File Integrity Monitor with Unauthorized Operation Prevention and Mass-Tampering Detection

Subasri N.[1], Janarthan J.[2], Kavin Raj V J.[3], Kaushalya Harini S.[4], Vimal Prasanth B.[4]
[1,2,3,4,5]*Department of Computer Science and Engineering, SNS College of Technology, Coimbatore, Tamil Nadu, India*

***Abstract*—Manual, periodic file integrity checks rarely catch tampering, unauthorized edits or ransomware-style mass encryption in time, and they say nothing about who was responsible. This paper describes a Cloud-Based File Integrity Monitor (CFIM) built to close that gap. It checks protected folders against a trusted SHA-256 baseline, logs users in with bcrypt-protected credentials, and refuses any operation their role does not permit. Each change is tied to a user and a process, and a time-window risk engine scores bursts of changes so that mass tampering stands out. Changed files are backed up automatically, and only an administrator can restore them. Security events and trusted hashes go to Amazon DynamoDB, while protected files and their archived versions go to Amazon S3. A scheduled AWS Lambda function re-examines recent events and, when risk is high, e-mails the administrator through Amazon SNS. CFIM ships as a Tkinter desktop application, a Flask web dashboard and a Windows executable, and it drops back to local storage if the cloud cannot be reached. In functional tests, including a simulated 50-file attack, single edits, deletions, new files and bursts of changes were detected and classified correctly, and unauthorized operations were blocked and logged.**

***Index Terms*—Amazon Web Services, cloud security, file integrity monitoring, mass-tampering detection, ransomware detection, role-based access control, SHA-256**

I. INTRODUCTION

Financial records, source code, configuration files and customer data are exactly what attackers go after, and they are just as often damaged by insiders or by ransomware. One quiet change to a protected file can break a business process, cover up fraud or give an intruder a way in. File integrity monitoring (FIM) tackles this by comparing files against a trusted reference and reporting any differences. Security frameworks treat it as a standard control; NIST SP 800-53, for instance, calls for detecting unauthorized changes to information [8].

Classical FIM tools such as Tripwire [1] work from cryptographic digests of files and are normally run on a schedule. Many small organizations and academic labs never get that far and still check files by hand, which is slow and easy to get wrong. Even the lighter tools that exist leave four things undone. They do not check who is operating them. They report unauthorized operations after the fact instead of stopping them. They cannot say which user or process made a change. And they judge each change alone, so a ransomware-style burst looks no different from one ordinary edit.

We present the Cloud-Based File Integrity Monitor (CFIM), which addresses all four gaps in a single system. Its main contributions are: (i) real-time SHA-256 detection of modified, deleted and new files across several protected folders; (ii) bcrypt authentication with role-based authorization that blocks and logs unauthorized operations; (iii) attribution of every event to a user and process; (iv) a behavior-based risk engine that scores bursts of events inside a sliding time window to expose mass tampering; (v) administrator-controlled backup and recovery with S3 version archiving; and (vi) a serverless cloud pipeline built on Amazon DynamoDB, Lambda, EventBridge and SNS that sends e-mail alerts for high-risk activity.

The paper is organized as follows. Section II reviews related work and Section III states the problem and objectives. Section IV describes the architecture, and Section V explains the methodology and algorithms. Implementation details are given in Section VI, evaluation results in Section VII, and limitations and future work in Section VIII. Section IX concludes.

II. RELATED WORK

Kim and Spafford's Tripwire [1] introduced the idea of keeping signed digests of critical files and re-checking them later to spot unauthorized changes. Modern tools generate those digests with the SHA-2 family of hash functions, and SHA-256 is standardized in FIPS 180-4 [2]. Its collision and pre-image resistance is what makes it dependable as proof that a file is unchanged.

Work on ransomware detection has found that behavior says more than any single file event. Scaife et al. [5] watch user data for signs such as bulk content changes and changes of file type, and stop the offending process once a threshold is crossed. Kharaz et al. [6] study file-system activity to tell ransomware apart from harmless software. CFIM follows the same principle on a smaller scale. It does not examine file entropy; instead, it scores how many changes happen in a short window and what kind they are.

Access control and credential protection are standard parts of a secure design. Role-based access control (RBAC) [4] gives permissions to roles rather than to individual users, which keeps administration simple, and the adaptive bcrypt scheme of Provos and Mazières [3] makes stored passwords costly to brute-force. Guidance on log management [7] asks for time-stamped, tamper-evident records held in a central place, and that is why CFIM keeps its events in DynamoDB. For the remaining pieces, Amazon S3 versioning [9], AWS Lambda [10], DynamoDB [11] and SNS [12] supply durable storage, scheduled serverless computing, a scalable event store and notifications, with no servers to maintain. [13]. CFIM extends the monitoring concept to file integrity by detecting unauthorized changes, attributing events to users and processes, and identifying bursts of suspicious activity.

[14] Recent research has also explored blockchain-assisted verification and encryption techniques to protect data integrity and prevent unauthorized modification [13]. In contrast, the proposed CFIM focuses on monitoring file changes, blocking unauthorized operations, attributing events to users and processes, and detecting mass-tampering behavior.

III. PROBLEM STATEMENT AND OBJECTIVES

The problem is the absence of an affordable, automatic and accountable way to detect tampering with protected files, prevent unauthorized operations, identify the responsible user and process, recognize mass tampering and keep a recoverable copy of the data. The objectives are:

1. To detect modified, deleted and added files in real time using SHA-256 digests.
2. To limit the tool and its file operations to authenticated users according to their roles, and to block and log any violation.
3. To attribute each event to a user and process and to classify its risk.
4. To back up changed files and let only an administrator restore them.
5. To keep events, baselines and backups in the cloud, alert the administrator to high-risk activity, and still work offline.

IV. SYSTEM ARCHITECTURE

CFIM uses a layered architecture, shown in Fig. 1. Users reach it through either a Tkinter desktop GUI or a Flask web dashboard, and every request first goes through the authentication and authorization layer. Behind that, the file integrity engine compares each protected folder with its baseline, the attribution module adds user and process details, the risk engine scores the events, and the logger records them. Changed files are backed up locally, and a cloud synchronization layer built on the boto3 library copies events, trusted hashes and files to AWS.

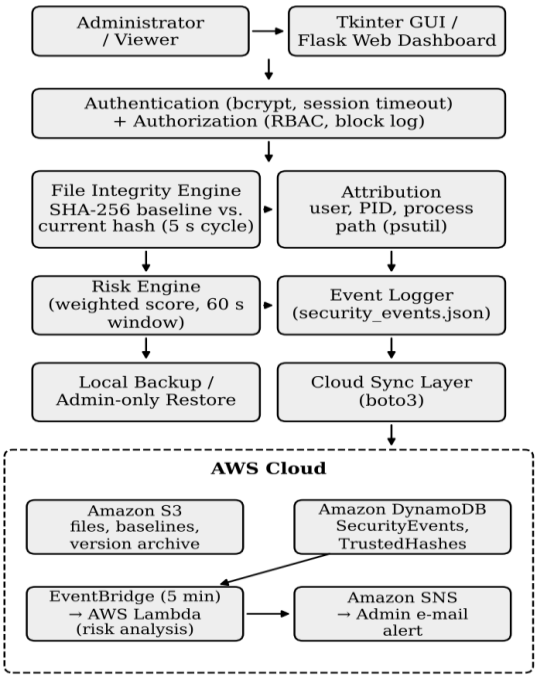

Fig. 1. Layered architecture of the proposed Cloud-Based File Integrity Monitor.

Table I lists the main software modules. When AWS credentials are missing, the cloud modules fall back to a local JSON cache. The local functions therefore keep working, and the cloud pipeline can be demonstrated without an AWS account.

Table I. Main modules of CFIM

Module	Function
integrity.py	SHA-256 hashing, baselines, comparison
monitor.py	5-second monitoring loop, backup trigger
auth.py, authorization.py	bcrypt login, RBAC, block log
attribution.py	User and process details (psutil)
risk_engine.py	Sliding-window mass-tampering score
recovery.py	Backup, manifest, admin restore
s3_storage.py	S3 upload, version archive, restore
dynamodb.py	Event and trusted-hash tables
lambda_function.py, sns_alerts.py	Cloud risk check, e-mail alerts
gui.py, web_dashboard.py	Desktop and browser interfaces

V. METHODOLOGY

A. Baseline Creation and Integrity Checking

For each protected folder, the system reads every file in 4096-byte chunks and computes its SHA-256 digest [2]:

$$H(f) = \text{SHA-256}(f) \quad (1)$$

The names and digests of all files are saved as the trusted baseline B in JSON. In each monitoring cycle (5 s by default) the current digests are compared with B. If a file in B is gone, the system raises a DELETED event. If a file exists in both but its digest differs from B, it raises a MODIFIED event, and if a file is present now but missing from B, it raises a NEW FILE event. Files whose digests match produce nothing. The baseline can also be uploaded to Amazon S3 and to the DynamoDB table TrustedHashes, so a copy of the reference lives outside the monitored machine.

B. Authentication and Authorization

Users log in with a username and password. Passwords are stored only as bcrypt hashes [3], and a session expires after 30 minutes. There are two roles. An Administrator can manage folders and users, create baselines, view and export events, restore files, start and stop monitoring, and edit risk settings. A Viewer has read-only access to events and files. Before any file operation (READ, WRITE, DELETE, CREATE, MODIFY or RENAME) goes ahead, the authorization manager checks it against permission rules kept in a JSON file. If the role does not allow the operation, it is blocked, a block event recording the user, role, folder, file, reason and risk level is written to the block log, and the user sees an access-denied message.

C. User and Process Attribution

Whenever an event is recorded, the attribution module collects the logged-in operating-system user together with the process identifier, parent identifier, name, executable path, command line and start time of the process involved, using the psutil library. We want to be clear about what this means: it identifies the user and process context in which the monitor saw the change. It does not claim to identify an external attacker.

D. Behavior-Based Risk Engine

Each event type carries a weight w: MODIFIED = 1, NEW FILE = 2 and DELETED = 3 points (Table II).

The mass-tampering score at time t is the weighted count of events in the last T seconds, with T = 60 by default:

$$R(t) = \Sigma\, w(e_i), \text{ for } t - T \le \tau_i \le t \qquad (2)$$

Here e_i is an event and τ_i is its time. The score is mapped to a risk level through configurable thresholds stored in risk_config.json. In the cloud analyser the levels are LOW ($R < 5$), MEDIUM ($5 \le R < 10$), HIGH ($10 \le R < 20$) and CRITICAL ($R \ge 20$). Deletions weigh the most, so deleting just seven files within a minute is enough to reach CRITICAL, which fits how destructive malware behaves, whereas one or two isolated edits stay LOW.

Table II. Event weights and risk levels

Event type	Points	Risk level	Score R (60 s)
Modified	1	Low	< 5
New file	2	Medium	5 – 9
Deleted	3	High	10 – 19
		Critical	≥ 20

E. Backup and Administrator-Controlled Recovery

When a new or modified file is detected, the monitor copies it to a protected backup directory and records its source folder in a manifest, so a restore always puts the file back where it belongs. Only an administrator can restore a file, and does so from the Recovery tab. In the cloud, the S3 module keeps files under protected/<folder>/<file> and baselines under baselines/<folder>. json. Before an object is overwritten, the previous copy is archived under versions/<key>/<timestamp>. An archived version can be downloaded into a separate target directory, which keeps the evidence from being overwritten.

F. Cloud Event Pipeline and Alerting

Every security event is written to the DynamoDB table SecurityEvents, whose partition key is event_id and which also stores the timestamp, event type, file name, risk level, score, user and process. An Amazon EventBridge rule triggers the AWS Lambda function every five minutes. The function reads recent events and applies the scoring rules from Section V-D. When the level is HIGH or CRITICAL, it writes an alert record and publishes a message to an Amazon SNS topic, to which the administrator's e-mail address is subscribed. The same Lambda code can also run locally against cached events, and this is how we tested the cloud logic without incurring any cost.

VI. IMPLEMENTATION

CFIM is written in Python 3. It uses hashlib for SHA-256, bcrypt for passwords, psutil for process data and boto3 for AWS access. The desktop interface is built with Tkinter and has tabs for monitored files, security events, risk analysis, authorization, user management, recovery and cloud storage; the last four are visible only to administrators. The web dashboard is a Flask application that shares the same user accounts and offers dashboard, events, alerts, files and cloud-status pages. A build script can package the application as a Windows executable. We recommend a Windows 11 PC with an Intel Core i5 or AMD Ryzen 5 processor, 8 GB of RAM, Python 3.11 or newer and an Internet connection. The cloud services involved (Amazon S3, DynamoDB, Lambda, EventBridge and SNS) all fit within the AWS Free Tier for a project of this size.

VII. RESULTS AND DISCUSSION

A. Functional Evaluation

We tested the system on a protected folder holding 59 files: the 50 demonstration files plus nine others. All tests ran against the project code. The risk-engine tests gave the outcomes in Table III.

Table III. Risk engine test results

Scenario (60 s window)	Score	Level
2 modified files	2	low
5 modified files	5	medium
2 deleted + 1 new + 2 modified	10	medium
20 deleted files	60	critical
Event older than the window	0	low

For the attack simulation, a script created 50 files, set a baseline and then quickly modified all of them. The monitor picked up the modified files, the risk engine raised the activity to CRITICAL, and the alert was written to the alert log. When we ran the Lambda analyser in local-simulation mode on a window of normal activity, it correctly reported 0 points and LOW risk, and it reported HIGH or CRITICAL in the simulated burst tests. Over the course of testing, the security-event log collected 54 records with user and process attribution. The block log recorded 10 unauthorized attempts, such as write and delete requests from a Viewer account, and all were refused.

B. Discussion

These results suggest that per-file digests combined with a time-window score can tell normal edits from destructive bursts without any analysis of file content. Detection time is limited by the polling interval (5 s) for local events and by the EventBridge schedule (5 min) for the cloud alert. The local interface therefore gives immediate feedback, while the cloud path adds an independent, off-machine alert and record. Because events and baselines are also kept in DynamoDB and S3, an attacker who wipes the local log or baseline still cannot touch the cloud copy. Table IV compares CFIM with the manual checking used in current practice.

Table IV. existing practice vs. proposed CFIM

Feature	Existing	CFIM
Detection	Manual / periodic	Automatic, 5 s
User login	None	bcrypt + roles
Unauthorized ops	Not blocked	Blocked, logged
Who changed	Unknown	User + process
Mass tampering	Not detected	Risk score
Recovery	Manual	Admin restore, S3 versions
Alerts	None	SNS e-mail

VIII. LIMITATIONS AND FUTURE WORK

CFIM has several limits. The monitor polls folders instead of using kernel-level file-system notifications, so a file that is changed and then restored between two cycles goes unseen. Attribution describes the monitor's process context and the logged-in user; it cannot trace the true origin of a remote attacker. The default demonstration credentials must be changed before any real use, and the system has not yet been evaluated on very large folders or at enterprise scale. Future work includes event-driven monitoring, recursive folder support, SMS alerts, a mobile application and a richer real-time analytics dashboard. We also plan to try machine-learning anomaly detection on the event stream, with models such as LSTM or Transformers, and to support more than one cloud provider.

IX. CONCLUSION

This paper presented CFIM, a cloud-based file integrity monitor that brings together SHA-256 tamper detection, bcrypt authentication, role-based blocking of unauthorized operations, user and process attribution, a sliding-window risk engine for mass tampering, administrator-only recovery, and a serverless AWS pipeline for storage and e-mail alerts. In our tests, single changes, deletions, new files and 50-file attack bursts were all detected and classified correctly. The system is inexpensive, runs both offline and in the cloud, and offers a practical way for small organizations and academic environments to protect their important files.

ACKNOWLEDGMENT

The authors thank Ms. B. Sangeetha Priya, Assistant Professor, Department of CSE (IoT), College of Technology, Coimbatore, for her guidance throughout this project.

REFERENCES

[1]. Kim, G. H., and Spafford, E. H., “The design and implementation of Tripwire: A file system integrity checker,” in Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, VA, USA, 1994, pp. 18–29.

[2]. National Institute of Standards and Technology, Secure Hash Standard (SHS), FIPS PUB 180-4, Aug. 2015.

[3]. Provos, N., and Mazières, D., “A future-adaptable password scheme,” in Proceedings of the USENIX Annual Technical Conference, FREENIX Track, Monterey, CA, USA, 1999, pp. 81–91.

[4]. Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E., “Role-based access control models,” Computer, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[5]. Scaife, N., Carter, H., Traynor, P., and Butler, K. R. B., “CryptoLock (and drop it): Stopping ransomware attacks on user data,” in Proceedings of the IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, 2016, pp. 303–312.

[6]. Kharaz, A., Arshad, S., Mulliner, C., Robertson, W., and Kirda, E., “UNVEIL: A large-scale, automated approach to detecting ransomware,” in Proceedings of the 25th USENIX Security

Symposium, Austin, TX, USA, 2016, pp. 757–772.
[7]. Kent, K., and Souppaya, M., Guide to Computer Security Log Management, NIST Special Publication 800-92, Sep. 2006.
[8]. Joint Task Force, Security and Privacy Controls for Information Systems and Organizations, NIST Special Publication 800-53, Rev. 5, Sep. 2020.
[9]. Amazon Web Services, "Amazon S3 User Guide: Using versioning in S3 buckets."
[10]. Amazon Web Services, AWS Lambda Developer Guide.
[11]. Amazon Web Services, Amazon DynamoDB Developer Guide.
[12]. Amazon Web Services, Amazon Simple Notification Service Developer Guide.
[13]. Senthil Kumar, R., Balaji, R., Dhivya, M., Deepan Chakravarthy, M., and Sanjay Kumar, G., "HVAC duct cleaning bot and monitoring using BlynkIoT," in Applied Mathematics, Automation and Computing, CRC Press, 2026, doi: 10.1201/9781003770497-5.
[14]. Kaleeswaran, D., Priyadharshini, S., Mubaraali, L., and Rajaram, A., "Quantum adaptive pixel shuffling and blockchain integrated steganographic encryption for securing data in the quantum era," International Journal of Pattern Recognition and Artificial Intelligence, vol. 40, Art. no. 2650017, 2026, doi: 10.1142/S0218001426500175.