УДК 681.3.06:006.354

Я.В. ЧУПРИЙ, Р.В. ОЛЕЙНИКОВ

ПЕРЕХВАТ СИСТЕМНЫХ ВЫЗОВОВ В ОПЕРАЦИОННОЙ СИСТЕМЕ WINDOWS 2000

Введение

Последние достижения в области информационных технологий непосредственно связаны с появлением персональных компьютеров и сетей передачи данных. Так, с появлением глобальной сети Internet значительно упростилась циркуляция информации, и решился целый круг вопросов информационного обмена.

Развитие Internet-технологий увеличило оперативность передачи информации и обусловило их широкое распространение в повседневной жизни и в важнейших процессах всего мира. Это в конечном итоге способствовало открытию новых направлений науки и техники, сокращению сроков научных разработок и изысканий.

Первоначально поставки первых ЭВМ осуществлялись без единой управляющей программы, и пользователи были вынуждены сами создавать программные средства, охватывающие все аспекты решаемой задачи. Так как программирование того времени осуществлялось на уровне машинных кодов, это приводило к тому, что квалификация пользователя должна была быть очень высокой, что весьма сужало их потенциальный круг и делало ЭВМ доступной только для программиста.

Со временем стали появляться программы, облегчающие труд программиста, затем программы-утилиты общего применения, которые могли использовать пользователи при решении различных задач. Однако эти компьютерные программы были неудобными и громоздкими. Простые пользователи в расчет не брались. Даже после того, как в программном обеспечении компьютеров появилась командная строка, пользователям необходимо было помнить множество команд и опций, которые не были представлены на экране. Создание хорошего пользовательского интерфейса ограничивалось нехваткой оперативной памяти и низкой производительностью процессора. В дальнейшем такие программы систематизировались и проектировались так, чтобы выполнять основные функции взаимодействия пользователя с аппаратными средствами ЭВМ. Это привело к появлению операционных систем (ОС). А появление новых технологий в области микроэлектроники (увеличение емкости оперативной памяти и снижение ее себестоимости) создало предпосылку для улучшения пользовательского интерфейса.

Программа, которая является дружелюбной к пользователю, требует больших затрат от программиста. Основой концепцией Microsoft является разработка программного обеспечения с максимально удобным интерфейсом. Именно благодаря этой концепции в мире около 90% персональных компьютеров работают под управлением ОС семейства Windows. ОС семейства Windows – это унифицированный интерфейс, позволяющий пользователю с минимальными знаниями среды выполнять операции различной сложности. Следовательно, если программа пишется для компьютеров, совместимых с IBM PC, то в первую очередь она пишется для ОС семейства Windows.

Программирование в Windows основано на использовании интерфейса программирования приложений (Application Programming Interface, API). Помимо API, существует целый ряд средств, облегчающих работу программиста (например, использование библиотек классов Microsoft Foundation Classes (MFC)), но в конечном итоге любая из этих оболочек использует все тот же API [1]. И хотя Microsoft выпускает операционные системы Windows с разными типами ядер (Win9x/Me, WinNT/2000/XP, WinCE), API у них практически полностью совпадает. Это означает, что код, написанный для одного ядра [2], может быть применен с небольшими изменениями для другого. Но, несмотря на то, что API предоставляет широкий круг возможностей для программиста, в

некоторых случаях его необходимо дополнить или изменить. Наиболее часто для этих целей используется перехват вызовов API. Целью настоящей статьи является анализ защищенности и разработка метода защиты от перехватов API-вызовов в ОС Windows.

1. Анализ методов перехвата

Круг применения перехватов этого вида весьма значителен. Это и изучение поведения исследуемого приложения (определение порядка вызываемых функций с возможностью модификации кода каждой из них), и тестирование приложения на наличие программных закладок, и даже попытки несанкционированного вторжения с целью нарушения целостности и корректного функционирования программного продукта; именно из этих соображений рассмотрение данного вопроса имеет большое значение.

Предположим, в системе запущен некий процесс Authentic.exe, который реализует услугу управления доступом. Его задачей является авторизация пользователя, т.е. подтверждение, что пользователь соответствует тому, за кого себя выдает. Эта задача выполняется с использованием парольной защиты. Каждый вновь входящий пользователь идентифицирует себя секретным паролем, известным только ему. После ввода пароля процесс Authentic.exe выполняет процедуру хеширования и сравнивает полученный хеш-код с соответствующим значением учетной записи. Все учетные записи хранятся в файле Password.prv, для которого средствами операционной системы установлено только монопольное использование (защита от несанкционированного считывания информации). Обычно такие действия реализуются с помощью передачи следующих аргументов функции CreateFile() [4]:

```
HANDLE CreateFile(
L“Password.prv”,                    // имя файла
GENERIC_READ | GENERIC_WRITE,       // режим доступа по чтению и по записи
0,                                  // запрещение совместного использования
                                    // (параметр dwShareMode)
NULL,                               // дескриптор защиты по умолчанию:
                                    // разрешение доступа владельца и
                                    // администратора и
                                    // запрещение всем остальным
OPEN_EXISTING,                      // открыть существующий
NULL,   NULL);                      // атрибуты файла по умолчанию
```

Наиболее значимой угрозой в данном случае является утечка информации из файла Password.prv, так как у злоумышленника появляется возможность получить хеш-код привилегированного пользователя и осуществить подбор пароля. Одним из возможных вариантов реализации данной атаки является использование перехвата API-вызова.

Злоумышленник запускает процесс Manager.exe, который внедряет в адресное пространство процесса Authentic.exe динамическую библиотеку CarrierDll.dll. После проецирования на адресное пространство Authentic.exe библиотека CarrierDll.dll загружает библиотеку злоумышленника HookFunction.dll и производит замену вызова функции CreateFile() на функцию HookCreateFile() из библиотеки HookFunction.dll. На рис. 1 представлена функциональная схема взаимодействия процессов Manager.exe и Authentic.exe. После этого библиотеку CarrierDll.dll необходимо выгрузить, так как необходимость в ней исчерпана. Рассмотрим более детально функционирование данной схемы.

Необходимо отметить, что при разработке приложений, ориентированных на ОС Windows 2000, актуальным является использование Unicode-строк вместо привычных ANSI-строк, которые применяются в Windows 9x. Во-первых, появляется возможность обмена данными на разных языках, во-вторых, это связано с тем, что ОС Windows 2000 полностью построена на Unicode и, соответственно, функции ОС ожидают передачу Unicode-строк в

качестве аргументов (название этих функций заканчиваются литерой ‘W’, например LoadLibraryW()) [2]. Реализация функций для работы с ANSI-строками основана на выделении буфера преобразования входной ANSI-строки в Unicode-строку и последующем вызове функции, предназначенной для Unicode (функции, принимающие ANSI-строки заканчиваются литерой ‘A’, например LoadLibraryA()), что определяет снижение производительности и увеличение необходимого объёма доступной памяти для такого преобразования.

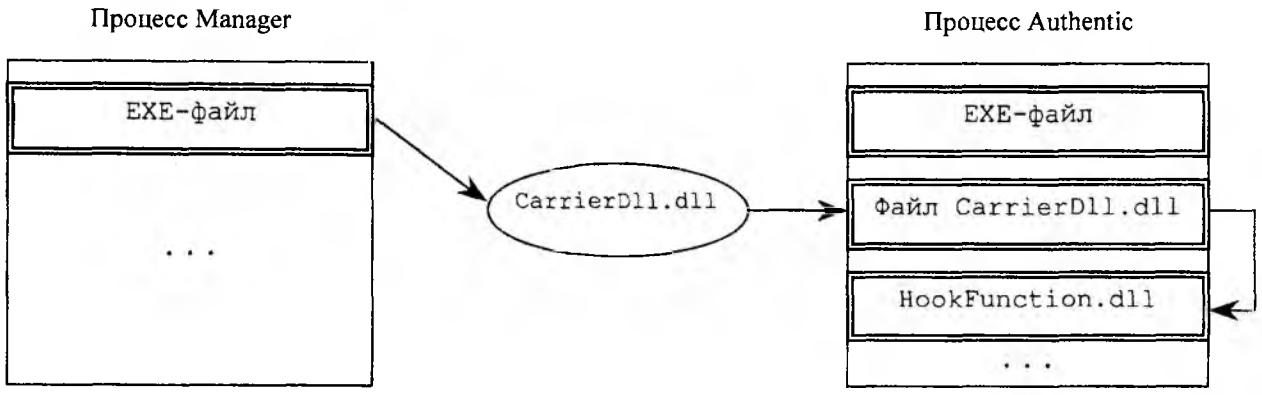

Рис. 1

Следовательно, применение Unicode оправдано с точки зрения производительности и экономии ресурсов; это можно сделать, объявив директивы:

```
#define UNICODE
#define _UNICODE
```

2. Внедрение CarrierDll.dll в адресное пространство процесса Authentic.exe

Средства операционной системы Windows2000 позволяют использовать эффективный метод внедрения DLL с помощью удаленных потоков.

Согласно концепции операционных систем семейства Windows NT, к которому относится и Windows 2000, каждый процесс имеет свое виртуальное адресное пространство. Доступ к адресному пространству процесса контролируется диспетчером виртуальной памяти и разрешается только потокам данного процесса, чтобы исключить возможность несанкционированной модификации или считывания адресного пространства одного процесса другим. Это возможно потому, что диспетчер виртуальной памяти во время трансляции виртуальных адресов в физические занимается вопросами разграничения доступа к адресным пространствам процессов. Однако в наборе API Win32 существуют функции ReadProcessMemory() и WriteProcessMemory(), которые позволяют считывать и модифицировать память процесса, они используются отладчиками для получения информации об отлаживаемом процессе и установки точек останова [3].

Принцип внедрения DLL в заданный процесс основан на вызове функции LoadLibrary() потоком этого процесса. Рассмотрим последовательность операций, необходимых для внедрения библиотеки CarrierDll.dll в адресное пространство процесса Authentic.exe:

а) Узнав идентификатор процесса Authentic.exe (ProcessId, с помощью Windows Task Manager), необходимо получить описатель этого процесса (ProcessHandle). Для этого целесообразно использовать функцию OpenProcess() [5] со следующим набором флагов доступа:

```
HANDLE ProcessHandle = OpenProcess(
PROCESS_CREATE_THREAD |      // разрешение на использование описателя процесса
```

```
                                    // для создания потока в этом процессе
                                    // CreateRemoteThread()
    PROCESS_VM_OPERATION |          // разрешение на использование описателя процесса
                                    // для операций с виртуальной памятью процесса
                                    // VirtualAllocEx()
    PROCESS_VM_WRITE,       // разрешение на использование описателя процесса
                            // для модификации виртуальной памяти процесса
    FALSE,                  // запретить наследование прав доступа
                            // порожденным процессам
    ProcessId);             // передаваемый идентификатор процесса
```

б) Выделить виртуальный блок памяти в адресном пространстве Authentic.exe для последующего размещения в этом блоке полного имени внедряемой библиотеки(CarrierDll.dll) [6]:

```
    PWSTR pszVirtualBaseAddress = (PWSTR)VirtualAllocEx(
    ProcessHandle,          // описатель процесса
    NULL,                   // место распределения памяти определяет система
    dwSize,                 // число байт, необходимых для строки, содержащей
                            // полное имя внедряемой библиотеки
    MEM_COMMIT,             // распределение памяти в оперативной памяти или
                            // в файле подкачки
    PAGE_READWRITE);        // обеспечить доступ для чтения и записи
                            // к выделенной области страниц
```

в) Записать строку в адресное пространство процесса Authentic.exe:

```
    WriteProcessMemory(
    ProcessHandle,          // описатель процесса в память которого будут
                            // записываться данные
    pszVirtualBaseAddress,  // указатель на адрес памяти, куда записываются данные
    (PVOID)pszDllName,      // полное имя внедряемой библиотеки CarrierDll.dll
    dwSize,                 // число байтов, необходимых для строки, содержащей
                            // полное имя внедряемой библиотеки
    NULL);                  // число байтов, записанных функцией, игнорируется
```

г) Определить точный адрес функции LoadLibraryW() в модуле Kernel32.dll:

```
    PTHREAD_START_ROUTINE lpStartAddress = (PTHREAD_START_ROUTINE)
    GetProcAddress(GetModuleHandle(
    TEXT("KERNEL32.DLL")),  // описатель модуля библиотеки Kernel32.dll
    "LoadLibraryW");        // имя определяемой функции
```

д) Создать удаленый поток в процессе Authentic.exe:

```
    HANDLE ThreadHandle = CreateRemoteThread(
    ProcessHandle,          // описатель процесса в который производиться внедрение
    NULL,                   // дескриптор защиты по умолчанию
    0,                      // размер стека потока по умолчанию, соответствует
                            // размеру стека основного потока процесса
    lpStartAddress,         // передаем адрес функции LoadLibrary()
    pszVirtualBaseAddress,  // в качестве аргумента функции LoadLibrary()
                            // передаем полное имя CarrierDll.dll
    0,                      // поток начинает немедленно выполняться после создания
    NULL);                  // идентификатор потока не возвращается
```

```
WaitForSingleObject(ThreadHandle, INFINITE);     // ожидаем завершения
                                                 // удаленного потока
```

е) Освободить выделенные ресурсы:

```
                                  // освобождение виртуальной памяти
VirtualFreeEx(hProcess, pszVirtualBaseAddress, 0, MEM_RELEASE);
CloseHandle(ThreadHandle);  // закрытие описателя удаленного потока
CloseHandle(ProcessHandle); // закрытие описателя процесса Authentic.exe
```

В результате выполнения этих операций мы получаем спроецированную библиотеку CarrierDll.dll на адресное пространство процесса Authentic.exe. Библиотека должна иметь функцию входа с использованием следующего механизма:

```
BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul_reason_for_call,
                                  LPVOID lpReserved)
{
  switch(ul_reason_for_call)
  {
  case DLL_PROCESS_ATTACH:
          MyInit();
          break;
  default:        break;
  }
  return TRUE;
}
```

Вызов функции DllMain() с уведомлением DLL_PROCESS_ATTACH происходит после проецирования CarrierDll.dll на адресное пространство Authentic.exe. Функция MyInit() выполняет все задачи, возложенные на библиотеку CarrierDll.dll, а именно: загрузки библиотеки HookFunction.dll и подмены API-функции CreateFile() в таблице адресов импорта исполняемого модуля Authentic.exe на функцию HookCreateFile(), (с разрешением на совместное использование файла по чтению FILE_SHARE_READ, параметр dwShareMode) содержащуюся в модуле HookFunction.dll. Рассмотрим более детально процедуру перехвата.

3. Перехват функции CreateFile()

Перехват функции CreateFile() может быть реализован следующим образом:

а) Получить описатель модуля в котором нужно произвести перехват функции. Так как библиотека CarrierDll.dll находится в адресном пространстве Authentic.exe, целесообразно использовать функцию GetModuleHandle(NULL):

```
HMODULE hTargetModule = GetModuleHandle(NULL);
```

б) Определяем адрес функции, которую необходимо подменить:

```
PCSTR pszModule = "Kernel32.dll";
lpOriginFunc = GetProcAddress(GetModuleHandle(pszModule), "CreateFileW");
```

в) Находим раздел импорта в исполнительном модуле Authentic.exe:

```
PIMAGE_IMPORT_DESCRIPTOR pImportDescriptor =
(PIMAGE_IMPORT_DESCRIPTOR) ImageDirectoryEntryToData(
hTargetModule,                          //описатель исполнительного модуля
TRUE,                                   // файл спроецирован системой
IMAGE_DIRECTORY_ENTRY_IMPORT,  // определяет номер индекса необходимого
```

```
                                        // входа соответствующего раздела импорта
                                        // исполняемого модуля
    &Size);                             // размер переданного параметра
```

г) Организуем цикл для поиска описателя раздела импорта нужной DLL (Kernel32.dll)

```
    while(pImportDescriptor -> Name)
    {
    PSTR pszCurrentModule = (PSTR)((PBYTE)hTargetModule + pImportDescriptor -> Name);
    if(!lstrcmpiA(pszCurrentModule, pszModule)) break;
    pImportDescriptor++;
    }
```

д) Выделяем таблицу адресов импорта нашей библиотеки

```
    PIMAGE_THUNK_DATA pImportAddressTable = (PIMAGE_THUNK_DATA)((PBYTE)
    hTargetModule + pImportDescriptor -> FirstThunk);
```

е) Реализуем цикл поиска функции, требуемой для замены и в случае нахождения производим замену адреса в разделе импорта исполняемого модуля:

```
    while(pImportAddressTable -> u1.Function)
    {
      // определяем указатель на адрес текущей функции
      PROC *ppCurrentFunc = (PROC*)&pImportAddressTable -> u1.Function;

      // является ли текущая функция искомой?
      if(*ppCurrentFunc == lpOriginFunc)
      {
            WriteProcessMemory(
            GetCurrentProcess(),
            ppCurrentFunc,
            &lpReplFunc,              // адрес функции в библиотеке HookFunction.dll
            sizeof(lpReplFunc),
            NULL);
            return;                   // операция завершена успешно
      }
      pImportAddressTable++;          // перемещаемся внутри таблицы импорта
    }
```

Также необходимо повторить данную процедуру перехвата для функции CreateFileA(). Функция HookCreateFile() имеет вид:

```
    HANDLE WINAPI HookCreateFile(
    LPCTSTR        lpFileName
    DWORD          dwDesiredAccess,
    DWORD          dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD          dwCreationDisposition,
    DWORD          dwFlagsAndAttributes,
    HANDLE         hTemplateFile)
    {
      return CreateFile(
                   lpFileName,
```

```
dwDesiredAccess,
FILE_SHARE_READ,        // разрешение на совместное
                        // использование файла по чтению
lpSecurityAttributes,
dwCreationDisposition,
dwFlagsAndAttributes,
hTemplateFile)
```

Следует отметить, что функция HookCreateFile() производит только лишь модификацию значения dwShareMode с последующим вызовом CreateFile() из модуля Kernel32.dll. Так как мы модифицировали адрес вызова функции CreateFile() в таблице адресов импорта исполняемого модуля, то из модуля HookFunction.dll этот вызов осуществляется без изменения, из библиотеки Kernel32.dll. После того как модуль CarrierDll.dll загрузил HookFunction.dll и перенаправил вызов CreateFile() на HookCreateFile(), он успешно выполнил поставленные задачи и требует незамедлительной выгрузки для освобождения занимаемых ресурсов. Процедура выгрузки заключается в создании удаленного потока в целевом процессе Authentic.exe и передачи в качестве стартовой функции потока адреса функции FreeLibrary() из модуля Kernel32.dll.

4. Выгрузка библиотеки CarrierDll.dll

а) Получить снимок состояния процесса:

```
HANDLE SnapshotModuleListHandle = CreateToolhelp32Snapshot(
TH32CS_SNAPMODULE,          // снимок включает в себя список модулей
                            // процесса
ProcessId);                 // идентификатор процесса Authentic.exe
```

б) Определить описатель CarrierDll.dll

```
MODULEENTRY32W EntryListModuleProcess = { sizeof(EntryListModuleProcess) };
BOOL fCurrentModule = Module32FirstW(SnapshotModuleListHandle,
                                     &EntryListModuleProcess);
BOOL CheckFound = NULL;
While(fCurrentModule)
{
CheckFound = (lstrcmpiW(EntryListModuleProcess.szModule, pszDllName) == 0) ||
(lstrcmpiW(EntryListModuleProcess.szExePath, pszDllName) == 0);
if (CheckFound) break;
fCurrentModule = Module32NextW(SnapshotModuleListHandle, &EntryListModuleProcess)
}
```

в) Определить описатель Authentic.exe:

```
HANDLE ProcessHandle = OpenProcess(
PROCESS_CREATE_THREAD | PROCESS_VM_OPERATION, FALSE, ProcessId);
```

г) Получить действительный адрес FreeLibraryW() в Kernel32.dll

```
PTHREAD_START_ROUTINE lpStartAddress =
(PTHREAD_START_ROUTINE)GetProcAddress(GetModuleHandle(TEXT("Kernel32")),
                                      "FreeLibrary");
```

д) Создать удаленный поток в адресном пространстве Authentic.exe и передать в качестве стартовой функции истинный адрес функции FreeLibraryW() в Kernel32.dll:

```
HANDLE ThreadHandle = CreateRemoteThread(ProcessHandle, NULL, 0, lpStartAddress,
EntryListModuleProcess.modBaseAddr, 0, NULL);
```

е) Ожидаем завершения удаленного потока
WaitForSingleObject(ThreadHandle, INFINITE);

ж) Освободить выделенные ресурсы:
CloseHandle(SnapshotModuleListHandle);
CloseHandle(ThreadHandle);
CloseHandle(ProcessHandle);

В результате библиотека CarrierDll.dll, выполнив свою задачу, была выгружена из адресного пространства процесса Authentic.exe, перенаправив вызов функции CreateFile() в исполняемом модуле на функцию HookCreateFile() из модуля HookFunction.dll, которая разрешает совместное использование файла Password.prv по чтению.

5. Метод защиты от перехватов API-вызовов

Технология перехвата API-вызовов может быть использована не только для отладочных, диагностических целей, но и при написании программных закладок, нарушении корректного функционирования приложения, поэтому необходимо предусмотреть механизмы защиты от использования перехватов несанкционированным пользователем.

Одним из методов защиты от перехватов API-вызовов является включение в охраняемый программный продукт механизма предварительного перехвата API-вызовов, которые могут быть причастными к несанкционированному внедрению дополнительных библиотек и подмене API-функций в адресном пространстве защищаемого процесса. К таким функциям относятся: LoadLibraryA, LoadLibraryW, LoadLibraryExA, LoadLibraryExW, GetProcAddress и WriteProcessMemory.

Следует отметить, что при грамотном администрировании и применении приложений, созданных с учётом вопросов безопасности, Windows2000 является одной из наиболее защищенных ОС. Однако, несмотря на это, она содержит множество недокументированных функций и возможностей. Хотя ни одна из них не является специально направленной на снижение уровня безопасности, тем не менее, не исключается возможность наличия закладок, что делает нежелательным ее применение в организациях с высоким уровнем защищенности. Это подчеркивает необходимость в создании для Украины своей собственной операционной системы.

Список литературы: 1. *Петзолд Ч.* Программирование для Windows 95. Т. 1. СПб.: BHV, 1996. 495 с. 2. *Рихтер Дж.* Windows для профессионалов. СПб.: Питер, 2001. 722 с. 3. *Кастер Х.* Основы Windows NT и NTFS: Пер. с англ. М.: Изд. отдел "Русская редакция" ТОО "Channel Trading Ltd", 1996. 440 с. 4. *Рихтер Дж., Кларк Дж.* Программирование серверных приложений для Windows 2000. СПб.: Питер, 2001. 566 с. 5. Microsoft Platform SDK (Windows 2000) 6. *Саймон Р.* Microsoft Windows 2000 API. Энциклопедия программиста. 2001 1086 с.

Харьковский национальный университет радиоэлектроники

Поступила в редколлегию 29.04.2002