The Stealth Interceptor

Group Members:

1. **Muhammad Adeel Haider** (Student ID: 241541)
2. **Umar Farooq** (Student ID: 241575)

1. Introduction

As Cyber Security students, we are learning how malware functions and how security software detects it. A core technique used by both rootkits (malware) and antivirus software is "API Hooking."

This project proposes the development of our own **API Hooking Engine** using **MASM x86 Assembly**. The goal is to intercept standard Windows system calls (such as `MessageBox`) by modifying memory during program execution. This project will allow us to demonstrate a deep understanding of low-level system behavior and control flow.

2. Problem Statement

The challenge we want to solve is modifying a running process's instruction set without causing the application to crash. This is difficult because it requires precise manual management of:

- **CPU Registers** (to prevent data corruption).
- **The Stack Pointer** (to ensure the program can return to its previous state).
- **Memory Permissions** (modifying Read-Only code segments).

3. Methodology (The "Trampoline" Technique)

We will implement a "Detour Hook" using the following steps:

1. **Target Acquisition:** The program will use Indirect Addressing to locate the exact memory address of a system function (e.g., inside `user32.dll`).
2. **Memory Overwrite:** We will programmatically change the memory protections of the target function and overwrite the first 5 bytes with a `JMP` (Jump) instruction pointing to our custom code.
3. **The Trampoline Execution:**
    - **Phase 1:** Our custom Assembly code runs (e.g., logging "Interceptor Active!").
    - **Phase 2:** We execute the original "stolen" bytes that were overwritten.
    - **Phase 3:** We jump back to the original function to allow the program to finish normally, ensuring the user does not notice the interception.

4. Tools & Deliverables

- **Development Environment:** Visual Studio Community Edition.
- **Assembler:** MASM (Microsoft Macro Assembler).
- **Debugging:** x64dbg (for analyzing memory segments).
- **Final Output:** A working demonstration where a standard message box is intercepted by our Assembly engine before it appears on the screen.