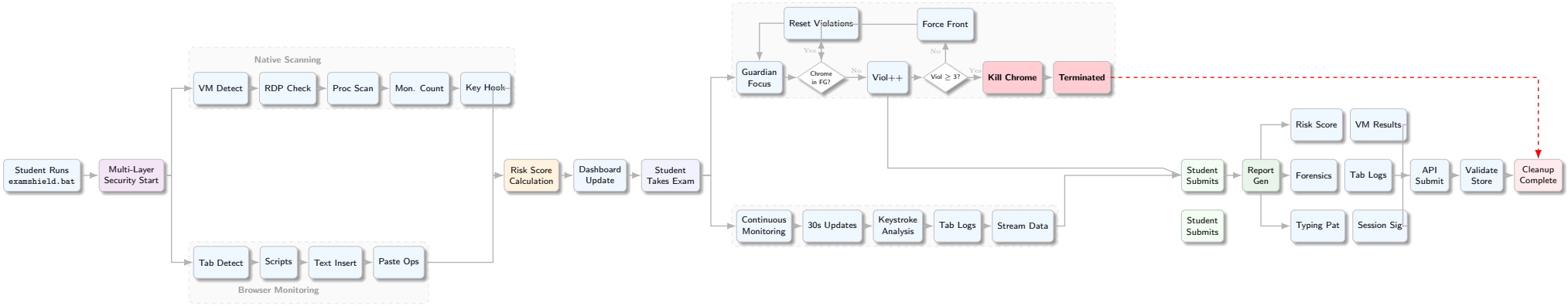


CORE EXECUTION UNITS

1. Native Security Monitor

File: `./native/main.cpp` → `./ExamShield.exe`
The central kernel for deep system scanning.

- **VM Detection:** Analyzes CPUID hypervisor bits, WMI queries, registry signatures, and VM-specific threads.
- **RDP/Remote:** Calls `GetSystemMetrics(SM_REMOTESESSION)` and scans for tools (TeamViewer, AnyDesk, VNC).
- **Hardware Hooks:** Low-level keyboard hooks analyze typing speed and hardware vs. software injection. (This is a two layer security, if the hardware keyboard is not working but software is getting signals, this means a VM or a false connection, this is a rare case - it implies that our VM analyzer failed to detect a VM, this can happen but very rare, it will need a complete new signature of a VM)
- **Display Auditing:** Enumerates connected monitors to detect secondary display usage.

2. Guardian Process (Watchdog)

File: `./native/guardian.cpp` → `ExamGuardian.exe`
Ensures environment persistence and focus.

- **Kiosk Enforcement:** Monitors Chrome process health; enforces foreground focus.
- **Violation Protocol:** Auto-refocuses window on switch. Terminates session upon ≥ 3 violations.

3. Chrome Extension Bridge

Files: `./extension/bg.js`, `./extension/content.js`
Links the web context to native system binaries.

- **Background:** Manages native messaging bridge, connection retries (max 5), and kiosk locking.
- **Content:** Captures DOM events for tab visibility/blur, analyzes keystroke bursts, and flags paste operations.

WEB & INFRASTRUCTURE

4. Web Interface

File: `./webface/src/page.tsx`
The candidate-facing dashboard and visualizer.

- **Live Dashboard:** Renders color-coded risk scores and status of VMs, RDP, and Process lists.
- **Forensics:** Tracks tab-switch timestamps and visualizes typing patterns in real-time.
- **Submission:** Packages session metadata with the exam payload.

5. API Endpoint

File: `./webface/src/api/submit/route.ts`
Server-side validation and log retention.

- Validates session tokens and generates submission IDs.
- Archives keystroke forensic data and comprehensive security logs.

ALGORITHMIC RISK SCORING

Cumulative Risk Formula

The system calculates a total risk score based on weighted detections, capped at 100.

$$\text{Risk} = \min \left(100, \begin{array}{l} 47 \cdot \mathbb{I}_{\text{VM}} \\ +36 \cdot \mathbb{I}_{\text{SuspiciousProcess}} \\ +27 \cdot \mathbb{I}_{\text{RDP}} \\ +27 \cdot \mathbb{I}_{\text{MultiMonitor}} \\ +10 \cdot \text{KeystrokeRisk} \end{array} \right)$$

SECURITY RISK MATRIX

Vector	Detection Method	Pts	Sev
Virtual Machine	CPUID + WMI + Reg	47	High
Remote Tools	Process Enumeration	36	High
RDP Session	System Metrics API	27	Med
Multi-Monitor	Display Enumeration	27	Med
Tab Switching	Visibility/Focus Events	–	Med
Keystroke	Heuristic Analysis	10	Low+

DEFENSE MECHANISMS

Anti-Virtualization

A multi-layered approach combining hardware interrogation (CPUID hypervisor bit) with software footprinting (VM-specific processes, registry keys, and WMI `Win32_ComputerSystem` checks).

Keystroke Forensics

- **Injection Detection:** Distinguishes between hardware interrupts and software-simulated input.
- **Speed Analysis:** Flags typing speeds > 300 WPM as physically impossible.
- **Pattern Recognition:** Detects low-variance "robotic" inter-key latencies.
- **Copy-Paste:** Correlates large text insertions with the absence of preceding keystrokes.

Session Integrity

Guarantees data authenticity via cryptographic session tokens and SHA256 data signing. All violations are tracked in real-time to prevent tampering during submission.

The whole idea is to build a .exe(writen in pure C++ to work with system level operations and operate with OS more easily) which runs Chrome in kisok mode(from the .exe itself), the user will also be required to install an extension which will allow this .exe to talk with the running chrome tab(this is only one tab in full screen). There will be two level scans, from the OS itself using the .exe and browser level scan, these data will be analysed on IICPC servers and logs will be stored(each 30sec - big check, 5sec small check). These will also be send to the client system, when there is any suspicious activity, the .exe automatically flags it and sends a close notification to the server and the client with the latest collected data. After this, it is at the IICPC team to block the results or pass the check, At a later stage once we have more data, we can simply train a small ML model which classifies the flow as cheating/non-cheating. This data-ML model is for keystroke only, as if there is any system level interference from the client side it automatically uses an if-else statement to flag cheating.

SYSTEM ARCHITECTURE OVERVIEW

The ExamShield system operates on a three-tier architecture designed to bridge low-level system monitoring with a high-level web interface.

1. **Native Host (C++)**: Provides system-level access (global keystrokes, process list, window management) unavailable to standard web browsers.
2. **Browser Extension (JavaScript)**: Acts as a bridge between the Native Host and the Web Interface, and performs DOM-level monitoring.
3. **Web Interface (Next.js)**: The user-facing exam application that aggregates security data and submits it to the server.

COMPONENT ANALYSIS

1. Native Component (C++)

The native component is split into two primary responsibilities: Data Collection and Environment Enforcement.

7.1.1 Native Messaging Host (`main.cpp`)

This executable communicates with the Chrome Extension via `stdin/stdout` using the Native Messaging protocol.

- **KeystrokeMonitor Class**:
  - **Hooking**: Uses `SetWindowsHookEx(WH_KEYBOARD_LL, ...)` to install a low-level keyboard hook. This captures keystrokes globally, even when the browser is not focused.
  - **Analysis**:
    - `calculateVariance()`: Computes the variance in time intervals between keystrokes to detect robotic/automated typing.
    - `rapidSequences`: Tracks bursts of typing that exceed human capabilities.
  - **Risk Calculation**: Aggregates metrics into a risk score sent to the extension.
- **Cryptographic Functions**:
  - Uses Windows CryptoAPI (`wincrypt.h`) for SHA-256 hashing (`sha256`) and Base64 encoding (`base64`). This is likely used to sign security payloads to prevent tampering.

7.1.2 Window Guardian (`guardian.cpp`)

A separate process responsible for maintaining the integrity of the exam environment.

- **Window Enumeration**: Uses `EnumWindows` to find the Chrome window hosting the exam (matching title "ExamShield" or "localhost:3000").
- **Focus Enforcement**:
  - `isChromeInForeground()`: Checks if the active window is the exam window.
  - `bringChromeToFront()`: If the user alt-tabs away, this function forces the exam window back to the foreground using `SetForegroundWindow` and `ShowWindow(SW_MAXIMIZE)`.
- **Process Monitoring**: Uses `EnumProcesses` and `GetModuleBaseNameA` to scan for prohibited applications (e.g., remote desktop tools, communication apps).

2. Browser Extension

The extension serves as the secure pipeline and DOM monitor.

7.2.1 Background Script (`bg.js`)

- **Native Connection**:
  - `connectNative()`: Establishes a persistent port to `com.exam.shield` using `chrome.runtime.connectNative`.

- Handles disconnection events with automatic retry logic (exponential backoff or fixed interval).
- **Message Routing**: Receives JSON payloads from the Native Host (containing risk scores) and forwards them to the active tab's Content Script.

7.2.2 Content Script (`content.js`)

Injected into the DOM of the exam page.

- **KeystrokeAnalyzer Class**:
  - Unlike the C++ monitor, this listens to DOM events (`keydown`, `input`).
  - **Paste Detection**: Specifically listens for `Ctrl+V` or large instantaneous text changes in input fields (`lengthDelta > 20`).
  - **Typing Profile**: Builds a local profile of typing speed and backspace usage to cross-reference with the native monitor.
- **Focus Tracking**: Monitors `window.onblur` and `window.onfocus` to track tab switching attempts within the browser.

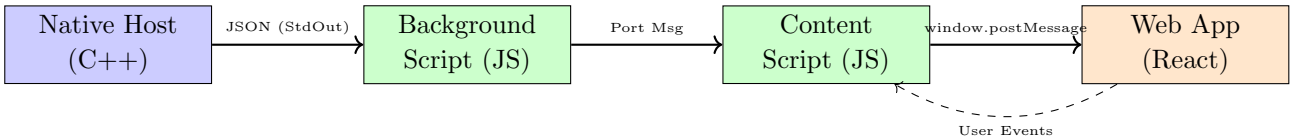
3. Web Interface (Next.js)

The frontend application located in `webface/src/app`.

7.3.1 Event Aggregation (`page.tsx`)

- **Message Listener**: A `window.addEventListener('message')` hook receives data from the Content Script.
- **State Management**:
  - `securityData`: Stores the risk score and flags (VM detected, RDP detected) from the C++ layer.
  - `tabSwitchCount`: Tracks how many times the user left the tab.
- **Submission**: When the exam is submitted, it bundles the exam answers with the `securityData` and `keystrokeData` for backend verification.

DATA FLOW SEQUENCE



SECURITY MECHANISMS DETAIL

Keystroke Dynamics

The system employs a dual-layer keystroke analysis:

1. **Native Layer**: Captures raw hardware interrupts. This detects if keystrokes are injected by software (macros) rather than physical hardware.
2. **Browser Layer**: Analyzes the resulting text insertion.

Detection Logic: If *Native Keystrokes* ≈ 0 but *Text Length* increases significantly, a **Paste/Injection** event is flagged.

Anti-Tamper

The `guardian.cpp` process runs independently. If the user attempts to terminate the Chrome extension, the native process persists and can potentially force-close the browser or log the violation (though specific termination logic depends on the full implementation of `guardian.cpp`).