Laptop tamper detection system: A software-based approach for monitoring unauthorized laptop movement

O. Krishna Raju[*1], B. G. Deepika[2], L. Devidas[3], Ch. Nikshitha[4]
Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management (HITAM), Hyderabad, Telangana, India

DOI: 10.5281/zenodo.24006324

Abstract:
Laptops store personal, academic, and professional information and are frequently carried between homes, classrooms, offices, and public places. Conventional security mechanisms such as passwords and PINs mainly control digital access; by themselves, they do not provide continuous awareness of physical movement or possible tampering. This paper presents the design of a Laptop Tamper Detection System, a browser-based security monitoring prototype intended to demonstrate how suspicious movement events can be identified, reported, and recorded. The system provides a protection dashboard, configurable detection sensitivity, a tamper-event decision process, alarm and browser-notification interfaces, and a timestamped event history. The current academic demonstration uses software-generated or simulated movement events so that the workflow can be tested without additional physical sensors. Browser motion-sensor integration is treated as a possible extension and is not claimed as a validated feature of the present simulation. The paper describes the system architecture, rule-based detection workflow, implementation technologies, representative test cases, benefits, and limitations. The proposed design offers a low-cost starting point for demonstrating physical-security monitoring, while further device-level testing is required before it can be considered a dependable real-world anti-theft solution.

Keywords:
Laptop Security, Tamper Detection, Movement Monitoring, Browser-Based Application, Security Alerts, Event Logging, Software Prototype.

1. Introduction:
Laptops are used to store and access personal documents, academic work, business information, and other valuable data. Their portability makes them convenient, but it also means that they may be left unattended in classrooms, libraries, offices, laboratories, or shared spaces. A person may attempt to move or handle a laptop without the owner's permission. Such activity is a physical-security concern that is not directly addressed by ordinary login passwords.

Traditional laptop protection commonly relies on passwords, PINs, device tracking, and physical locks. These measures serve useful purposes, but they do not always provide an immediate indication that a laptop has been moved or disturbed. A monitoring mechanism that can identify a suspicious movement event and create a visible alert can provide an additional layer of awareness.

This paper presents the Laptop Tamper Detection System, also referred to in the prototype interface as Laptop Sentinel. It is designed as a browser-based monitoring interface with a protection status, sensitivity setting, alert mechanism, and event history. The academic prototype demonstrates the workflow through software-controlled movement/tamper events. Actual sensor access depends on device and browser support and is considered an integration step for future development.

2. Problem statement:
Laptops contain valuable personal, academic, and professional information, making them a target for theft and unauthorized handling. Existing security measures such as passwords and PINs mainly protect digital access but do not independently detect physical movement, lid opening, or other tampering attempts. Some tracking

and sensor-based solutions may require additional hardware, connectivity, or installation, which can increase cost and complexity.

There is therefore a need for a simple monitoring system that demonstrates how a suspicious physical event can be detected, how an alert can be generated, and how the event can be recorded for later review. The project addresses this need through a browser-based prototype that presents security status, accepts movement-event input, applies a configurable threshold, and records alerts with timestamps.

3. Objectives:

The main objectives of the project are:

1. To design a simple browser-based interface for laptop security monitoring.
2. To represent laptop movement and possible tampering events through sensor values or software-generated test events.
3. To apply a configurable sensitivity or movement threshold to classify an event as normal or suspicious.
4. To provide an alarm sound and browser notification interface when a suspicious event is detected, subject to browser permissions and settings.
5. To maintain a security-event history containing relevant event information and timestamps.
6. To display the current protection state, sensor/event status, sensitivity, and previous alerts on a dashboard.
7. To provide a prototype that can be demonstrated without requiring external physical hardware.

4. Literature review:

The project presentation identifies four related works. Wankhade et al. (2022) are described as using GPS, GSM, motion sensors, and IoT for laptop tracking and theft alerts. Santika et al. (2023) are described as combining IoT, a pressure sensor, a camera, and tracking to support image capture and owner notification. Sai Kiran et al. (2020) are described as using an IoT testbed and machine-learning classifiers to identify network attacks from sensor data; the presentation notes that this work focuses on network attacks rather than physical laptop tampering. Swetha et al. (2026) are described as using IoT sensors and automatic alerts for theft detection.

These works, as summarized in the project presentation, indicate that laptop security research may use tracking, sensors, cameras, IoT connectivity, or automated alerts. They also highlight practical constraints such as additional hardware and internet requirements. The present project takes a prototype-oriented approach: it focuses on the monitoring workflow, configurable event threshold, browser alert interface, and timestamped event records in a lightweight web application.

Bibliographic information for the four studies (full paper titles, venue, volume, issue, page numbers, and DOI/URL) was not included in the supplied presentation. Their details must be verified from the original publications before a final academic submission; they are not expanded or guessed in this paper.

5. Research gap:

According to the literature summary in the project presentation, several existing approaches emphasize password protection, tracking, physical locks, or sensor-based systems. Some approaches depend on additional hardware and/or internet connectivity. The project presentation identifies a need for a simple interface that can demonstrate suspicious movement detection, adjustable sensitivity, instant user alerts, and security-event records.

The research gap addressed by this prototype is the integration of these basic monitoring functions into one accessible browser-based demonstration. The contribution is not a new physical sensor or a proven theft-prevention mechanism. Rather, it is a software prototype that models the decision and response workflow and provides a foundation for later connection to supported device sensors.

6. Proposed system:

The proposed Laptop Tamper Detection System is organized around a user interface, movement-event input, processing logic, decision stage, alert module, event storage, and dashboard. The user opens the Laptop Sentinel website and activates the protection workflow. Movement-related values are then supplied to the processing logic. In the current demonstration, events may be simulated through the software rather than measured by a dedicated external sensor.

The processing logic evaluates the received movement value against a configurable threshold. If the value does not exceed the threshold, the interface retains or displays a normal/secure state. If the value meets the

suspicious-event condition, the alert module can play an alarm sound and request a browser notification. The event is added to the security history with a timestamp and is shown on the dashboard.

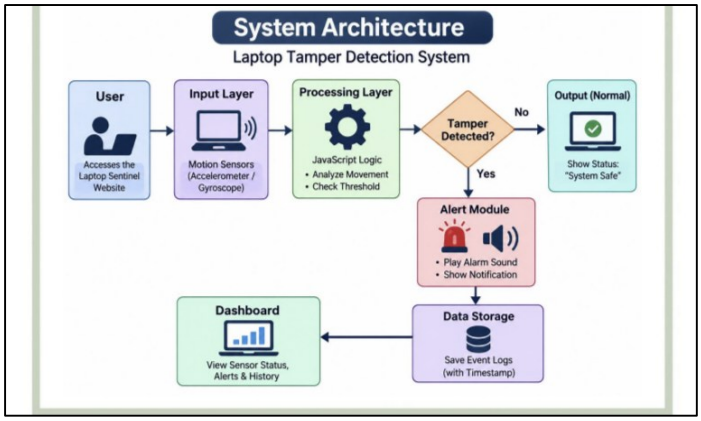
The architecture is designed to separate event input, decision logic, alert handling, and presentation. This separation makes it easier to replace simulated input with a verified sensor source in a later version without changing the overall monitoring workflow.

7. System architecture:

The system follows a sequence of functional layers, as represented in the architecture diagram in the project presentation:

1. **User layer:** The user accesses the Laptop Sentinel website and starts or stops protection.
2. **Input layer:** Movement-related values are supplied to the application. The presentation identifies accelerometer/gyroscope data as the intended sensor input; the current demonstration can use simulated values where live sensor access is unavailable.
3. **Processing layer:** JavaScript logic reads the input and evaluates it against the configured movement threshold and sensitivity setting.
4. **Decision layer:** The system determines whether the event is within the normal range or should be treated as suspicious.
5. **Alert layer:** For a suspicious event, the application activates the alarm interface and requests a browser notification, where permitted.
6. **Data-storage layer:** Security events and timestamps are retained in the event history. The project presentation lists local storage as the browser-side storage mechanism.
7. **Dashboard layer:** The interface displays protection status, movement/event information, alert totals, sensitivity, and previous security events.

Normal path: User → Input → Processing → Decision → Secure status. Alert path: User → Input → Processing → Suspicious event → Alarm/notification → Event record → Dashboard.



8. Methodology:

The system workflow is organized into the following steps:

1. User access: Open the Laptop Sentinel web application in a supported browser.
2. Protection activation: Start the monitoring/protection mode from the dashboard.
3. Event input: Receive movement-related values or generate a software-based test event.
4. Sensitivity configuration: Select the required sensitivity or minimum movement threshold.
5. Event evaluation: Compare the incoming movement value with the configured threshold.
6. Tamper decision: Classify the event as normal or suspicious according to the rule.
7. Alert generation: For a suspicious event, activate the alarm sound and request a browser notification, if enabled and permitted.
8. Event recording: Store the event description, severity/status, and timestamp in the event history.
9. Dashboard update: Refresh the visible protection state and alert history so the user can review the event.

The methodology is rule-based. It does not require a machine-learning model. Its behavior depends on the quality and source of the movement input, the selected threshold, and browser permissions.

9. Detection logic:

Let m represent the movement value supplied to the application and τ represent the configured movement threshold. A simplified decision rule is:

TamperStatus = Suspicious, if $m \geq \tau$; otherwise TamperStatus = Normal.

The exact movement calculation and threshold units depend on the input source and implementation. If live accelerometer/gyroscope data is integrated, the application must define how axis readings are combined, calibrated, filtered, and converted into a comparable movement measure. For a software-only demonstration, a test event can supply a representative value to exercise the same decision branch.

An event record may be represented as E = {event_id, timestamp, event_type, movement_value, threshold, status}. The dashboard can use these records to display recent activity and the total number of alerts. This representation describes the intended logical record; the exact fields should be aligned with the final source code.

10. Tools and technologies:

HTML5 is used to structure the web interface. CSS3 is used for the layout, colors, responsive presentation, and dashboard styling. JavaScript implements the application behavior, threshold evaluation, interface updates, and event-handling logic. The Device Motion/Orientation browser APIs are relevant to a future live-sensor integration, subject to browser and device support and user permission. The Web Audio API can be used to generate alarm sounds, while the Notifications API can request browser notifications. Browser Local Storage can retain event history on the same browser profile. Visual Studio Code is identified in the presentation as the development environment, and a web browser is used to run and test the prototype.

These technologies describe the project stack as presented by the team. The use of a browser API does not guarantee that every laptop exposes motion sensors or that every browser permits access. The current software demonstration should therefore be described as simulated event monitoring unless live sensor readings have been verified on the target device.

11. Experimental setup and test cases:

The project presentation shows a browser-based Laptop Sentinel dashboard with protection controls, sensitivity settings, movement-status information, alarm controls, and alert indicators. To evaluate the workflow, the following representative functional tests can be performed. The table describes expected behavior, not measured results.

Table. 1: Representative functional test cases

ID	Test scenario	Input / action	Expected outcome
T1	Start protection	User selects Start Protection	Dashboard changes to active/protected state
T2	Normal event	Movement value remains below threshold	No tamper alert; normal/secure status remains
T3	Suspicious event	Movement value reaches or exceeds threshold	Suspicious status and alert workflow are triggered
T4	Sensitivity change	User changes sensitivity/threshold	Subsequent event decisions use the updated setting
T5	Alarm enabled	Suspicious event occurs with alarm enabled	Alarm sound is requested/played if browser allows
T6	Notifications enabled	Suspicious event occurs after permission is granted	Browser notification is requested/displayed if permitted
T7	Event history	One or more alerts are generated	Event record and timestamp appear in dashboard history
T8	Stop/reset	User stops protection or resets system	Monitoring state and dashboard follow the selected control behavior

12. Results and discussion:

The prototype interface demonstrates the main stages of a laptop security-monitoring workflow: activating protection, configuring sensitivity, displaying movement/event status, generating an alert, and reviewing previous events. The dashboard provides a single place to observe the security state and alert history. Software-generated events allow the decision and notification flow to be demonstrated without connecting additional physical hardware.

The results should be interpreted as prototype functionality rather than proof of reliable real-world tamper detection. The supplied PPT contains screenshots of the dashboard but does not provide controlled measurements of detection accuracy, false-positive rate, notification delay, or performance across different laptop models. These values are therefore not claimed. Live sensor tests and repeatable experiments are required before quantitative performance can be reported.

13. Advantages:

1. Provides a simple, centralized dashboard for monitoring security status and events.
2. Demonstrates a configurable threshold-based detection workflow.
3. Supports an alarm and browser-notification response path.
4. Maintains a timestamped event history for review.
5. Can be demonstrated as a software prototype without extra physical hardware.
6. Uses common web technologies and can be extended with device-level sensor input.

14. Limitations:

1. Software-generated movement events do not prove that actual physical movement is being sensed.
2. Live motion-sensor access varies by laptop hardware, operating system, and browser support.
3. Browser notifications require permission and may be blocked or suppressed by browser settings.
4. An alarm played through the laptop may not be effective if the device is muted, powered off, or inaccessible.
5. Threshold-based detection may produce false alerts or miss events if the threshold is not calibrated appropriately.
6. Local browser storage is generally limited to the relevant browser profile and device; it is not a remote backup or owner-notification service.
7. The prototype does not by itself prevent theft, guarantee recovery, or replace operating-system security, encryption, or physical safeguards.

15. Future scope:

Future development can connect the application to supported accelerometer or gyroscope sources and validate readings on different laptop models. Further work may include calibration procedures, noise filtering, lid-state monitoring where supported, secure remote notifications, cloud-based event backup, authenticated owner accounts, mobile alerts, and controlled evaluation of detection accuracy and false alarms. Any camera-based evidence capture or external hardware integration should be added only after privacy, permission, compatibility, and security requirements are addressed.

16. Conclusion:

This paper presented the design of a Laptop Tamper Detection System intended to provide a browser-based demonstration of physical-security monitoring. The system organizes movement-event input, threshold-based processing, a normal-or-suspicious decision, alarm and browser-notification handling, timestamped event records, and dashboard visualization. Its current prototype can demonstrate the workflow through software-generated events without requiring additional physical hardware.

The project provides a practical foundation for further development, but the current demonstration should not be interpreted as validated live sensor-based theft detection. Future work must integrate and test real sensor input, evaluate false alerts and missed detections, and assess notification reliability across supported devices and browsers. With these qualifications, the system illustrates how a lightweight web application can support awareness of possible unauthorized laptop movement.

17. References:

(1) N. R. Wankhade et al., work on laptop tracking and theft alerts using GPS, GSM, motion sensors, and IoT, 2022. Full title and publication details were not provided in the project presentation; verify before submission.

(2) Reva Santika et al., work on laptop tracking using IoT, pressure sensor, camera, and owner notification, 2023. Full title and publication details were not provided in the project presentation; verify before submission.

(3) K. V. V. N. L. Sai Kiran et al., work on network-attack identification using an IoT testbed and machine-learning classifiers, 2020. Full title and publication details were not provided in the project presentation; verify before submission.

(4) Dr. A. Swetha et al., work on theft detection and automatic alerts using IoT sensors, 2026. Full title and publication details were not provided in the project presentation; verify before submission.
(5) World Wide Web Consortium (W3C), "Device Orientation and Motion," W3C Recommendation/Specification. https://www.w3.org/TR/orientation-event/ (accessed Nov. 5, 2026).
(6) Mozilla Developer Network (MDN), "Notifications API." https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API (accessed Nov. 5, 2026).
(7) Mozilla Developer Network (MDN), "Web Audio API." https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API (accessed Nov. 5, 2026).
(8) Mozilla Developer Network (MDN), "Window: localStorage property." https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (accessed Nov. 5, 2026).

