Realtime Process Detection System

1. 環境 (Environment)

- 操作系統：Ubuntu 20.04.6 LTS
- 編譯器：gcc 和 g++，支持 C++17
- CMake：版本 3.10 或更高
- Clang/LLVM：版本 9 或更高，用於編譯 * eBPF 程式
- libbpf：eBPF 用戶態庫
- nlohmann_json：JSON 處理庫
- Google Test：單元測試框架
- 其他依賴：
  libelf-dev
  libssl-dev
  pkg-config
  libgtest-dev

2. 程式功能 (Program Functionality)

- 實時監控進程創建和執行：使用 eBPF 技術監控 execve 系統調用，捕獲新的進程啟動事件。
- process掃描：對新啟動的進程執行文件進行掃描，檢查是否包含特定的字符串或模式，以檢測潛在的惡意程式。
- 結果記錄和輸出：
- 日誌記錄：使用單例模式的 Logger 類，將重要事件和錯誤記錄到日誌文件中。
- JSON 輸出：使用適配器模式的 JSONWriter 類，將掃描結果輸出為 JSON 格式，便於後續分析和處理。

3. 單元測試的設計想法 (Unit Test Design Principles

為了確保系統的可靠性和安全性，特別是從資安的角度，我們使用了 Google Test 框架編寫了全面的單元測試。設計原理如下：

- 完整性與覆蓋率：測試涵蓋核心功能，包括正常情況、異常情況和惡意輸入，確保代碼在各種情況下都能正常運行。
- thread安全性測試：對多線程相關的代碼，如 Logger 類，進行線程安全性的測試，確保沒有競態條件（Race Condition）發生。

- 安全性測試：模擬惡意輸入和攻擊向量，測試系統對異常和惡意情況的處理，防止潛在的安全漏洞。
- 使用專業工具：採用業界標準的測試框架，增強測試的可靠性和可維護性。

4. 設計模式的應用 (Design Pattern Applications)

為了提高系統的可維護性、擴展性和靈活性，我們在設計中應用了以下設計模式：

4.1 Singleton 模式 (Logger 類)

目的：確保全局只有一個 Logger 實例，統一管理日誌記錄。
實現：
使用靜態方法 getInstance() 獲取唯一實例。
私有化構造函數，防止外部創建新實例。
使用 std::mutex 保證線程安全。

4.2 Factory 模式 (ProcessScanner 類)

目的：為掃描器的創建提供統一接口，支持多種掃描策略的擴展（如字符串匹配、正則表達式等）。
實現：
定義掃描器接口 IScanner，具體實現繼承該接口。
使用 ScannerFactory 根據需要創建對應的掃描器實例。

4.3 Adapter 模式 (JSONWriter 類)

目的：封裝第三方 JSON 庫，為系統提供簡單的 JSON 輸出接口，降低與外部庫的耦合。
實現：
JSONWriter 將內部數據結構轉換為 JSON 格式，並使用第三方庫進行序列化。
客戶端只需調用簡單的方法，而無需關注 JSON 庫的細節。

4.4 Proxy 模式 (eBPFProgram 類)

目的：作為用戶態程序與內核態 eBPF 程式之間的代理，封裝底層細節，保護內核資源。
實現：
eBPFProgram 類管理 eBPF 程式的加載、啟動和停止。
提供事件監聽器，將內核事件轉發給用戶態程序。

5. 線程處理 (Thread Handling)

系統中涉及多線程操作的部分，主要關注點如下：

Logger 類的線程安全性：

多個線程可能同時寫入日誌文件。
使用 std::mutex 保護寫入操作，防止競態條件。

eBPFProgram 類的事件監聽器：

使用單獨的線程運行事件監聽器，異步處理內核事件。
使用 std::atomic 控制線程的運行狀態。
使用 std::mutex 和 std::vector 實現事件緩衝區，保證數據一致性。
線程安全性測試：

單元測試中模擬多線程環境，確保系統在高併發情況下仍能正常運行。
測試 Logger 的線程安全性，驗證沒有競態條件

6. 使用指南 (Usage Guide)

編譯方法:

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
make
```

運行:

```
sudo ./realtime_detection
```