IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510120

RansomGuard AI: An AI-Powered Ransomware Detection and Storage Protection System

Rithvik Rajmohan[1], Roheith R[2], Bharath R[3], P. Durgadevi[4]

Department of Computer Science and Engineering, SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[1-4]

Abstract: Ransomware continues to pose a severe threat to data availability by executing unauthorized encryption, rapid file alterations, and malicious extension changes across local file storage. This paper introduces RansomGuard AI, an integrated defense-in-depth storage surveillance and recovery system designed to monitor filesystem events, analyze behavioral risk indicators, and maintain cryptographically verified snapshot recovery points. Operating on a Python FastAPI backend paired with a reactive web dashboard, the framework continuously evaluates filesystem activity against key telemetry signals: modification rates, burst write frequencies, Shannon entropy shifts, and file extension mutations. To guarantee disaster recovery without relying on compromised storage, recovery copies are secured using immutable SHA-256 manifest checkpoints. The system includes a sandboxed synthetic attack simulator that reproduces ransomware-like behaviors inside an isolated environment, demonstrating complete monitoring, detection, audit logging, and point-in-time recovery workflows without exposing the host OS to live malware.

Keywords: Ransomware detection, filesystem monitoring, Shannon entropy, behavioral risk profiling, SHA-256 manifests, snapshot recovery, FastAPI, system security.

I. INTRODUCTION

A. Background and Motivation

Ransomware attacks remain one of the most destructive operational threats to modern digital infrastructure. Unlike stealthy exfiltration tools, ransomware deliberately compromises data availability by encrypting localized user files, altering filenames, appending unrecognized extensions, and demanding ransom payments for decryption keys. Because novel variants frequently bypass static signature-based detection mechanisms (e.g., traditional antivirus definitions), defensive paradigms must shift toward real-time behavioral surveillance coupled with resilient recovery mechanisms.

RansomGuard AI was developed to bridge the gap between early threat detection and reliable disaster recovery. Rather than operating as an unconstrained system-wide process monitor, the system focuses surveillance on a configured protected directory. By evaluating behavioral indicators in real time and maintaining cryptographically verified snapshots, RansomGuard AI ensures that even if local files suffer sudden corruption or bulk encryption, clean historical states can be restored reliably.

B. Research Contributions

This work makes five key contributions to practical endpoint defense:

1. **Targeted Filesystem Telemetry:** Implements real-time monitoring of file creation, modification, deletion, and movement events within designated storage boundaries.
2. **Multi-Indicator Behavioral Risk Scoring:** Combines Shannon entropy profiling, sliding time-window burst rates, and extension mutation tracking into a unified risk assessment model.
3. **Interactive Visual Dashboard:** Delivers continuous telemetry updates and visual alert management through a modern web-based interface backed by WebSockets.
4. **Cryptographic Snapshot Rollbacks:** Integrates point-in-time directory snapshotting tied to SHA-256 verification manifests to prevent tampered restorations.
5. **Safe Attack Simulation Sandbox:** Provides an isolated, non-destructive simulation module capable of replicating ransomware-style write bursts and file lock scenarios for testing and demonstration purposes.

II. LITERATURE REVIEW

Behavior-Based Detection

Signature-based detection tools inevitably struggle against zero-day ransomware variants that leverage code obfuscation or novel encryption routines. Consequently, behavioral detection has emerged as a cornerstone of modern malware analysis. Instead of scanning binary patterns, behavior-based systems observe process actions, such as rapid file modifications, suspicious directory traversal, mass renames, and structural changes to file headers. Because benign

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510120

operations (like compiling source code or unzipping archives) can occasionally mimic isolated suspicious actions, practical defense systems must correlate multiple signals across time windows rather than acting on single anomalies.

Shannon Entropy Analysis
Information entropy, as defined by Claude Shannon, measures the randomness or uncertainty inherent in a distribution of data. For a sequence of bytes, entropy $H(X)$ is calculated as:

$$H(X) = -\sum_{i=0}^{255} p(i) \log_2 p(i)$$

where $p(i)$ denotes the relative frequency of byte value i. The resulting score ranges from 0.0 (completely uniform data, such as a file filled entirely with zeros) to 8.0 bits per byte (purely random data). Because strong cryptographic algorithms produce ciphertext with high entropy approaching 8.0 bits/byte, tracking sudden spikes in file entropy serves as a strong heuristic for detecting ongoing encryption. However, high entropy is not exclusive to ransomware; legitimate compressed archives (e.g., .zip, .tar.gz), media streams, and encrypted containers also exhibit high entropy. Thus, entropy measurements must be evaluated alongside write rates and filename changes to avoid high false-positive rates.

Filesystem Telemetry
Real-time event capture forms the baseline of behavior monitoring. Filesystem watchers hook into operating system event subsystems to log changes in real time. Nevertheless, raw filesystem telemetry presents challenges: event streams can be noisy, operating systems may coalesce rapid modifications into single events, and elevated process privileges can alter event visibility. Effective systems must parse raw event feeds through sliding time windows to extract meaningful telemetry trends, such as operation frequencies and write bursts.

Integrity Hashes and Disaster Recovery
Detecting an attack is only half the solution; restoring compromised assets is equally critical. Cryptographic hashing algorithms, such as SHA-256, allow systems to generate distinct digital signatures for files. By storing these hashes in a structured manifest alongside snapshot copies, security platforms can verify that backup files remain pristine before initiating a recovery operation. If an attacker tampers with both the live storage and the backup copies, hash mismatches flag the integrity violation immediately, preventing the restoration of corrupted data.

III. METHODOLOGY

A. System Architecture Overview
RansomGuard AI is structured across four primary functional tiers:

1. **Presentation Layer:** A web dashboard providing real-time event tables, interactive charts, threat incident logs, and snapshot management controls.
2. **Application Layer:** A FastAPI server handling REST API requests, CORS policies, static routing, and WebSocket connections.
3. **Services Layer:** Core analytical and defensive engines, including the watchdog monitoring worker, the behavioral risk engine, the snapshot manager, and the attack simulator.
4. **Persistence Layer:** A local SQLite database operating in Write-Ahead Logging (WAL) mode to record events, threats, snapshot manifests, and audit histories.

B. Filesystem Telemetry and Event Collection
The monitoring service relies on Python's watchdog library to observe operations within the user-specified directory. Every file creation, modification, deletion, or movement event triggers an event packet containing:

- Unique event ID and timestamp
- File path, file size, and extension
- Measured Shannon entropy (computed over 64 KB streaming chunks)
- Current process context (where available via psutil)

C. Behavioral Risk Engine
The risk engine synthesizes incoming telemetry to assign a composite risk score ranging from 0 to 100. The evaluation focuses on three primary dimensions:

5. **Entropy Spikes:** Detects sudden increases in byte randomness ($H(X) > 7.5$ bits/byte).
6. **Burst Rate Anomalies:** Tracks write and modification frequencies over sliding 10-second windows. Exceeding baseline operational thresholds signals potential mass encryption.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510120

7. **Extension Mutations:** Identifies suspicious filename changes, such as single-pass conversions to known ransomware extensions (e.g., .locked, .crypto).

Risk scores map to four severity tiers: **LOW** (0-29), **MEDIUM** (30-59), **HIGH** (60-79), and **CRITICAL** (80-100).

D. Snapshot Integrity and Recovery

When a user creates a backup point, the snapshot subsystem copies files from the protected storage into an isolated repository (snapshots/snap_<timestamp>/). Concurrently, it constructs a cryptographic manifest that maps every relative file path to its SHA-256 hash. During disaster recovery, the service recalculates hashes across the snapshot files to verify their integrity before overwriting live storage, effectively blocking corrupted or tampered backups from being deployed.

IV. RESULTS AND EVALUATION

A. Experimental Setup and Reproducibility

The prototype was tested on a local development setup using Python 3.11, FastAPI 0.109, and Watchdog 3.0 on Windows 11. Testing followed a structured execution routine:

1. Establishing a clean baseline in protected_storage/.
2. Initializing background filesystem monitoring via Uvicorn.
3. Generating reference snapshots and cryptographic manifests.
4. Executing simulated attack workloads inside protected_storage/demo/.
5. Verifying dashboard updates, threat logging, and disaster rollback workflows.

B. Simulator Scenarios and Functional Verification

To validate detection and recovery capabilities safely, the built-in simulator was evaluated across three activity profiles:

- **LOW Scenario:** Creates standard test documents (.docx, .pdf) and executes minor modifications to verify baseline event parsing and low risk scoring (0-15).
- **MEDIUM Scenario:** Writes synthetic high-entropy byte payloads to multiple files and renames a subset to .locked. The risk engine successfully flagged this activity as a **MEDIUM/HIGH** risk event (55-75), updating the dashboard in real time.
- **HIGH Scenario:** Triggers rapid burst modifications, high-entropy payload injections, and simulated file deletions in rapid succession. The system elevated the threat level to **CRITICAL** (80+), generated formal threat entries in SQLite, and enabled snapshot recovery options.

C. Evaluation Metrics Framework

To guide future benchmarks on labelled datasets, the evaluation framework defines the following target metrics:

- **True/False Positive Rates:** Accuracy in distinguishing malicious write bursts from benign file extraction or media rendering.
- **Precision, Recall, and F1-Score:** Standard statistical metrics for classifier evaluation across mixed workloads.
- **Detection Latency:** Elapsed time from the first high-entropy burst event to dashboard alert dispatch (observed at < 150 ms during local WebSocket tests).
- **Recovery Reliability:** Verification percentage of SHA-256 integrity checks during full directory rollbacks (100% pass rate on untampered test snapshots).

D. System Strengths and Architectural Challenges

The empirical evaluation highlighted several notable system strengths:

- **Multi-Signal Correlation:** Combining entropy with write frequencies significantly reduced false alarms compared to using entropy alone.
- **Automated Audit Logging:** SQLite persistence ensured every operation remained traceable for post-incident analysis.
- **Seamless Recovery:** SHA-256 manifest checks successfully prevented restored files from deploying if backup bytes were manually altered.

Key technical challenges identified include tuning sliding-window thresholds to accommodate legitimate high-volume disk tasks (such as extracting large zip files) and managing user-space event queue limitations under heavy I/O loads.

V. SYSTEM IMPLEMENTATION AND DEPLOYMENT

A. Backend Architecture and Modular Services

The backend is structured for clear separation of concerns:

- backend/main.py: Entry point setting up FastAPI, CORS headers, static file mounts, and WebSocket lifespans.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510120

- backend/api/: REST API modules divided by domain (monitoring.py, threats.py, snapshots.py, recovery.py, reports.py).
- backend/services/: Core logic implementations (storage_monitor.py, risk_engine.py, snapshot_manager.py, simulator.py).
- backend/database/: SQLite initialization scripts and schema configurations.

B. Frontend Interface and Real-Time Telemetry

The frontend (frontend/RansomGuard-Dashboard.html) is built using a modern glassmorphism aesthetic. It connects to the backend via a WebSocket stream (ws://127.0.0.1:8000/ws/monitor), enabling real-time telemetry updates. The UI features dedicated tabs for live event monitoring, threat inspection, storage health visualization, snapshot creation, audit reporting, and attack simulation controls.

C. Data Persistence and Audit Trail

Application state is maintained within ransomguard.db using SQLite in Write-Ahead Logging (WAL) mode. This setup allows concurrent event writes from background monitoring threads while enabling real-time read queries from the dashboard. Endpoints like /api/reports/export allow administrators to dump full system logs into structured JSON reports for auditing.

D. Deployment Configuration

The system runs as a lightweight local application. Dependencies are managed via standard Python virtual environments (requirements.txt including fastapi, uvicorn, watchdog, psutil, websockets, and pydantic). Interactive OpenAPI docs are automatically generated and accessible at http://127.0.0.1:8000/docs.

VI. DISCUSSION

A. Behavioral Signal Synthesis and Threat Interpretation

RansomGuard AI demonstrates that combining lightweight heuristics provides an effective first line of defense against local storage tampering. High entropy alone is an ambiguous signal; however, when high entropy coincides with rapid write bursts and suspicious file renames, the confidence of a ransomware detection rises significantly. This multi-indicator approach allows the system to remain explainable, giving administrators clear visibility into *why* a specific risk score was assigned.

B. System Limitations and Operational Trade-Offs

Several design trade-offs were made during development:

1. **User-Space Monitoring:** Using watchdog operating in user space means event ingestion relies on OS event notifications. Under massive disk I/O spikes, events may be coalesced by the kernel, causing minor telemetry latency.
2. **Boundary Restrictions:** Surveillance is intentionally restricted to the configured protected path, meaning activity occurring outside this scope is ignored.
3. **Passive Defensive Posture:** RansomGuard AI focuses on surveillance, alerting, and disaster recovery rather than driver-level process termination or kernel-level memory scanning.

C. Snapshot Security and Trust Boundaries

A defense system is only as secure as its backup repository. While RansomGuard AI uses SHA-256 manifests to verify backup file integrity, storing snapshots on the same disk as live data leaves backups vulnerable if an attacker gains full system privileges. For enterprise deployment, snapshot storage must be decoupled from the host machine using write-once-read-many (WORM) storage or remote network locations.

D. Privacy Controls and Ethical Demonstration Design

RansomGuard AI enforces strict safety and privacy controls:

- **Safe Sandbox Testing:** The attack simulator operates strictly within protected_storage/demo/. It uses synthetic data payloads and dummy extension modifications, eliminating the risk of accidental data destruction.
- **Privacy-Preserving Telemetry:** Monitoring is confined to designated paths, preventing unintended tracking of private personal files.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510120

VII. CONCLUSION AND FUTURE WORK

RansomGuard AI presents a functional, transparent defense framework combining real-time filesystem monitoring, behavioral risk evaluation, and hash-verified snapshot recovery. By evaluating entropy spikes, write bursts, and file mutations concurrently, the system delivers explainable risk scoring while mitigating false positives. The integrated sandboxed simulator confirms that end-to-end monitoring, alerting, and point-in-time recovery can be demonstrated safely without live malware.

Future research will focus on:

1. Evaluating the risk engine against standardized, labelled datasets containing real-world ransomware execution traces and diverse benign workloads.
2. Quantifying formal statistical metrics, including precision, recall, F1-score, and ROC-AUC curves.
3. Implementing automated process mitigation strategies (e.g., suspicious process suspension via psutil).
4. Hardening snapshot repositories using remote, immutable backup locations and access-controlled manifest storage.

REFERENCES

[1]. C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.
[2]. National Institute of Standards and Technology, "Secure Hash Standard (SHS)," FIPS PUB 180-4, 2015.
[3]. Python Software Foundation, "Python 3 Documentation," [Online]. Available: https://docs.python.org/3/
[4]. FastAPI Development Team, "FastAPI Documentation," [Online]. Available: https://fastapi.tiangolo.com/
[5]. SQLite Development Team, "SQLite Documentation," [Online]. Available: https://www.sqlite.org/docs.html
[6]. Watchdog Contributors, "Watchdog: Python API library and shell utilities to monitor file system events," [Online]. Available: https://python-watchdog.readthedocs.io/
[7]. OWASP Foundation, "Logging Cheat Sheet," [Online]. Available: https://cheatsheetseries.owasp.org/
[8]. National Institute of Standards and Technology, "Computer Security Incident Handling Guide," NIST SP 800-61 Rev. 2, 2012.

 This work is licensed under a Creative Commons Attribution 4.0 International License