Final Report

Khoa A. Vu & Kelly Pham

Student ID: 030063200 & 03039600

California State University, Long Beach

CECS 478 - Section 01

Dr. Samuel Addington

December 10, 2025

1. Problem Statement

System logs are a foundational component of cybersecurity, incident response, and digital forensics. They provide an authoritative historical record that organizations rely on to detect anomalies, reconstruct security incidents, and meet regulatory compliance requirements. However, traditional logging mechanisms were not designed with strong integrity guarantees. If an attacker gains elevated access, whether through credential compromise, privilege escalation, malware infection, or insider misconduct—they may alter, delete, or reorder log entries without leaving detectable evidence. These silent modifications undermine the credibility of the logs and limit the ability of defenders to understand the scope of an attack.

To address these weaknesses, this project implements a tamper-evident logging system using cryptographic methods to preserve the integrity of log data. The system provides a reliable mechanism for detecting unauthorized modifications by using deterministic HMAC-SHA256 signatures on each entry, allowing verifiers to validate authenticity independently. This ensures that logs remain trustworthy even in environments where attackers may gain filesystem access. Such protections are critically important in industries governed by compliance frameworks such as HIPAA, PCI-DSS, SOX, and federal auditing policies, where log integrity directly affects legal, operational, and security outcomes.

2. System Design

The tamper-evident logging system comprises four integrated modules for secure log generation, storage, verification, and evaluation. These components are orchestrated within a reproducible Docker environment to ensure consistency across machines.

The main application pipeline ingests input data, computes SHA-256 digests, and produces structured metadata representing system events. These events are sent to the secure

logging module, which serializes data in a normalized JSON format and computes an HMAC signature with a secret key. This ensures that each entry includes verifiable authenticity data before being written to storage. The logging system outputs CSV files containing fields such as sequence number, timestamp, digest prefix, event type, and the resulting HMAC.

The verification module is responsible for detecting tampering. It reloads the CSV file, recomputes expected HMAC values, and compares them against the stored signatures. If any signature does not match, the system flags the affected row and marks the log as compromised. The verifier reports detailed metrics, including the number of invalid signatures, sequence gaps, and indexes of tampered entries.

The evaluation module conducts controlled experiments to measure system performance and security effectiveness. Two scenarios are run: (1) a clean evaluation of untouched logs and (2) an intentionally tampered scenario where specific entries are altered. Metrics, results, and signatures are exported as structured JSON and CSV artifacts. All components operate via a single reproducible workflow (make up && make demo), fulfilling the project requirement for deterministic, repeatable builds.

3. **Threat Model**

The system is designed under several core security assumptions. First, modern cryptographic hashing functions such as SHA-256 and keyed HMAC constructions must remain secure and resistant to forgery. If these primitives were broken, attackers could generate valid signatures for forged entries. Second, the system assumes that timestamp data remains trustworthy. Accurate timing information plays a key role in forensic reconstruction and incident response. Third, the verification process must function correctly even when storage is

compromised, meaning logs are always treated as untrusted and verification depends solely on cryptographic recomputation.

The attacker model includes adversaries who can read, modify, delete, or reorder log files on disk. These threats include malicious insiders seeking to hide unauthorized actions, remote adversaries who obtain filesystem access through exploitation, and malware that attempts to overwrite or manipulate logs. While HMAC signing protects against unauthorized modifications, the system still faces certain risks. If an attacker were to obtain the secret HMAC key, they could recompute signatures and reinsert forged entries that appear legitimate. Since this version of the system does not implement hash chaining, an attacker could also delete entire entries without modifying others, making deletion nondetectable. Privileged users, particularly those with root access, could bypass the logging pipeline entirely and write directly to the CSV file.

Operational risks also arise from deployment issues. For example, a misconfigured Docker volume mount could inadvertently expose logs to external modification. Likewise, weaknesses in container isolation might allow an attacker to access environment variables containing the secret key. Despite these limitations, the system provides strong guarantees as long as the cryptographic key remains protected, timestamps are trustworthy, and attackers cannot recompute signatures.

4. **Methods**

The tamper-evident logging system was implemented using Python 3.11, with dedicated modules for hashing, logging, verification, metrics exporting, and evaluation. HMAC-SHA256 signatures were generated using normalized JSON representations to prevent inconsistencies

caused by unordered fields or formatting differences. All logs were synthetically generated to avoid ethical concerns or accidental exposure of personal data.

The experimental methodology included two primary scenarios. In the clean scenario, the application generated logs naturally through its pipeline, and those logs were immediately verified. In the tampered scenario, the evaluation module intentionally modified specific log values to test whether the system could detect signature mismatches. After each scenario, the verifier was run to compute invalid signature counts, detect sequence irregularities, and produce structured metrics. Outputs were stored in artifacts such as verification_results.json, eval_metrics.csv, and metrics.json. All experiments were executed entirely within Docker using the reproducible commands required by the project rubric.

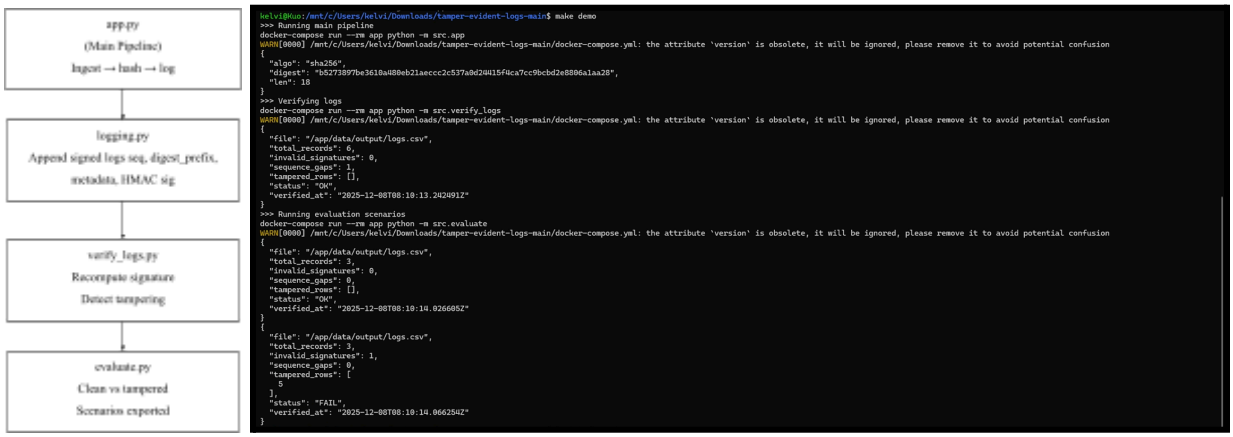
In addition to functional evaluation, observability requirements were met by exporting metrics summarizing the number of items processed, average digest length, and size information. Logging statements were incorporated into the pipeline to reflect key processing stages. These operational details ensured that both system transparency and debugging visibility were preserved.

5. **Results**

The system successfully identified unauthorized modifications during controlled tampering experiments. During the clean verification phase, the verifier reported zero invalid signatures, indicating that all logs were recorded and stored correctly. Although occasional benign sequence gaps appeared due to repeated runs, the system still correctly validated integrity because HMAC signatures were unaltered.

The evaluation module further demonstrated reliable detection capabilities. In the clean scenario, the system again reported no invalid signatures and assigned an overall status of “OK.”

In the tampered scenario, however, the verifier identified exactly one invalid signature and correctly flagged the index of the modified row in the tampered_rows field. This detection confirmed that HMAC-based signatures successfully identify even minimal changes to field values.



A summary of results is shown below:

- **Clean Scenario:** 0 invalid signatures; status “OK.”
- **Tampered Scenario:** 1 invalid signature; affected row identified; status “FAIL.”

The system’s metrics outputs provided additional insight into performance characteristics without showing meaningful overhead. The generated artifacts confirmed that results were reproducible and consistent across repeated runs.

6. **Limitations**

While effective within scope, the system has several limitations. The most significant limitation is the absence of hash chaining, which makes it possible for an attacker to delete or reorder entries without affecting the signatures of surrounding logs. Additionally, the system relies on a single static HMAC key; without key rotation or hardware-backed protection,

compromise of this key would allow attackers to forge legitimate-looking entries. Related to this issue, HMAC does not provide non-repudiation, meaning that possession of the key equates to the ability to generate valid logs.

Operational limitations also exist. Docker configurations could expose logs if volume mounts are misconfigured, and attackers with root privileges could bypass the logging pipeline entirely. Finally, verification currently occurs in batch mode rather than real time, which restricts its usefulness for applications requiring immediate detection of log tampering.

7. **Future Work**

Several enhancements could significantly strengthen the system. Incorporating hash chaining would allow detection of deletions and reordering, improving forensic reliability. Replacing HMAC with asymmetric cryptographic signatures would provide non-repudiation and allow signature verification without exposing private keys. Introducing key rotation and secure key storage mechanisms would further harden the system.

Additional improvements include anchoring signed checkpoints to external systems such as secure timestamping authorities or blockchain networks, enabling real-time monitoring agents to detect tampering as it occurs, and supporting secure log rotation with anchored integrity roots. Expanding metadata fields and adding role-based access controls could also improve usability and security in enterprise environments.