Lab_4_Final.cpp

Importarea Bibliotecilor

```
#pragma comment(lib, "Wintrust.lib")
#include <windows.h>
#include <psapi.h>
#include <wintrust.h>
#include <softpub.h>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
```

Acestea sunt bibliotecile necesare pentru a accesa funcționalitățile API-ului Windows utilizate în program. `Wintrust.lib` este necesar pentru verificarea semnăturilor digital

Definirea Structurii `ProcessInfo`

```
struct ProcessInfo {
    DWORD processID;
    std::string executablePath;
    std::string signatureStatus;
};
```

Această structură stochează informațiile despre un proces: ID-ul procesului, calea executabilului și statusul semnăturii digitale.

Conversia de la `wchar_t` la `std::string`

```
std::string WideCharToString(const wchar_t* wideCharStr)
{
    int bufferSize = WideCharToMultiByte(CP_UTF8, 0,
wideCharStr, -1, nullptr, 0, nullptr, nullptr);
    std::string str(bufferSize, 0);
    WideCharToMultiByte(CP_UTF8, 0, wideCharStr, -1,
&str[0], bufferSize, nullptr, nullptr);
    return str;
}
```

Această funcție convertește un `wchar_t` (folosit pentru șiruri Unicode în Windows) într-un `std::string` (folosit pentru șiruri de caractere în C++).

Verificarea Semnăturii Digitale

```
std::string CheckSignature(const std::string& filePath) {
    WINTRUST_FILE_INFO fileData;
    memset(&fileData, 0, sizeof(fileData));
    fileData.cbStruct = sizeof(WINTRUST_FILE_INFO);
    std::wstring wideFilePath =
std::wstring(filePath.begin(), filePath.end());
    fileData.pcwszFilePath = wideFilePath.c_str();

    WINTRUST_DATA winTrustData;
    memset(&winTrustData, 0, sizeof(winTrustData));
    winTrustData.cbStruct = sizeof(winTrustData);
    winTrustData.dwUIChoice = WTD_UI_NONE;
    winTrustData.fdwRevocationChecks = WTD_REVOKE_NONE;
    winTrustData.dwUnionChoice = WTD_CHOICE_FILE;
    winTrustData.dwStateAction = WTD_STATEACTION_VERIFY;
    winTrustData.dwProvFlags = WTD_SAFER_FLAG;
```

```
    winTrustData.pFile = &fileData;

    GUID policyGUID = WINTRUST_ACTION_GENERIC_VERIFY_V2;

    LONG status = WinVerifyTrust(NULL, &policyGUID,
&winTrustData);
    if (status == ERROR_SUCCESS) {
        return "Trusted";
    }
    else {
        return "Untrusted";
    }
}
```

Această funcție folosește API-ul Windows pentru a verifica semnătura digitală a unui fișier. Returnează "Trusted" dacă semnătura este validă și "Untrusted" în caz contrar.

Obținerea Informațiilor despre Procese

```
std::vector<ProcessInfo> GetProcessInfo() {
    std::vector<ProcessInfo> processes;
    DWORD aProcesses[1024], cbNeeded, cProcesses;
    if (!EnumProcesses(aProcesses, sizeof(aProcesses),
&cbNeeded)) {
        return processes;
    }
    cProcesses = cbNeeded / sizeof(DWORD);
    for (unsigned int i = 0; i < cProcesses; i++) {
        if (aProcesses[i] != 0) {
            TCHAR szProcessName[MAX_PATH] = TEXT("
<unknown>");
            HANDLE hProcess =
```

```
OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ,
FALSE, aProcesses[i]);
            if (hProcess ≠ NULL) {
                HMODULE hMod;
                DWORD cbNeeded;
                if (EnumProcessModules(hProcess, &hMod,
sizeof(hMod), &cbNeeded)) {
                    GetModuleFileNameEx(hProcess, hMod,
szProcessName, sizeof(szProcessName) / sizeof(TCHAR));
                }
                ProcessInfo pInfo;
                pInfo.processID = aProcesses[i];

#ifdef UNICODE
                pInfo.executablePath =
WideCharToString(szProcessName);
#else
                pInfo.executablePath = szProcessName;
#endif

                pInfo.signatureStatus =
CheckSignature(pInfo.executablePath);
                processes.push_back(pInfo);
                CloseHandle(hProcess);
            }
        }
    }
    return processes;
}
```

Această funcție enumeră toate procesele care rulează pe sistem, obține calea executabilului pentru fiecare proces și verifică

semnătura digitală a fișierului. Informațiile despre fiecare proces sunt stocate într-un vector de `ProcessInfo`.

Salvarea Informațiilor într-un Fișier JSON

```
void SaveToJson(const std::vector<ProcessInfo>&
processes, const std::string& filename) {
    std::ofstream file(filename);
    if (!file.is_open()) {
        std::cerr << "Could not open the file for
writing!\n";
        return;
    }
    file << "[\n";
    for (size_t i = 0; i < processes.size(); ++i) {
        file << "  {\n";
        file << "    \"ProcessID\": " <<
processes[i].processID << ",\n";
        file << "    \"ExecutablePath\": \"" <<
processes[i].executablePath << "\",\n";
        file << "    \"SignatureStatus\": \"" <<
processes[i].signatureStatus << "\"\n";
        if (i != processes.size() - 1) {
            file << "  },\n";
        }
        else {
            file << "  }\n";
        }
    }
    file << "]\n";
    file.close();
}
```

Această funcție salvează informațiile despre procese într-un fișier JSON. Parcurge vectorul de `ProcessInfo` și scrie fiecare element în format JSON într-un fișier specificat.

Funcția `main`

```
int main() {
    std::vector<ProcessInfo> processes =
GetProcessInfo();
    SaveToJson(processes,
"C:/Users/Adminstrator/Desktop/UTM_2023-
2024/Sem_II/PMRI/Lab_4/processes.json");
    std::cout << "Process information saved to
processes.json\n";
    return 0;
}
```

Aceasta este funcția principală a programului. Ea obține informațiile despre procesele curente și le salvează într-un fișier JSON. Mesajul „Process information saved to processes.json” este afișat la final pentru a indica faptul că operația a fost realizată cu succes.

Concluzie

Programul obține și salvează informații despre procesele active în format JSON, verificând și semnătura digitală a fiecărui executabil. Acest lucru poate fi util pentru administrarea sistemului sau pentru securitate, oferind o modalitate de a monitoriza procesele și a verifica integritatea acestora.

Bibliografie

Link Git