-фильтра.
5. Установка функции мониторинга загрузки исполняемых файлов и DLL.
6. Модификации объектов ядра без перехвата функций.

Перехват функции с помощью правки адресов в KiST

Рассматриваемый метод наиболее распространен и практически является классической реализацией данной задачи. Установка перехватчика производится по следующему алгоритму:

1. Перехватчик получает адрес таблицы SDT. Адрес этой таблицы экспортируется ядром, поэтому получение адреса SDT не вызывает проблем.
2. Перехватчик анализирует первую SST – это KiSystemService.
3. Из поля Service Table в KiSystemService перехватчик узнает адрес таблицы адресов функций, индекс функции позволяет найти ему нужную ячейку.
4. Адрес из обнаруженной ячейки таблицы сохраняется в некоторой переменной драйвера, после чего в нее заносится адрес функции-перехватчика («мнимой» функции). Перед записью адреса перехватчика драйвер должен запретить прерывания и сбросить бит WP регистра CR0. Бит WP отвечает за защиту страниц от записи при обращении с уровня супервизора.
5. После записи в Service Table драйвер должен восстановить исходное состояние бита WP и разрешить прерывания.

В данной работе предлагается реализация драйвера, перехватывающего функции в режиме ядра, был осуществлен перехват функции ZwCreateFile для мониторинга ее вызовов и демонстрации блокировки доступа к файлам. Для установки перехватчика необходимо получить адрес SDT, и узнать версию операционной системы:

```
extern «C» {
// SDT pointer
extern PSERVICE _ DESCRIPTOR _ TABLE KeServiceDescriptorTable;
extern  PWORD NtBuildNumber;}
```

Вторым шагом является объявление прототипа перехватываемой функции и переменной для хранения ее адреса:

```
typdef NTSTATUS
(NTAPI *PZwCreateFile) (
    OUT PHANDLE  FileHandle,
    IN ACCESS _ MASK  DesiredAccess,
    IN POBJECT _ ATTRIBUTES  ObjectAttributes,
    OUT PIO _ STATUS _ BLOCK  IoStatusBlock,
    IN PLARGE _ INTEGER  AllocationSize  OPTIONAL,
    IN ULONG  FileAttributes,
    IN ULONG  ShareAccess,
    IN ULONG  CreateDisposition,
    IN ULONG  CreateOptions,
    IN PVOID  EaBuffer  OPTIONAL,
    IN ULONG  EaLength    );
PZwCreateFile OldZwCreateFile;
```

Следующим шагом в разработке драйвера является написание функции и переменной для хранения ее адреса:

```
VOID SetKiSTHook() {
DWORD OldCR0;
KIRQL OldIRQL = KeRaiseIrqlToDpcLevel();// Повышение приоритета
_ asm {  // Сброс WP бита
```

```
        mov eax,CR0
        mov OldCR0,eax
        and eax,0xFFFEFFFF
        mov cr0, eax }
     switch (*NtBuildNumber) {
      case  2195:  // Win 2k
         OldZwCreateFile =
(PZwCreateFile)*KeServiceDescriptorTable->ntoskrnl.ServiceTable[0x20];
      KeServiceDescriptorTable->ntoskrnl.ServiceTable[0x20]=
(NTPROC)*MyZwCreateFile;
…
// для других ОС – аналогично с другим смещением
…
}
_ asm {  // Восстановление WP бита
    mov eax,OldCR0
    mov cr0,eax
  }
  // Восстановление приоритета
  KeLowerIrql(OldIRQL);}
```

Установка перехвата является типовой операцией из нескольких шагов:

1. Повышение приоритета, текущий уровень запоминается в переменной OldIRQL.
2. Сброс бита WP в регистре CR0 процессора, что приведет к отключению защиты памяти ядра от записи.
3. Считывание из Service Table адреса перехватываемой функции и запись вместо него адреса своей «мнимой» функции.
4. Восстановление бита WP в регистре CR0 и понижение приоритета до исходного.

Работа функции перехвата сводится к выводу имени объекта через DbgPrint. Затем выполняется поиск заданной подстроки в имени объекта. В случае обнаружения этой подстроки перехватчик завершает работу и возвращает код STATUS_ACCESS_DENIED:

```
NTSTATUS MyZwCreateFile(
     OUT PHANDLE  FileHandle,
     IN ACCESS _ MASK  DesiredAccess,
     IN POBJECT _ ATTRIBUTES  ObjectAttributes,
     OUT PIO _ STATUS _ BLOCK  IoStatusBlock,
     IN PLARGE _ INTEGER  AllocationSize  OPTIONAL,
     IN ULONG  FileAttributes,
     IN ULONG  ShareAccess,
     IN ULONG  CreateDisposition,
     IN ULONG  CreateOptions,
     IN PVOID  EaBuffer  OPTIONAL,
     IN ULONG  EaLength    )
{
    // Наша обработка ZwCreateFile
    DbgPrint(«%ws \n», ObjectAttributes->ObjectName->Buffer);
    // Блокировка доступа к файлу
        if (wcsstr(ObjectAttributes->ObjectName->Buffer, L»\\??\\E:\\») !=
```

```
NULL) {
            if (wcsstr(ObjectAttributes->ObjectName->Buffer, L» E:\\.rtf») «)
!= NULL)
         {  return STATUS _ ACCESS _ DENIED;}
     }
        // Вызов исходной функции
        return OldZwCreateFile(FileHandle, DesiredAccess, ObjectAttributes,
      IoStatusBlock, AllocationSize, FileAttributes, ShareAccess,
      CreateDisposition, CreateOptions, EaBuffer, EaLength);}
```

Результаты работы программы перехвата приведены на рис. 1.

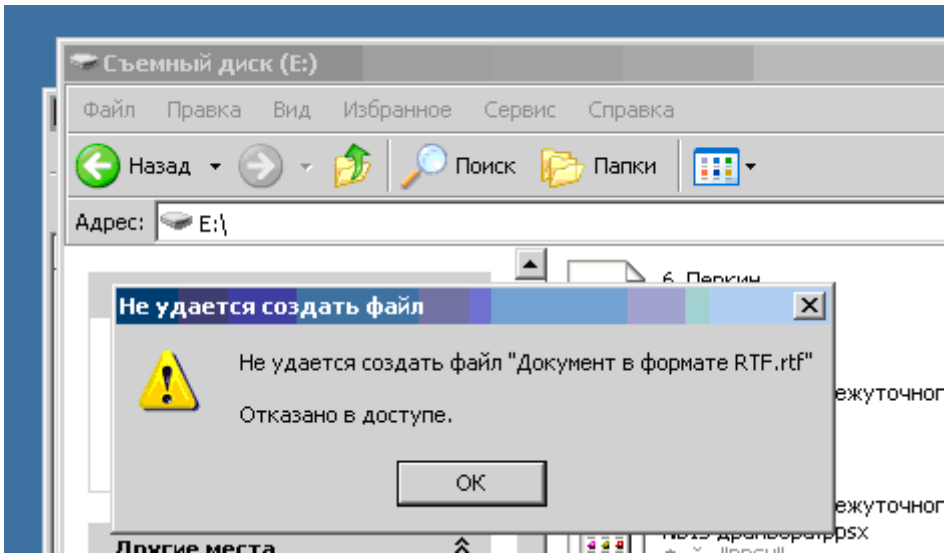

Рисунок 1. Результаты работы программы-перехватчика

Таким образом, на основе приведенного типа перехвата API-функций предлагается способ организации контроля пользовательских потоков данных, при помощи которого имеется возможность контролировать запись и редактирование пользователем определенных данных в соответствии с заданными критериями.

6

Выводы

В работе рассмотрены существующие методы перехвата API-функций, а также предложен способ организации контроля потоков данных. Реализация предложенного способа может быть использована в рамках комплексной системы обеспечения информационной безопасности на предприятиях.

В рамках предлагаемого способа реализована тестовая программа перехвата, которая позволяет отслеживать и контролировать запись, изменение и редактирование пользовательских потоков данных по заданным критериям.

Дальнейшая работа авторов будет направлена на реализацию расширенного алгоритма фильтрации потоков данных. Данный алгоритм позволит более точно вычленять информацию по ряду заданных критериев, что позволит улучшить точность распознавания необходимых данных и снизить количество используемых ресурсов системы.

Перечень источников

[1] Варавка А.В., Цололо С.А., Демеш Н.С. Исследование алгоритма обеспечения информационной безопасности в компьютерных системах предприятий на базе интерфейса USB // Информатика и компьютерные технологии / Сборник трудов VII международной конференции студентов, аспирантов и молодых ученых – 22-23 ноября 2011 года, г. Донецк, ДонНТУ – 2011, в 2-х томах. – С. 264-267

[2] Варавка А.В., Цололо С.А. Идентификация USB-устройств в системе анализа пользовательских потоков данных. / Інформаційні управляючі системи та комп'ютерний моніторинг (ІУСКМ-2012): ІІІ Всеукраїнська науково-технічна конференція студентів, аспірантів та молодих вчених, 16-18 квітня 2012 р., м. Донецьк / Донец. національ. техн. ун-т. – Донецьк: ДонНТУ, 2012. – С. 371-475.

[3] Перехват API-функций в Windows NT/2000/XP[элетронный ресурс]. – Режим доступа: http://www.rsdn.ru/article/baseserv/IntercetionAPI.xml

[4] Номера системных функций в SDT Windows [элетронный ресурс]. – Режим доступа: http://hex.pp.ua/service-descriptor-table.php