

DNAlerter

1.0

Создано системой Doxygen 1.13.2

Глава 1

DNAlert

Оповещение о небезопасном подключении к сети

Глава 2

Список файлов

2.1 Файлы

Полный список документированных файлов.

code.cpp	
Код для определения профиля сети	??

Глава 3

Файлы

3.1 Файл code.cpp

Код для определения профиля сети

```
#include <windows.h>
#include <iostream>
#include <thread>
#include <fstream>
#include <atomic>
#include <cwchar>
```

Макросы

```
#define IDI_MYICON 101
#define ID_TRAY_APP_ICON 1
#define WM_TRAYNOTIFY (WM_USER + 1)
#define IDM_EXIT 1000
```

Функции

```
std::atomic< bool > running (true)
int startup ()
LRESULT CALLBACK WndProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
void InitTray (HWND hwnd)
void RemoveTray (HWND hwnd)
std::chrono::system_clock::time_point makeTimePoint (int year, int month, int day, int hour, int minute, int second)
int check_win_registry ()
    Получение профиля сети
int main ()
    Основная функция
```

Переменные

HWND hwnd

3.1.1 Подробное описание

Код для определения профиля сети

Открывает реестр Windows и проверяет тип сети, к которой выполнялось последнее подключение.

3.1.2 Функции

3.1.2.1 check_win_registry()

int check_win_registry ()

Получение профиля сети

Каждые десять секунд сканирует реестр виндовс

Получает последний профиль сети

Записывает в файл is_threat тип сети

```

00154 {
00155     HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
00156     WCHAR lastProfile[255] = L"None";
00157     auto lastDate = makeTimePoint(0,0,0,0,0);
00158     int lastCategory = 0;
00159     while (running){
00160         DWORD index = 0;
00161         WCHAR subKeyName[255];
00162         DWORD cbName = 255;
00163
00164
00165         int year;
00166         int month;
00167         int day;
00168         int hour;
00169         int minute;
00170         int second;
00171
00172
00173         HKEY hKey;
00174
00175         LONG result = RegOpenKeyExW(HKEY_LOCAL_MACHINE,
00176                                     L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\NetworkList\\Profiles",
00177                                     0,
00178                                     KEY_READ,
00179                                     &hKey);
00180         if (result != ERROR_SUCCESS) {
00181             std::wcerr « L"Failed to open registry key. Error code: " « result « std::endl;
00182             return 1;
00183         }
00184         //std::cout«RegEnumKeyExW(hKey, index, subKeyName, &cbName, NULL, NULL, NULL, NULL)«std::endl;
00185         while (RegEnumKeyExW(hKey, index, subKeyName, &cbName, NULL, NULL, NULL, NULL) ==
ERROR_SUCCESS) {
00186             HKEY hSubKey;
00187             result = RegOpenKeyExW(hKey, subKeyName, 0, KEY_READ, &hSubKey);
00188             if (result == ERROR_SUCCESS) {
00189                 WCHAR profileName[255];
00190                 DWORD cbData = sizeof(profileName);
00191                 result = RegQueryValueExW(hSubKey, L"ProfileName", NULL, NULL, (LPBYTE)profileName, &cbData);
00192                 if (result == ERROR_SUCCESS) {
00193                     DWORD type;
00194                     cbData = 0;
00195                     result = RegQueryValueExW(hSubKey, L"DateLastConnected", NULL, &type, NULL, &cbData);
00196                     if (result == ERROR_SUCCESS && type == REG_BINARY) {
00197
00198                         BYTE* dateData = new BYTE[cbData];
00199                         result = RegQueryValueExW(hSubKey, L"DateLastConnected", NULL, NULL, dateData, &cbData);
00200                         if (result == ERROR_SUCCESS) {

```

```

00201
00202     year = int(dateData[0]+dateData[1]*256);
00203     month = int(dateData[2]+dateData[3]*256);
00204     day = int(dateData[6]+dateData[7]*256);
00205     hour = int(dateData[8]+dateData[9]*256);
00206     minute = int(dateData[10]+dateData[11]*256);
00207     second = int(dateData[12]+dateData[13]*256);
00208
00209     auto date = makeTimePoint(year,month,day,hour,minute,second);
00210
00211     DWORD category;
00212     RegQueryValueExW(hSubKey, L"Category", NULL, &type, (LPBYTE)&category, &cbData);
00213
00214     if (date>lastDate or (wcscmp(profileName,lastProfile)==0 and category!=lastCategory)){
00215         lastDate=date;
00216         lastCategory=category;
00217         WriteConsoleW(hConsole, L"Changed Profile Name: ", 22, nullptr, nullptr);
00218         WriteConsoleW(hConsole, profileName, wcslen(profileName), nullptr, nullptr);
00219         WriteConsoleW(hConsole, "\n", 1, nullptr, nullptr);
00220
00221         if (category==0){
00222             MessageBoxW(NULL, L"Вы подключены к небезопасной сети!", L"Внимание", MB_OK |
MB_ICONINFORMATION);
00223         }
00224         wcsncpy(lastProfile, profileName, 255);
00225     } else {
00226         std::wcerr << L"Failed to read DateLastConnected. Error code: " << result << std::endl;
00227     }
00228 } else if (result == ERROR_FILE_NOT_FOUND) {
00229     std::wcerr << L"DateLastConnected not found for profile: " << profileName << std::endl;
00230 } else {
00231     std::wcerr << L"Failed to query DateLastConnected. Error code: " << result << std::endl;
00232 }
00233 }
00234 } else{
00235     std::wcerr << L"Error" << std::endl;
00236 }
00237 RegCloseKey(hSubKey);
00238 }
00239 index++;
00240 cbName = 255;
00241 }
00242 WriteConsoleW(hConsole, L"LastProfile Name: ", wcslen(L"LastProfile Name: "), nullptr, nullptr);
00243 WriteConsoleW(hConsole, lastProfile, wcslen(lastProfile), nullptr, nullptr);
00244 WriteConsoleW(hConsole, "\n", 1, nullptr, nullptr);
00245
00246 std::wcout << L"Last Category: " << lastCategory << std::endl;
00247 RegCloseKey(hKey);
00248
00249 std::ofstream myFile("./is_threat", std::ios::out | std::ios::trunc);
00250 if (myFile.is_open()) {
00251     myFile << lastCategory;
00252     myFile.close();
00253 }
00254 std::this_thread::sleep_for(std::chrono::seconds(10));
00255 }
00256 return 0;
00257 }
00258 }

```

3.1.2.2 InitTray()

```

void InitTray (
    HWND hwnd)

```

Создание иконки на панели задач при запуске программы

```

00093 {
00094     NOTIFYICONDATAW nid = {0};
00095     nid.cbSize = sizeof(NOTIFYICONDATA);
00096     nid.hWnd = hwnd;
00097     nid.uID = ID_TRAY_APP_ICON;
00098     nid.uFlags = NIF_ICON | NIF_MESSAGE | NIF_TIP;
00099     nid.uCallbackMessage = WM_TRAYNOTIFY;
00100     nid.hIcon = LoadIcon(GetModuleHandle(NULL), MAKEINTRESOURCE(IDI_MYICON));
00101     wcsncpy(nid.szTip, L"Оповещение о небезопасном подключении к сети");
00102     Shell_NotifyIconW(NIM_ADD, &nid);
00103 }

```

3.1.2.3 main()

```
int main ()
```

Основная функция

Запускает функцию для создания иконки на панели задач.

Запускает поток проверки подключения.

```
00269 {
00270     startup();
00271
00272     WNDCLASSEXW wc = {0};
00273     wc.cbSize = sizeof(WNDCLASSEXW);
00274     wc.lpfnWndProc = WndProc;
00275     wc.hInstance = GetModuleHandleW(NULL);
00276     wc.lpszClassName = L"TrayAppClass"; // Wide-character string
00277     RegisterClassExW(&wc);
00278
00279     hwnd = CreateWindowW(L"TrayAppClass", L"TrayApp", WS_OVERLAPPEDWINDOW,
00280         0, 0, 0, 0, NULL, NULL, wc.hInstance, NULL);
00281
00282     if (!hwnd) {
00283         std::cerr << "Error creating icon!" << std::endl;
00284         return 1;
00285     }
00286
00287
00288
00289     InitTray(hwnd);
00290
00291     std::thread cwrThread(check_win_registry);
00292
00293     MessageBoxW(NULL,
00294         L"Приложение запущено!",
00295         L"Успешно",
00296         MB_OK | MB_ICONINFORMATION);
00297
00298
00299     MSG msg;
00300     while (GetMessageW(&msg, NULL, 0, 0)) {
00301         TranslateMessage(&msg);
00302         DispatchMessage(&msg);
00303     }
00304
00305     // Очистка
00306     RemoveTray(hwnd);
00307     cwrThread.join();
00308
00309     std::ofstream myFile("./is_threat", std::ios::out | std::ios::trunc);
00310     if (myFile.is_open()) {
00311         myFile << "1";
00312         myFile.close();
00313     }
00314
00315     return 0;
00316 }
00317 }
```

3.1.2.4 makeTimePoint()

```
std::chrono::system_clock::time_point makeTimePoint (
    int year,
    int month,
    int day,
    int hour,
    int minute,
    int second)
```

Получает на вход год, месяц, день, час, минуту и секунду в формате числа

И конвертирует их в формат времени

```
00133 {
00134     std::tm tm = {0};
00135     tm.tm_year = year - 1900; // Years since 1900
```

```

00136     tm.tm_mon = month - 1;    // Months since January (0-11)
00137     tm.tm_mday = day;        // Day of the month (1-31)
00138     tm.tm_hour = hour;       // Hours since midnight (0-23)
00139     tm.tm_min = minute;      // Minutes after the hour (0-59)
00140     tm.tm_sec = second;      // Seconds after the minute (0-60)
00141
00142     std::time_t tt = std::mktime(&tm);
00143     return std::chrono::system_clock::from_time_t(tt);
00144 }

```

3.1.2.5 RemoveTray()

```

void RemoveTray (
    HWND hwnd)

```

Удаление иконки на панели задач при выключении программы

```

00111 {
00112     NOTIFYICONDATA nid = {0};
00113     nid.cbSize = sizeof(NOTIFYICONDATA);
00114     nid.hWnd = hwnd;
00115     nid.uID = ID_TRAY_APP_ICON;
00116     Shell_NotifyIcon(NIM_DELETE, &nid);
00117 }

```

3.1.2.6 startup()

```

int startup ()

```

Добавляет путь к программе в список автозагрузки windows

```

00030 {
00031     wchar_t path[MAX_PATH];
00032     GetModuleFileNameW(NULL, path, MAX_PATH);
00033     std::wstring progPath = path;
00034     HKEY hkey = NULL;
00035     LONG createStatus = RegCreateKeyW(HKEY_CURRENT_USER,
00036                                     L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
00037                                     &hkey);
00038
00039     if (createStatus == ERROR_SUCCESS) {
00040         LONG status = RegSetValueExW(hkey,
00041                                     L"DAlert",
00042                                     0,
00043                                     REG_SZ,
00044                                     (BYTE*)progPath.c_str(),
00045                                     (progPath.size() + 1) * sizeof(wchar_t));
00046
00047         RegCloseKey(hkey);
00048     }
00049
00050     return 0;
00051 }

```

3.1.2.7 WndProc()

```

LRESULT CALLBACK WndProc (
    HWND hwnd,
    UINT msg,
    WPARAM wParam,
    LPARAM lParam)

```

Функция для выключения программы, если нажата кнопка "Выключить"

```

00057 {
00058     switch (msg) {
00059         case WM_TRAYNOTIFY:
00060             if (lParam == WM_RBUTTONDOWN) {
00061                 POINT pt;
00062                 GetCursorPos(&pt);

```



```
00063         HMENU hMenu = CreatePopupMenu();
00064         AppendMenuW(hMenu, MF_STRING, IDM_EXIT, L"Выключить");
00065         SetForegroundWindow(hwnd);
00066         TrackPopupMenu(hMenu, TPM_RIGHTALIGN | TPM_BOTTOMALIGN | TPM_RIGHTBUTTON,
00067             pt.x, pt.y, 0, hwnd, NULL);
00068         DestroyMenu(hMenu);
00069     }
00070     break;
00071 case WM_COMMAND:
00072     if (LOWORD(wParam) == IDM_EXIT) {
00073         running = false;
00074         PostQuitMessage(0);
00075     }
00076     break;
00077 case WM_DESTROY:
00078     running = false;
00079     PostQuitMessage(0);
00080     break;
00081 default:
00082     return DefWindowProcW(hwnd, msg, wParam, lParam);
00083 }
00084 return 0;
00085 }
```