Lab 4. Automated digital forensic analysis tool

Write a tool in C++ **using WinAPI, not third party libraries**, which will implement one of these features:

> The task is chosen based on the group list. Just ask me, if you are not sure about your number.

1. Extract system processes details and save in a JSON file:

- Process ID: e.g., 1201;
- Executable File Path: e.g., `C:\Program Files (x86)\Mozilla Firefox\firefox.exe`;
- Digital Signature Validity: trusted & integral, untrusted/corrupted, or not present.

2. The network information for each process will be saved in a JSON file, including the following:

- Process ID;
- Local address and port;
- Destination address and port (if present);
- Transport layer protocol used;
- State (e.g., ESTABLISHED, LISTENING).

If a process has no open network sockets, it will not be included in the file.

3. Extract WinAPI Imports from a PE Executable and Identify Malicious Ones

Identify suspicious functions such as: IsDebuggerPresent, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread, SetWindowsHookEx, etc.

Useful resources:

- A dive into the PE file format;
- 101 Editor;

For the JSON part, you may use third-party libraries. A popular C++ implementation is nlohmann/json.

Conditions:

- Use WinAPI directly, without libraries (for the core task);
- The code will be accompanied by links to the respective documentation;
- The results of the tasks must include the malicious program developed in Practical Work No. 2.

What you should learn after completing these tasks:

- understand the difference between types like NT_IMAGE_HEADERS and PNT_IMAGE_HEADERS (the latter is a pointer to the structure);
- what a handle is and how it is used in the WinAPI;
- the difference between A and W function suffixes (ANSI vs. Wide/Unicode);
- the distinction between Ex and non-Ex function variants (e.g., CreateFile vs. CreateFileEx);

- how to create and configure a project in Visual Studio;
- how to use what Windows provides for developers (headers, libraries, SDKs);
- how to read and navigate Microsoft's documentation (MSDN / learn.microsoft.com);
- the importance of checking return values and handling errors properly;
- basic usage of debugging tools (e.g., Visual Studio debugger).

These principles, such as understanding low-level system interaction, handling resources, and utilizing platform-specific development tools, are universal across all operating systems and major frameworks, including systems like Linux, macOS, Android, and large frameworks such as .NET or Java.