

LUMEN: A Modular, Local-First Architecture for Lightweight Host-Based Threat Detection on Windows Endpoints

Dr. Mrunal Pathak, Hrishikesh Wadile, Pranav Dhote, Sanaya Kulkarni

Department of Information Technology

Final-Year B.Tech, Department of Information Technology

AISSMS Institute of Information Technology, Pune, India

mrunal.pathak@aissmsioit.org, hrishikesh.wadile@aissmsioit.org

pranav.dhote@aissmsioit.org, sanaya.kulkarni@aissmsioit.org

Abstract Endpoint threats increasingly evade conventional signature-based tools, while many learning-based alternatives depend on cloud services and heavy resource budgets that are unsuitable for ordinary personal and small-office machines. This paper presents LUMEN, a modular host-based threat detection system for Windows that performs all collection, analysis, storage, and response locally. LUMEN organises endpoint defence into four cooperating layers: telemetry collection, independent detection, risk and alert evaluation, and local storage. Host-level telemetry covering executable and process activity, Windows Event Logs, and browser URL activity is routed through source-specific detectors whose outputs feed a common alert and response layer. The system is implemented in Rust with native Windows interfaces, local ONNX-based inference, SQLite persistence, and a Dioxus desktop interface. This paper focuses on the design rationale, architecture, data flow, and implementation of LUMEN, and demonstrates through functional validation that its components operate together end to end. Quantitative evaluation of the individual detectors and extension to multi-source correlation are outside the scope of this work.

Keywords: host-based intrusion detection, endpoint security, local inference, modular architecture, Windows, Rust

I. INTRODUCTION

Endpoints remain the primary point of entry and execution for malware, phishing-delivered payloads, and malicious web content. Host-based intrusion detection has therefore been studied extensively, and recent reviews describe a broad landscape of data sources, techniques, datasets, and open challenges [1, 2]. Practical deployment on ordinary machines nevertheless raises concerns that are often secondary in the research literature: detectors must be light enough to run beside everyday workloads, and they must not require sensitive host telemetry to leave the device.

Reducing the cost of detection has long been a design concern. Work on feature reduction, for example, shows that a smaller input representation can substantially lower computational demand while retaining strong detection behaviour [1]. Other recent work reduces overhead through host-level aggregation and efficient host-based analysis [3, 4]. LUMEN adopts the same philosophy at the system level: each component is kept small, specialised, and replaceable rather than relying on a single heavy monolithic engine.

Existing approaches also leave a gap for users who need protection but lack enterprise infrastructure. Enterprise platforms typically assume centralised collection and analysis, whereas smaller organisations are better served by hybrid designs that combine several detection mechanisms and risk scoring [5]. LUMEN targets this space with a local-first architecture.



The contributions of this paper are:

- a layered and modular architecture for local host-based threat detection on Windows;
- a unified data flow in which heterogeneous telemetry sources share a common detection-to-alert interface;
- a description of the Rust-based implementation, including local model deployment, storage, and desktop interface; and
- a functional validation showing that the components cooperate end to end.

The remainder of the paper is organised as follows. Section II reviews related work. Section III states the design requirements and Section IV presents the architecture. Sections V and VI describe the host-based modules and the integration of learned detectors. Section VII covers alerting and response, Section VIII the implementation, and Section IX the functional validation. Sections X–XII discuss resource-aware design, limitations, and conclusions.

II. RELATED WORK

This section positions LUMEN against prior work along the dimensions that shape its architecture. It is deliberately brief, as the paper concerns system design rather than comparative detector performance.

A. Process and Malware Behaviour

Host threat detection from process and audit information has been explored with graph representation learning to improve efficiency [6], and program anomaly detection has been studied with attention to computational requirements [7]. Learned approaches to malware detection are surveyed in [8]. These works motivate a dedicated process-level detection path in LUMEN but do not address a complete local endpoint system.

B. Windows Event and Log Telemetry

System event data is a rich source of evidence on Windows hosts. Machine learning applied to Sysmon telemetry has been shown to support detection of multi-step attacker activity [9], and recent work addresses the processing and normalisation of security event logs, including locally run models [10]. LUMEN relies on the same principle of converting raw log entries into structured, model-ready records.

C. Browser and URL Protection

Web-borne threats are commonly addressed by classifying URLs, and real-time browser-integrated detection has been demonstrated using lightweight feature-based models [11]. Deep learning has also been applied to URL classification [12]. LUMEN adopts the browser-to-local-detector pattern, keeping URL analysis on the host.

D. Risk, Alerting, and Latency

Detection output is only useful if it is turned into actionable alerts. Dynamic, learned thresholds have been proposed to reduce repeated and spurious alerts in security monitoring [13], and low-impact, near real-time risk assessment has been investigated for infrastructures with limited resources [14]. The end-to-end latency of detection has also been argued to be an underreported but important quality criterion [15]. These observations motivate LUMEN's separate risk layer and its event-driven operation.

E. Summary

Prior work offers strong individual components, but few describe a self-contained, local-first system that unifies several host telemetry sources behind a common alert interface. LUMEN contributes at this system level.



III. DESIGN REQUIREMENTS AND OBJECTIVES

The design of LUMEN is guided by four requirements.

A. Lightweight Operation

The system must coexist with normal user workloads. Component choices therefore favour compact detectors, an efficient systems language, and event-driven processing over continuous heavy scanning. Detection latency is treated as a first-class concern rather than an afterthought [15].

B. Local and Privacy-Preserving Processing

Telemetry such as process activity, system events, and visited URLs is sensitive. LUMEN performs collection, inference, storage, and response entirely on the host, with no requirement for cloud upload.

C. Modular Detection

Different telemetry types need different representations and detectors. Each source is handled by an independent detector so that it can be developed, replaced, or updated without modifying the remainder of the pipeline. This mirrors the hybrid, multi-mechanism approach recommended for resource-limited organisations [5].

D. Extensibility

The architecture must admit new telemetry sources and detectors through a common interface, so that later capability can be added without redesigning the core.

IV. LUMEN SYSTEM ARCHITECTURE

A. Overall Architecture

LUMEN is composed of four layers, shown in Fig. 1: a telemetry collection layer, a detection layer, a risk and alert layer, and a local storage layer. Information flows in one direction from sources toward the user, while all layers read from and write to the local store.

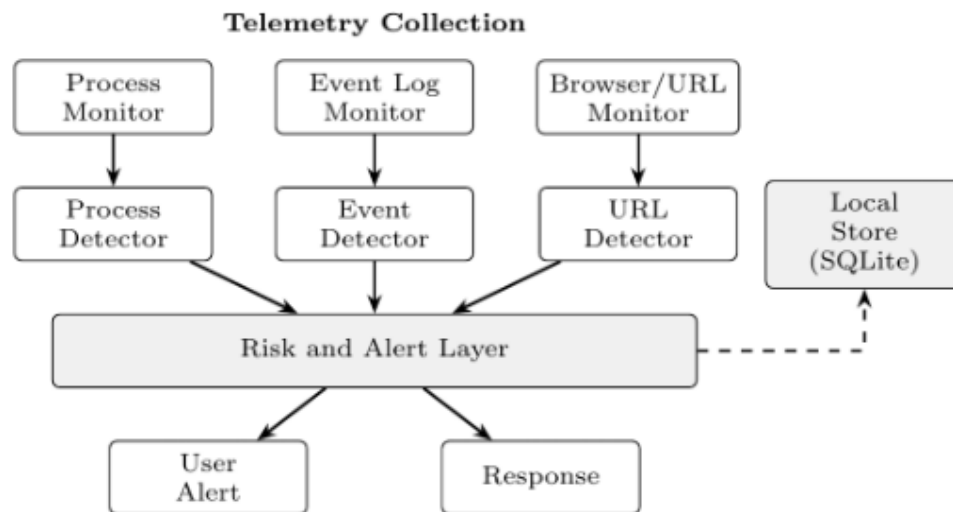


Fig. 1. High-level architecture of LUMEN. Source-specific detectors feed a common risk and alert layer; all layers persist state to a local store.



B. Telemetry Collection Layer

This layer gathers host observations from three sources: executable and process activity, Windows Event Logs, and browser URL activity. Each collector normalises its observations into a source-tagged event record, which hides collection details from downstream layers.

C. Detection Layer

Each source has a dedicated detector that consumes its normalised records and produces a detection result consisting of a score and supporting context. Detectors are independent and share only the result interface.

D. Risk and Alert Layer

The risk layer maps detector outputs to a common severity scale and decides whether an alert is raised. Treating risk evaluation as a distinct layer follows established practice in adaptive alert thresholding and low-impact risk assessment [13, 14].

E. Storage Layer

Events, detection results, alerts, and user decisions are persisted locally. The store supports review of past activity and provides state that survives restarts, without any external dependency.

V. HOST-BASED DETECTION MODULES**A. Process and Executable Monitoring**

The process module observes the creation of processes and the executables backing them. Static properties of the executable and context of the launch are transformed into model inputs for the process detector. Process-centred host detection [6], program anomaly detection [7], and learned malware detection [8] provide the background for this module, which LUMEN realises as a deliberately compact local component.

B. Windows Event Log Monitoring

The event module subscribes to relevant Windows Event Log channels and converts entries into structured records. Event-derived signals are informative for detecting suspicious behaviour such as lateral movement [9], and structured treatment of security event logs is an active research topic [10]. The event detector examines these records and emits a detection result in the common format.

C. Browser and URL Monitoring

Browser activity is exposed to LUMEN through a local interface. Each visited URL is passed to the URL detector, which scores it and returns the result to the alert layer. This follows the browser-integrated, real-time pattern of [11] while ensuring that no URL leaves the machine for analysis; alternative detector families [12] can be accommodated behind the same interface.

VI. LOCAL MACHINE-LEARNING INTEGRATION**A. Feature Extraction**

Each collector is paired with a feature extraction stage that converts raw telemetry into the numeric representation expected by its detector. Because representations differ across telemetry types, extraction is source-specific, and it is kept lightweight in keeping with the reduction philosophy of [1].

B. Independent Detector Architecture

Each telemetry source is processed by an independent model. This decoupling allows any detector to be retrained or replaced without affecting the others, and avoids forcing heterogeneous data into a single shared model.



C. Local ONNX Inference

Models are deployed in the ONNX format and executed locally. This provides a runtime-neutral deployment path, removes any need for a remote inference service, and keeps the application self-contained.

D. Detector-to-Alert Integration

Every detector emits the same result structure. This is the only contract between the detection and risk layers, so a detector's internals are opaque to the rest of the system. Fig. 2 summarises the per-source pipeline.

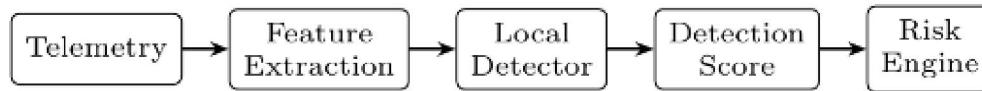


Fig. 2. Generic per-source detection pipeline in LUMEN.

VII. ALERT AND RESPONSE ARCHITECTURE

A. Detection Events

A detection event couples the originating source, the detector score, and contextual metadata. Events are persisted before any user-facing action is taken.

B. Alert Generation

The risk layer converts a detection event into an alert with an assigned severity when the evaluated risk warrants attention. Severity drives how prominently the alert is presented, and aims to avoid the alert fatigue associated with repeated, low-value notifications [13].

C. User Interaction

Alerts are surfaced in the desktop interface with the information required for an informed decision. The user remains in control of the final action.

D. Response Mechanisms

Responses are executed locally according to the decision taken, and the outcome is recorded in the local store so that the history of alerts and actions is auditable.

VIII. IMPLEMENTATION

A. Rust-Based Implementation

LUMEN is implemented in Rust, chosen for memory safety and low runtime overhead, both of which suit a continuously running security agent.

B. Windows Integration

Native Windows interfaces are used for process observation and event-log access, so that collection relies on facilities already provided by the operating system.

C. SQLite Storage

SQLite provides an embedded, serverless store for events, results, alerts, and decisions, consistent with the local-only requirement.

D. Desktop Interface

The user interface is built with Dioxus and exposes monitoring status, alerts, and response actions to the user.



E. Model Deployment

Detector models are packaged as ONNX artefacts and loaded by the application at runtime, which separates model lifecycle from application releases.

IX. FUNCTIONAL VALIDATION

This section validates that the components function together rather than measuring detector accuracy.

A. Process Detection

Executable and process events produced on a test host were correctly collected, passed to the process detector, and returned as detection results.

B. Event Detection

Windows Event Log entries were ingested, normalised, and scored by the event detector, with results reaching the alert layer.

C. URL Detection

URLs from browser activity were delivered through the local interface to the URL detector and the resulting scores were propagated.

D. End-to-End Alert Flow

For each source, a detection was traced through risk evaluation, alert generation, display in the interface, user action, and persistence in the local store, confirming that the full pipeline operates as designed.

X. RESOURCE-AWARE DESIGN

Resource consumption is addressed through design decisions rather than reported measurements: compact independent detectors, local inference without network round trips, an efficient native implementation, an embedded database, and event-driven collection. Together these choices aim to keep the footprint modest on commodity hardware, in line with prior emphasis on lightweight host-based designs [3, 4], feature reduction [1], and detection latency as a quality criterion [15].

XI. LIMITATIONS AND FUTURE EXTENSIONS

The current scope is Windows-focused and limited to three telemetry sources. Detection quality depends on the underlying detectors and is not assessed here. Future work will include rigorous quantitative evaluation of the detectors and the extension of the architecture to additional telemetry sources and to correlation across sources.

XII. CONCLUSION

This paper presented LUMEN, a modular, local-first host-based threat detection system for Windows. By separating telemetry collection, detection, risk evaluation, and storage, and by connecting independent detectors through a common result interface, LUMEN achieves a lightweight and privacy-preserving design that can be extended over time. The implementation in Rust with ONNX inference, SQLite, and a Dioxus interface, together with the functional validation, shows that the architecture operates coherently end to end.

REFERENCES

- [1] M. H. Nataj Solhedar and S. E. Alavi, "The effect of feature reduction on an ANFIS-based intrusion detection system," *Discover Artificial Intelligence*, vol. 6, art. no. 631, 2026, doi: 10.1007/s44163-026-01217-z.
- [2] H. Satılmış, S. Akleylek, and Z. Y. Tok, "A systematic literature review on host-based intrusion detection systems," *IEEE Access*, vol. 12, pp. 27237–27266, 2024, doi: 10.1109/ACCESS.2024.3367004.



- [3] Y. Katsura, A. Endo, I. Arai, and K. Fujikawa, "Efficient IDS for IoT networks using host-based data aggregation and multi-entropy analysis," *IEEE Access*, vol. 13, pp. 125406–125419, 2025, doi: 10.1109/ACCESS.2025.3589057.
- [4] Y. Nam, R. King, Q. Burke, M. Zhou, P. McDaniel, and T. Rosing, "Efficient host intrusion detection using hyperdimensional computing," in *Proc. IEEE Int. Conf. Big Data*, 2024, pp. 2645–2652, doi: 10.1109/BigData62323.2024.10825247.
- [5] A. Mersni, N. Šehović, N. Subašić-Hodža, N. Rustempašić, and M. Čeljo, "Hybrid intrusion detection system for small and medium enterprises," *IEEE Access*, vol. 13, pp. 216253–216271, 2025, doi: 10.1109/ACCESS.2025.3646112.
- [6] B. Xu *et al.*, "ProcSAGE: An efficient host threat detection method based on graph representation learning," *Cybersecurity*, 2024, doi: 10.1186/s42400-024-00240-w.
- [7] K. Wawryn and P. Widulinski, "Enhancing computer program anomaly detection: A decreasing-threshold approach," *Neural Computing and Applications*, vol. 37, pp. 16305–16326, 2025, doi: 10.1007/s00521-025-11344-w.
- [8] Y. Song *et al.*, "Application of deep learning in malware detection: A review," *Journal of Big Data*, vol. 12, art. no. 99, 2025, doi: 10.1186/s40537-025-01157-y.
- [9] C. Smiliotopoulos and G. Kambourakis, "Machine learning for lateral movement detection using Sysmon logs: An empirical comparison of imbalanced and resampled data," *International Journal of Information Security*, 2026, doi: 10.1007/s10207-025-01182-1.
- [10] R. Vaarandi and H. Bahşi, "Using large language models for template detection from security event logs," *International Journal of Information Security*, vol. 24, art. no. 104, 2025, doi: 10.1007/s10207-025-01018-y.
- [11] A. A. Alhemyari *et al.*, "Real-time phishing attack detection using machine learning," in *Proc. 5th Int. Conf. Emerging Smart Technologies and Applications (eSmarTA)*, 2025, doi: 10.1109/eSmarTA66764.2025.11132204.
- [12] T. N. Minh and N. D. Thai, "Enhanced phishing URL detection using convolutional neural networks: A deep learning approach," in *Proc. RIVF Int. Conf. Computing and Communication Technologies*, 2024, doi: 10.1109/RIVF64335.2024.11009054.
- [13] A. Kapera and M. Niemiec, "Dynamic risk thresholds for SIEM alerting based on machine learning," *IEEE Access*, vol. 13, pp. 121034–121047, 2025, doi: 10.1109/ACCESS.2025.3588441.
- [14] E. Kail, A. Riethné Nagy, R. Fleiner, A. Bánáti, and E. Rigó, "Low-impact, near real-time risk assessment for legacy IT infrastructures," *International Journal of Information Security*, vol. 24, art. no. 67, 2025, doi: 10.1007/s10207-024-00971-4.
- [15] S. Dash and J. M. Acken, "Intrusion detection latency: The neglected metric," *Cybersecurity*, vol. 9, art. no. 144, 2026, doi: 10.1186/s42400-026-00574-7

