

# Log Sentinel: A Lightweight Behavioural Threat-Hunting Tool for Web Attack Detection

Dhanraj Gangnaik<sup>1</sup>, Bala Krishna Sanneboyain<sup>2</sup>, Aditya Kale<sup>3</sup>, Dr. Sandeep Kulkarni<sup>4</sup>

<sup>1,2,3</sup> School of Engineering, Ajeenkya D Y Patil University, Pune

<sup>4</sup> Department of Computer Science, Ajeenkya D Y Patil University, Pune, Maharashtra, India

\*\*\*

**Abstract** - Web services face continuous threats from attacks such as SQL injection, cross-site scripting, path traversal, and command injection [18]. Traditional approaches create a difficult operational trade-off: simple signature-based systems lack the sophistication to detect evasive variants, while heavyweight SIEM platforms demand significant infrastructure and sustained operational overhead [19]. This paper introduces Log Sentinel, a lightweight behavioural threat-hunting framework designed for real-time web attack detection on access logs. The system bridges this gap by combining deterministic rule matching with per-IP behavioural analytics to identify both explicit attack indicators and suspicious traffic patterns. It continuously monitors Nginx-style logs, applies transparent URL normalization, and processes events through a staged detection pipeline. Key architectural contributions include a hybrid detection layer that fuses signature-based rules with temporal behavioural features such as endpoint diversity, error-rate shifts, and user-agent churn, an incident-correlation mechanism that groups related alerts into campaign-level views to reduce analyst fatigue, and a user-friendly Portal interface built with FastAPI and Next.js. Integration with observability infrastructure via Prometheus and Grafana enables operational monitoring. Evaluation results from two experiments demonstrate the system's effectiveness: a controlled benchmark with 280 records achieved zero false positives and sub-millisecond latency, while a live containerized test with 125 generated requests achieved precision of 0.9394, recall of 0.4429, and F1-score of 0.6019. The framework exhibits a precision-first profile suitable for low-noise alert triage. Log Sentinel provides small teams with an interpretable, maintainable solution for continuous threat hunting without complex infrastructure.

**Key Words:** Threat Hunting, Web Log Monitoring, Behavioural Detection, Access Log Analysis, Intrusion Detection, Attack Correlation, Anomaly Detection, Risk Scoring, Security Operations

## 1. INTRODUCTION

Public-facing applications operate in a continuously scanned environment. Commodity reconnaissance tools enumerate routes at scale, while patient attackers probe payload variants over longer windows to avoid obvious patterns [17]. Both behaviours leave measurable traces in access logs: repeated requests to rare paths, bursts of client errors, and structured request rhythms that differ from typical user activity. In practice, however, many teams still use these logs primarily for post-incident forensics rather than continuous detection [20].

This gap is largely operational. Signature detectors are computationally efficient and interpretable, yet brittle against encoding tricks, token fragmentation, and syntax mutation. At the opposite end, SIEM-centric platforms offer broad correlation but require infrastructure, tuning, and analyst capacity that smaller environments rarely have in abundance [12], [15]. The result is a recurring tooling gap: systems are either too narrow to be dependable or too heavy to maintain.

Behavioural monitoring offers a practical middle path. Rather than relying only on payload text in a single request, it evaluates source behaviour over time: endpoint diversity, error-rate shifts, burst intensity, and user-agent churn. These temporal signatures often remain visible even when payload strings are obfuscated. Crucially, this approach can remain lightweight and interpretable without introducing complex model-training pipelines [19].

Log Sentinel is built around this design choice [17]. It ingests Nginx-style logs as a stream, applies transparent normalization and detection stages, and emits prioritized alerts with explicit evidence. This paper contributes: (1) a modular hybrid detector that fuses signatures with per-IP behavioural state, (2) an incident-correlation layer that converts alert bursts into campaign-level context, and (3) a reproducible evaluation workflow that reports both accuracy and runtime characteristics. The remaining sections present scope, related work, methodology, implementation details, and results.

## 2. PROBLEM STATEMENT

This work targets web-exposed services that already generate structured access logs (for example, Nginx or Apache formats). The assumed adversary can send arbitrary HTTP requests over the network but does not control the host. Defensively, the operator seeks near-real-time visibility using only existing log telemetry, without introducing SIEM-scale infrastructure. The key attacker behaviours in scope are summarized below.

### 2.1 Adversary Capabilities

- **Reconnaissance and Enumeration:** Attackers may perform directory and endpoint discovery using automated scanners, producing high volumes of requests to non-existent resources (404 spikes) and probing common administrative paths (e.g., */admin*, */login*, */.git*). They may also attempt parameter discovery by varying query keys and paths systematically [17].
- **Payload Injection:** Attackers attempt to inject SQLi, XSS, command injection, and traversal payloads into query strings, URL paths, and headers. Payloads may be encoded (URL encoding, mixed case, comment insertion) to evade simple pattern checks [18].
- **Credential Attacks:** Adversaries may execute credential stuffing and brute-force attempts against authentication endpoints, observable via repeated POSTs to login forms, abnormal session behaviour, or high error rates.
- **Evasion:** Attackers may rotate user agents, distribute scanning across multiple IPs, introduce delays to mimic benign traffic, or use decoy requests to reduce detection confidence [17].

### 2.2 Defender Assumptions and Scope

The defender can read access logs and run the Log Sentinel engine on the same host or on a nearby monitoring node. The design does not require deep packet inspection, host-based EDR instrumentation, or TLS interception. Detection is driven by request metadata (IP, method, path, query, status code, user agent, timestamp) and temporal relationships across events [15].

### 2.3 Out-of-Scope

The framework is intended for detection and triage, not attack prevention. Capabilities outside scope include inspection of encrypted payload content beyond what is logged, insider misuse with privileged access, and application-specific semantic checks such as business-logic abuse detection. The system also does not attempt strong attribution; the objective is actionable, low-overhead alerting for resource-constrained environments.

## 3. LITERATURE REVIEW

Intrusion detection research divides broadly into signature matching, statistical anomaly detection, and hybrid approaches. Rule-based systems have remained the most widely deployed form because their output is easy to interpret and they perform reliably against documented threats. The core weakness is brittleness: a payload that evades a pattern through encoding, comment injection, or case variation goes undetected, and keeping rule files current demands ongoing effort [11], [12], [14].

Anomaly detection tackles this differently by modelling normal traffic and flagging deviations. Machine-learning classifiers have been applied extensively to this problem and often show strong results in controlled evaluations. In practice, building and maintaining such a system is harder: labelled training data is scarce, normal traffic shifts over time requiring regular retraining, and the model's decision is difficult to explain to an analyst who needs to respond quickly [13]. Compute and memory requirements also tend to exceed what small deployments can provide.

Log-based detection occupies a useful middle ground because access logs already exist as a by-product of running any web service. Tools that scan log lines for known bad paths, brute-force patterns, or injection strings can be deployed without changes to network topology or application code. The practical problem is alert noise: benign crawlers, health checkers, and misconfigured clients trigger the same signatures as real attackers, so purely rule-driven log scanners can generate more noise than signal unless correlation is applied [15].

Log Sentinel sits in the hybrid category. It pairs explicit pattern rules with per-IP behavioural counters so that attacks which evade pattern matching may still trigger a behavioural alert, and vice versa. The emphasis on log-only analysis, zero external dependencies, and a sub-millisecond event processing budget distinguishes it from tools that require dedicated agents, persistent databases, or cloud-based threat feeds.

### 3.1 Research Gap and Design Rationale

Prior work establishes that signature systems are interpretable and operationally trusted, while anomaly systems improve variant detection but often add training, maintenance, and explainability burden [13], [14]. In practical small-team deployments, this creates a recurring gap: available tools are either too narrow (single-request pattern scanners) or too heavy (full SIEM plus model ops). Existing literature discusses this trade-off, but fewer implementations demonstrate a reproducible middle path with explicit analyst-facing evidence and resource usage low enough for edge deployment on the monitored host itself.

This project targets that specific gap through four design choices. First, log parsing and normalization are deterministic and transparent, avoiding hidden model state. Second, rule and behavioural signals are fused at alert time rather than replaced, preserving explainability while broadening coverage. Third, incident correlation is treated as a first-class component to reduce analyst fatigue in scanner-heavy conditions. Fourth, evaluation includes both a controlled labelled harness and a live containerized run, because strong metrics on synthetic-only traffic can overestimate real-world performance.

### 3.2 Research Questions

The implementation and experiments are structured around four research questions. **RQ1:** Can a log-only hybrid detector sustain real-time performance on commodity hardware? **RQ2:** Does combining behavioural counters with rules improve operational usefulness (lower alert noise with acceptable recall) over a rule-only baseline? **RQ3:** Which attack categories remain weakest under current pattern coverage, and are those gaps linked to known obfuscation families? **RQ4:** Does simple incident correlation materially improve triage readability in scanning scenarios compared with raw alert streams?

The results reported in later sections answer RQ1 directly via latency, throughput, and memory measurements; answer RQ3 through per-category recall; partially answer RQ2 through the ablation template and conservative precision profile; and answer RQ4 qualitatively through campaign-level incident grouping. These questions provide a clear path for iterative improvement without changing the system's lightweight deployment objective.

## 4. METHODOLOGY

The framework is implemented as a staged pipeline with explicit boundaries between ingestion/parsing, detection, correlation, and presentation. This separation keeps components independently maintainable: rule updates do not require changes to correlation logic, and visualization layers can be replaced without altering the detector core. The architecture now has four principal components: the Log Sentinel Engine, the Dashboard/Integration Service, the Portal Workspace (API + UI), and the Evaluation Harness. Figure 1 provides an end-to-end view of the layering and data flow across these components.

### 4.1 Log Sentinel Engine

The engine runs continuously, tails a configured Nginx access log, and processes new lines as they arrive. Each event is parsed into a normalized record containing source IP, HTTP method, request path/query, response status, user agent, and timestamp. Before rule matching, path and query fields undergo two rounds of URL decoding and canonical cleanup. This normalization step is critical because many attacks use layered encoding to evade naive substring checks; for example, a single decode can leave %2527 unresolved, whereas a second decode recovers the apostrophe relevant to SQLi rules.

The normalized event then passes through three stages. Stage one applies compiled regex rule sets for SQLi, XSS, traversal, scanner, and command-injection indicators. Stage two updates per-IP behavioural windows and evaluates deviations such as endpoint bursts, sustained 4xx spikes, and request rates above local baseline. Stage three correlates temporally related alerts from the same source into incident records, producing campaign-level context instead of isolated event fragments.

### 4.2 Dashboard Service

A lightweight threaded HTTP service exposes detector output through a REST API. The endpoints `/api/alerts`, `/api/stats`, and `/api/incidents` provide evidence-backed alert and campaign views. Additional helper endpoints support dashboard composition and observability: `/api/prometheus/query-range` and `/api/grafana/embed-preview`. This layer enables both the classic dashboard UI and the newer Portal API/UI stack to consume the same enriched detection data.

### 4.3 Evaluation Harness

To support reproducible measurement, the project includes a self-contained benchmark harness. A dataset generator emits labelled traffic across five attack classes plus benign examples using a fixed seed, producing the same 280 records on each run. The evaluator then replays records through the live detector, compares predictions with labels, and reports accuracy, precision, recall, F1-score, per-event latency, throughput, and peak memory. Metrics persisted to a single `report.json` artifact for version-to-version comparison after rule or threshold updates.

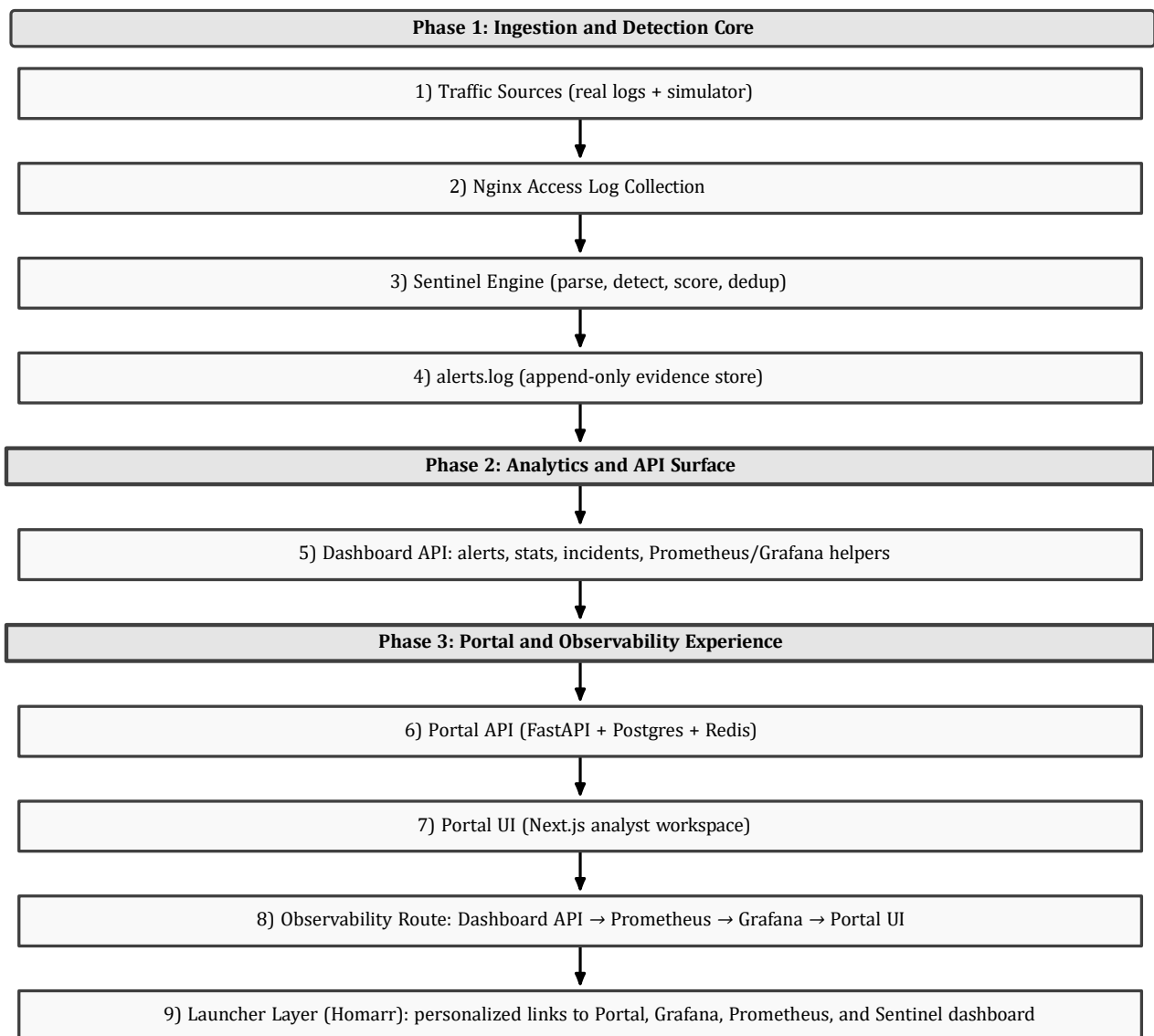


Fig. 1. Layered system architecture showing core ingestion/detection, analytics APIs, and portal-observability delivery.

#### 4.4 Detection Methodology

Detection combines two complementary signal families. Pattern matching captures explicit payload indicators such as SQL tautologies, script fragments, and traversal tokens. Behavioural analysis captures source-level deviations over time that align with scanning or probing behaviour. Either signal alone is incomplete: rules may miss encoded or fragmented variants, while behavioural thresholds can fire during legitimate bursts. Fusing both signals improves practical coverage while preserving interpretability and avoiding heavy model dependency. The full numbered processing sequence is shown in Figure 2.

#### 4.5 Log Parsing and Normalization

Log lines are parsed with a regular expression matching the Nginx combined log format [2]. The extractor pulls out IP, timestamp, method, request path, status code, response size, referrer, and user agent. Normalization applies two passes of URL decoding to the path and query string, lowercases them where appropriate, and strips redundant separators. Double decoding is important: some attacks encode characters twice so that a single-pass decoder still outputs an encoded string that bypasses substring checks. URI encoding conventions follow RFC 3986 [16].

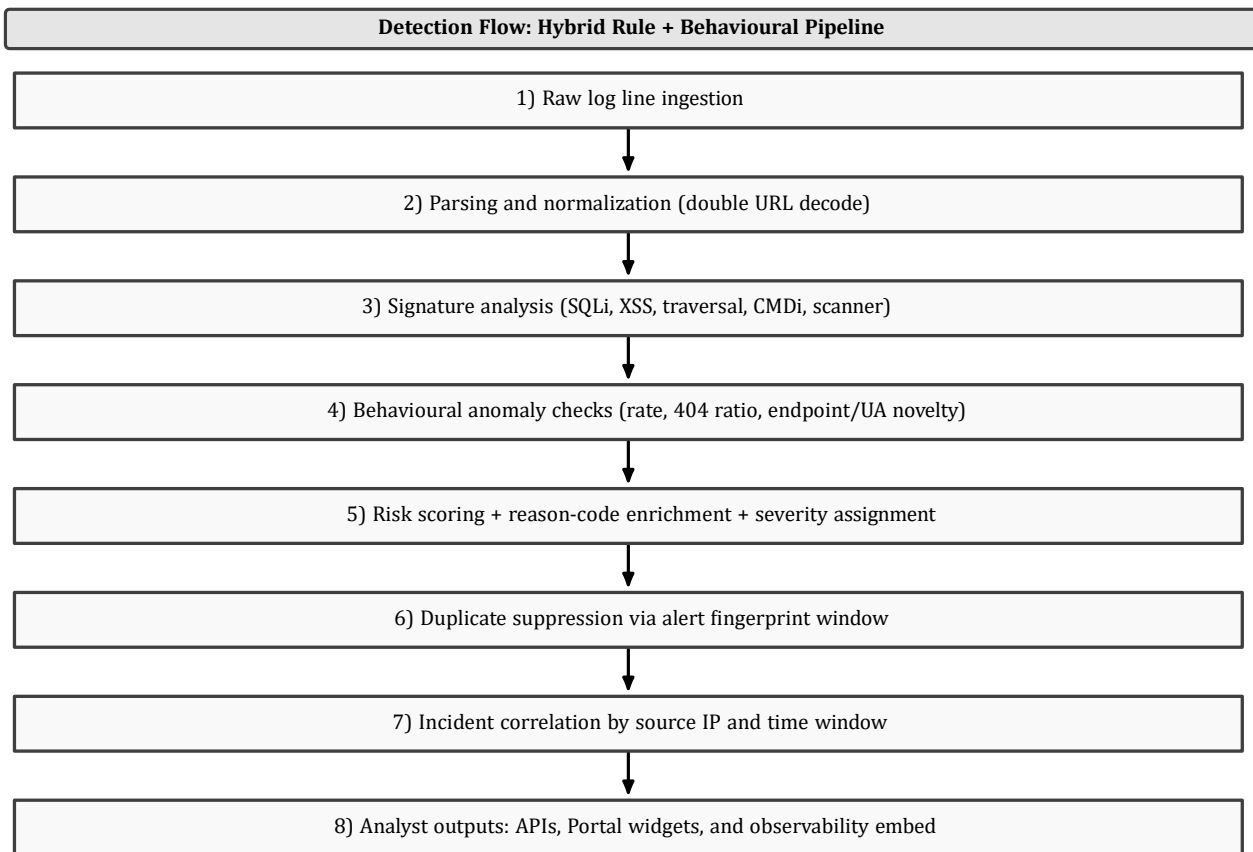


Fig. 2. Numbered detection pipeline from normalized ingestion through scoring, deduplication, and incident-level delivery.

#### 4.6 Rule-Based Detection

Pattern files for each attack category live in rules/ and are loaded at startup as compiled, case-insensitive regular expressions. The five files cover SQLi, XSS, path traversal, command injection, and scanner user agents. Matching runs against both the raw and the decoded-normalized path, so payloads using encoded special characters are still caught. When a pattern matches, the engine records the matched expression alongside the decoded request path, giving analysts direct evidence rather than a bare alert flag. Table 1 summarizes the category-level rule inventory used in the evaluation.

### 4.7 Behavioural Anomaly Detection

For each source IP, the engine maintains a small set of counters inside a sliding time window: requests per minute, count of distinct endpoints accessed, fraction of 4xx responses, and number of distinct user agents seen. When a metric crosses a configurable threshold relative to that IP’s own rolling baseline, a behavioural alert is raised. Global thresholds are deliberately avoided because a busy crawler and a quiet end user have very different normal traffic volumes; a per-IP baseline comparison avoids the false positives that global limits produce [14]. The new-endpoint and new-user-agent detectors additionally require a minimum history count before they fire, preventing spurious alerts on an IP’s very first few requests.

**TABLE -1: DETECTION RULE CATEGORIES AND PATTERN SUMMARY**

Attack Category	Rule File	Patterns	Example Tokens
SQL Injection	sqli_patterns	33	UNION SELECT, SELECT.*, FROM
XSS	xss_patterns	35	<script>, onerror=, onload=
Path Traversal	traversal_patterns	25	../, %2e%2e%2f
Cmd Injection	cmdi_patterns	38	;ls, whoami, \$.*
Sec Scanner	scanner_patterns	24	nikto, sqlmap

### 4.8 Risk Scoring and Correlation

Each alert is assigned a numeric risk score before emission. The base score depends on attack category, with command injection and SQLi rated higher than scanner detection. Context adjustments then add points: a doubly encoded path scores higher than a plaintext one; a spike in the requesting IP’s recent volume amplifies the score; a high anomaly ratio adds a further increment. The formula used is:

$$RiskScore = w_1P + w_2A + w_3F \tag{1}$$

**TABLE -2: RISK SCORING COMPONENT WEIGHTS**

Factor	Weight	Description
Pattern confidence ( <i>P</i> )	0.50	Base severity by attack category (High: 1.0, Medium: 0.6)
Anomaly intensity ( <i>A</i> )	0.30	Behavioural signal strength: rate spike, endpoint burst, error ratio
Frequency amplification ( <i>F</i> )	0.20	Repeat alert count within deduplication window

where *P* represents pattern-match confidence, *A* represents anomaly intensity, and *F* captures frequency amplification (e.g., repeated triggers in a short interval). Alerts from the same source IP that fall within a configurable time window are then merged into a single incident record, replacing a flood of individual events with a campaign-level summary that shows attack type, peak score, and duration. To ensure stable behaviour across deployments, each component is normalized to a bounded interval before weighting:  $P \in [0, 1]$ ,  $A \in [0, 1]$ , and  $F \in [0, 1]$ . The alert score is therefore constrained to  $RiskScore \in [0, 1]$  when  $(w_1 + w_2 + w_3) = 1$ . The normalized form prevents one detector from dominating purely due to scale and makes threshold tuning interpretable: for example, raising the alert threshold from 0.60 to 0.70 has consistent meaning independent of raw feature ranges. The baseline component weights are reported in Table 2.

## 4.9 Implementation

The entire system is written in Python 3, chosen for broad availability and ease of modifying detection logic without a compilation step. The sentinel process uses a tail-like loop to ingest new log lines; each parsed record passes through detection and is appended to *data/alerts.log* in a structured block format that the dashboard server reads without needing a database backend. The analyst workspace extends this Python core with a FastAPI-based Portal API and a Next.js Portal UI for configurable dashboard workflows and embedded visualizations [4], [5]. Pattern files are plain text with one regex per line, so adding a new indicator means editing a text file rather than changing source code. Behavioural counters use bounded deque structures per IP so that memory footprint stays predictable as the number of tracked sources grows. A duplicate-suppression window deduplicates identical alert fingerprints within a configurable interval, preventing a burst of identical requests from flooding the alert log.

The repository now ships a multi-profile Docker Compose stack. The core profile runs sentinel + dashboard, Portal adds Postgres, Redis, Portal API, and Portal UI, observability enables Prometheus/Grafana dashboards, homarr enables the launcher layer, demo runs a synthetic target and traffic generator, and full launches all components together. The run-stack wrappers for PowerShell, CMD, and Bash standardize profile lifecycle actions (up/down/logs/ps) and reduce operator error during demonstrations. This packaging keeps experiments reproducible while supporting both lightweight and full-stack analyst workflows. The complete source is publicly available [1], [6]–[10].

## 4.10 Computational Complexity and Resource Model

Per-event cost is intentionally bounded to support continuous operation. Let  $m$  denote the number of compiled regex patterns across all rule files,  $\ell$  the normalized request-string length, and  $k$  the number of metrics in the per-IP behavioural state. In the expected case, one event update has time cost  $O(m\ell + k)$ : regex matching dominates, while counter updates are constant-time deque operations per metric. Incident correlation uses a bounded time window and hashable fingerprints, so insertions and duplicate checks remain amortized  $O(1)$  per alert.

Memory usage scales with active source IPs in the current window rather than with total historical traffic. If  $n$  is active IP count and each IP stores bounded deques of maximum length  $b$  for  $k$  metrics, state memory is  $O(nkb)$  with fixed constants configured by policy. This bound explains why the measured peak memory remains low in the benchmark despite sustained event flow. Operationally, this makes capacity planning straightforward: increasing window size or per-IP history depth trades memory for smoother behavioural baselines.

**TABLE -3: OVERALL EVALUATION METRICS**

Metric	Value
Accuracy	0.7357
Precision	1.0000
Recall	0.5375
F1-score	0.6992
Average Latency (ms)	0.283
P95 Latency (ms)	0.666
Throughput (records/s)	3219.84
Peak Memory (KB)	271.17

## 4.11 Experimental Protocol

All controlled experiments are executed with deterministic data generation, single-process sentinel execution, and metric capture in one report artifact. To reduce run-to-run variability, the benchmark is performed on an idle host with background workloads minimized, and wall-clock measurements are taken per record from parse entry to alert decision completion. Throughput is derived as total processed records divided by elapsed processing time, while latency is reported as both mean and P95 to capture tail behaviour.

For the live containerized experiment, labels are generated by the traffic simulator and post-hoc matched to alerts by timestamp proximity and request-path similarity. Because no global request identifier is present in the Nginx log line, this

mapping introduces a controlled uncertainty channel that is discussed explicitly in the validity section. Reporting both controlled and live metrics is therefore essential: controlled runs isolate detector behaviour under known labels, while live runs reflect pipeline realities including timing jitter, container scheduling variance, and path canonicalization side effects.

**TABLE -4: PER-TYPE RECALL (CURRENT EVALUATION RUN)**

Attack Type	Recall
SQL Injection	0.2250
Cross-Site Scripting	0.3429
Path Traversal	1.0000
Command Injection	0.3333
Security Scanner	1.0000

## 5. Results

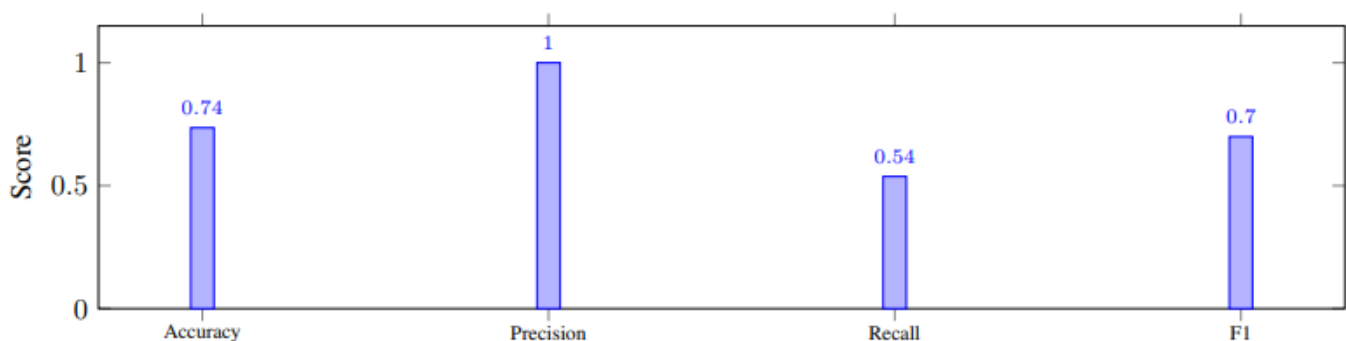
This section reports quantitative outcomes from two evaluation settings: (1) a controlled harness based on a 280-record labelled dataset (160 attack, 120 benign), and (2) a separate live containerized run. Controlled metrics are taken directly from *log-sentinel/evaluation/report.json* generated on the current codebase.

### 5.1 Reproducibility Setup

The dataset is generated deterministically using a fixed random seed, so the exact same 280 records appear on every run. Attack records span SQL injection, XSS, path traversal, command injection, and scanner user-agent patterns across five attack categories. Benign records use realistic-looking paths such as */home*, */products?id=2*, and */blog/1*. Running *python log-sentinel/evaluation/generate\_dataset.py* from the repository root writes the records to *dataset.json*. Running *python log-sentinel/evaluation/evaluate.py* then processes each line through the live sentinel engine, measures per-record latency and memory, and writes all metrics to *log-sentinel/evaluation/report.json*. This setup enables reproducible comparison after rule or threshold updates.

### 5.2 Detection Accuracy

Table 3 shows the aggregate results from *log-sentinel/evaluation/report.json*. Precision is 1.0, meaning no benign record was flagged as an attack. Recall is 0.5375, indicating that just over half of labelled attacks were detected. This precision-heavy profile is operationally desirable for reducing alert fatigue, while the recall gap highlights where signature and behavioural coverage should be expanded. Figure 3 visualizes the same aggregate metrics. Category-level performance is tabulated in Table 4 and plotted in Figure 4 to highlight recall gaps.



**Fig. 3.** Overall classification metrics from the current evaluation run.

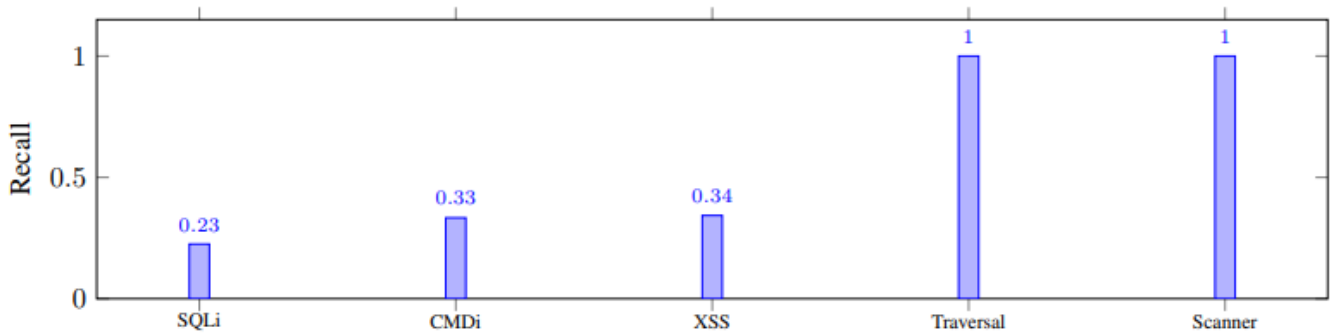


Fig. 4. Per-attack-type recall, highlighting coverage gaps in SQLi/XSS/CMDi.

TABLE -5: LIVE CONTAINERIZED EXPERIMENT METRICS (125 REQUESTS)

Metric	Value
TP	31
FP	2
FN	39
TN	53
Precision	0.9394
Recall	0.4429
F1-score	0.6019
Accuracy	0.6720
Total Labels	125
Total Alerts	33

### 5.3 Live Containerized Experiment

To validate behaviour outside the synthetic harness, we ran a live Docker-based testbed with attack-traffic replay and Log Sentinel tailing Nginx-style access logs using container orchestration for reproducibility [3]. These live-run results are reported from a separate experiment artifact and are intentionally presented alongside (not inside) the controlled *evaluation/report.json* benchmark file. The traffic generator emitted 125 labelled requests (70 attack, 55 benign) across SQLi, XSS, path traversal, command injection, and scanner-style probes. Alerts were matched to labels using timestamp and path similarity because a shared request identifier is not present in the log pipeline.

The resulting confusion-matrix counts were TP=31, FP=2, FN=39, and TN=53, yielding precision 0.9394, recall 0.4429, F1-score 0.6019, and accuracy 0.6720. Relative to the controlled harness, precision remains strong while recall drops, suggesting that the current configuration is intentionally conservative under mixed live traffic and may miss some evasive variants. Detailed live metrics are listed in Table 5, while direct controlled-vs-live comparisons are shown in Table 6 and Figure 5.

TABLE -6: CONTROLLED HARNESS VS LIVE EXPERIMENT

Metric	Controlled (280)	Live (125)
Accuracy	0.7357	0.6720
Precision	1.0000	0.9394
Recall	0.5375	0.4429
F1-score	0.6992	0.6019
False Positives	0	2

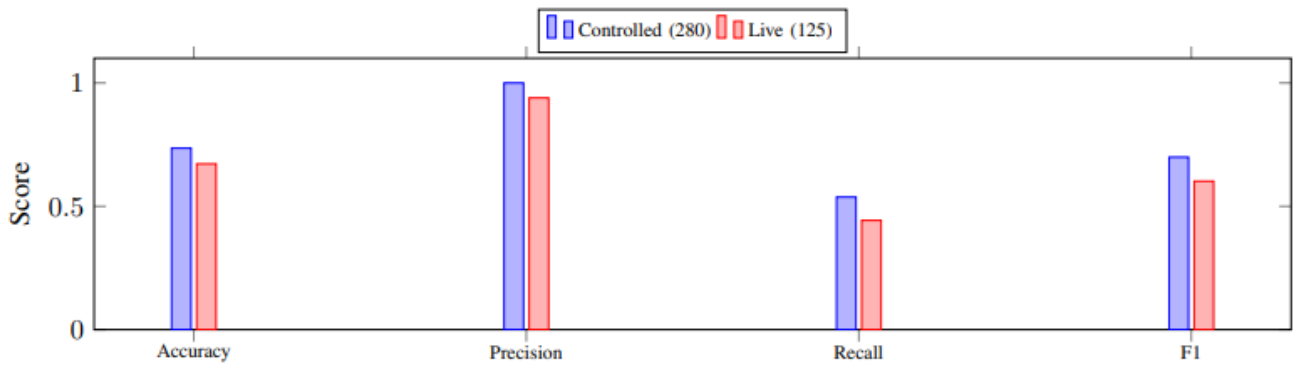


Fig. 5. Metric comparison between controlled harness and live containerized evaluation.

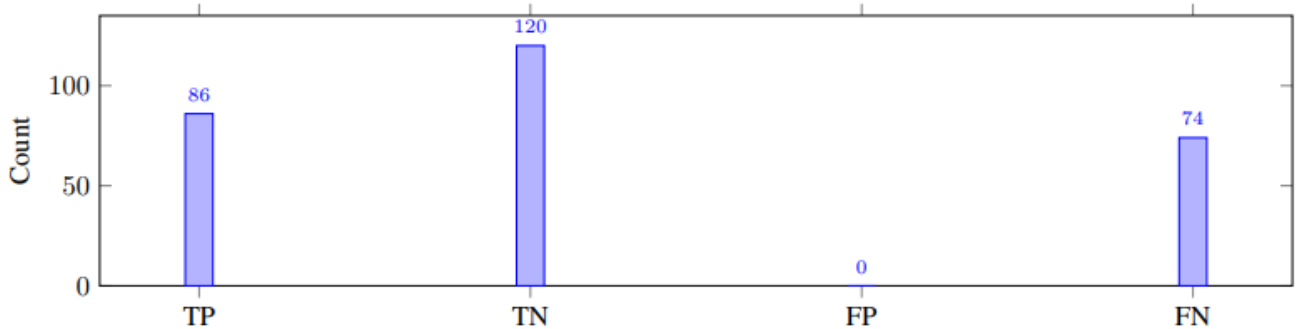


Fig. 6. Confusion matrix counts for the controlled 280-record dataset.

### 5.4 Latency and Throughput

Average per-record latency was 0.283 ms, with a 95th-percentile of 0.666 ms, both within the range required for real-time log tailing. Throughput reached approximately 3219.84 records per second in the benchmark. Peak memory during the run was about 271.17 KB, reflecting the bounded deque design: per-IP state is capped so that tracking a large number of sources does not cause unbounded memory growth. Figure 6 reports the controlled-run confusion matrix counts underlying these aggregate outcomes.

### 5.5 Incident Correlation Effectiveness

In a live deployment the window-based grouper collects alerts from the same IP within a 10-minute window and merges them into a single incident record. For scanning-heavy traffic, where one source can generate dozens of 404 alerts in quick succession, this compression is meaningful: instead of fifty individual alerts, an analyst sees one incident labelled with type breakdown, peak score, and duration. Moving analyst focus from isolated alerts to campaign-level context improves triage efficiency and reduces the chance that slow-moving activity is obscured by repetitive low-level noise.

TABLE -7: ABLATION TEMPLATE FOR BEHAVIOURAL CONTRIBUTION

Configuration	Precision	Recall	F1-score
Rule-only (behavior off)	-	-	-
Rule+ BEHAVIOURAL (baseline)	1.0000	0.5375	0.6992

## 5.6 Ablation Plan (Rule-Only vs Rule+ Behavioural)

To measure the behavioural component's contribution in isolation, the same dataset can be evaluated with `enable_behavioral_detection_false` in the config, and then with the default enabled setting. The table below shows the template; the rule-only column can be populated in the next evaluation cycle after the same dataset is replayed under both configurations. The planned reporting format appears in Table 7.

## 5.7 Security Analysis

Three design choices drive detection usefulness. First, every alert carries explicit evidence: the matched regex, a decoded copy of the flagged path, and behavioural context such as the IP's recent request rate. An analyst reading the alert output can immediately see why it fired without consulting raw log files. Second, behavioural signals are harder to evade than static patterns. An attacker who encodes every payload and cycles through user agents still produces a detectable spike in endpoint diversity or error rate because those signals come from traffic volume and distribution, not payload content [14]. Third, the small memory and CPU footprint means the tool can run on the same host as the service being monitored, with no separate forwarding pipeline required.

Known limitations are worth stating directly. Threshold values that work on a lightly loaded home lab server may generate false positives on a high-traffic production service. The current defaults favour precision over recall, which is the right trade-off for avoiding alert fatigue but means some attacks go undetected. Log-based monitoring also has a hard ceiling: POST body content that Nginx logs at default settings is not captured, so injection attacks hidden in request bodies are invisible to this tool. A patient adversary who spreads requests well below the rate spike threshold may also avoid behavioural detection, though the rule layer would still catch explicit payloads.

Despite these constraints, Log Sentinel fits a genuine operational need. It complements rather than replaces a full SIEM and is useful as a lightweight first-alert layer for environments where heavier tooling is not an option.

## 5.8 Discussion

Per-category results reveal a clear pattern. Path traversal and scanner traffic reach perfect recall because those classes usually contain stable lexical or structural indicators. By contrast, SQLi (0.225), XSS (0.343), and command injection (0.333) remain harder to capture under conservative rule settings, particularly when payloads are obfuscated.

The weakest area is SQLi coverage. The current pipeline correctly handles common URL encodings through double decoding (for example `%27`), but evasive forms such as comment-fragmented keywords (`un/**/ion`), mixed token separators, and unusual encoding combinations reduce recall. Similar limitations apply to XSS and command injection families where attack strings can be distributed across syntax elements that are individually benign.

To mitigate this, the rule corpus has already been expanded and currently contains 33 SQLi, 35 XSS, 25 traversal, 38 command-injection, and 24 scanner patterns. These additions emphasize observed evasion behaviours, including comment-based mutations, broader event-handler vectors, shell substitution forms, and command chains associated with privilege checks or exfiltration. Although the current benchmark reflects a comparatively basic controlled set, the expanded rules are intended to improve resilience under more adversarial payload variation.

The primary engineering trade-off is unchanged: improving recall without sacrificing the low false-positive profile. In practical operations, persistent false alarms quickly erode analyst trust and reduce response quality. For this reason, rule broadening should be validated incrementally against benign traffic partitions before full deployment, with the ablation process used as a guardrail.

Overall, the results support a precision-first deployment posture with explicit tuning controls (thresholds, whitelists, and rule sets) that operators can adapt to local traffic characteristics. The next step is not architectural redesign, but targeted refinement of variant coverage and baseline adaptation while keeping the detector lightweight and explainable.

## 5.9 Threats to Validity

- **Internal validity:** Controlled labels are generated from synthetic templates; if those templates under-represent evasive payload families, recall may appear stronger or weaker than in production. The label-to-alert matching used in the live run is based on time and path similarity rather than a unique request identifier, which can introduce occasional assignment error in both false-positive and false-negative counts.
- **External validity:** The dataset and live testbed focus on web-access log telemetry from Nginx-like formats. Results may not transfer directly to environments with very different logging schemas, reverse-proxy behaviour, or traffic profiles (for example, API-heavy backends with high legitimate error rates and bursty mobile-client retries).

- **Construct validity:** Precision, recall, and F1 quantify classification quality but do not fully capture analyst effort. Incident correlation quality, alert readability, and time-to-triage are only partially represented by these metrics. A user study or SOC-style tabletop exercise would better measure whether campaign-level grouping materially improves investigation outcomes.
- **Conclusion validity:** The reported metrics are based on specific rule versions and threshold settings. Small configuration changes can shift the precision-recall balance, especially in SQLi and XSS categories. For this reason, metric comparisons across versions should always include the exact rule snapshot and configuration hash used for evaluation.

## 5.10 Operational Deployment Guidance

For production-like usage, threshold tuning should follow a staged rollout. Begin with an observation phase and export alert counts per category, per-IP, and per-time-of-day to establish local baselines. Continue with a progressive hardening phase by tightening thresholds gradually while monitoring false-positive complaints from operators. Only after stable behaviour is observed should high-risk actions (such as automated blocking through external tooling) be enabled.

Whitelisting should remain minimal and evidence based. Permanent suppression of entire subnets can hide compromised internal hosts, so narrow path-aware rules are preferable. Periodic rule review is also necessary: scanner signatures and payload conventions evolve, and stale patterns can silently reduce recall. In resource-constrained environments, the recommended architecture is to keep Log Sentinel as an edge detector and forward only enriched incidents upstream, reducing storage and analyst load while preserving high-value context.

## 6. FUTURE WORK

The current implementation establishes a practical baseline for lightweight hybrid detection, but several technical extensions can improve recall and operational robustness while preserving the low-overhead design goal.

- **Adaptive baselines without heavy ML:** A key extension is replacing fixed behavioural thresholds with light statistical adaptation per source and per endpoint class. Exponential moving statistics and robust quantile bounds can capture traffic drift while remaining interpretable for analysts. This approach keeps the system explainable and avoids the maintenance burden of full model retraining pipelines.
- **Expanded normalization for evasions:** Current double URL decoding handles many encoded payloads but not all obfuscation families. Future preprocessing can add canonicalization for mixed encodings, repeated delimiter folding, and safer normalization of Unicode-like confusable. Improved canonicalization should be paired with strict regression testing on benign traffic to avoid raising false-positive rates.
- **Richer correlation semantics:** Incident grouping currently uses temporal and source-IP proximity. A stronger campaign view can include path-similarity graphs, attack-sequence motifs (for example recon to injection progression), and confidence-aware merge rules. These additions would improve analyst context by linking related low-severity signals into a higher-confidence narrative.
- **Evaluation breadth and portability:** Beyond the current controlled and live testbed runs, broader validation should include multiple log formats, higher-traffic profiles, and API-heavy workloads where benign error patterns are common. This would test external validity and help tune defaults for diverse deployment environments without changing core architecture.
- **Interoperability with existing SOC tooling:** A lightweight export layer for STIX-compatible events or SIEM-friendly JSON schemas can make Log Sentinel a drop-in edge signal generator for larger security programs. The emphasis should remain on forwarding enriched incidents, not raw event floods, so that upstream systems receive high-value context with manageable ingestion cost.
- **Portal-native editing and launcher automation:** The current stack supports dashboard registration through APIs and launcher-level navigation. Future releases can add direct in-portal drag-and-drop widget editing, template-based tenant presets, and automated Homarr tile synchronization so non-technical users can manage personalized dashboards without API calls.

## 7. ETHICAL AND RESPONSIBLE USE

Although Log Sentinel is designed for defensive monitoring, operational use must respect legal, ethical, and organizational boundaries. Access-log analysis can contain personal data elements such as IP addresses, user-agent strings, and requested paths that may indirectly reveal user behaviour. Deployments should therefore follow data-minimization principles: collect only required fields, retain records for the shortest practical duration, and apply role-based access controls to alert and incident views.

## 7.1 Privacy and Data Governance

Defensive visibility should be balanced with privacy protection. Where policy permits, identifying fields can be partially masked in analyst-facing dashboards while preserving forensic utility through reversible controls available only to authorized responders. Organizations should document retention policy, investigation access boundaries, and incident audit trails so that monitoring activities remain accountable and reviewable.

## 7.2 Fairness and False-Positive

Harm Even low false-positive rates can create operational harm if alerts repeatedly target shared NAT gateways, university proxies, or regional mobile carriers. Such concentration can bias response behaviour against benign users behind noisy infrastructure. To reduce this risk, response actions should be proportional to confidence level and evidence quality; high-impact controls (for example, hard blocking) should require corroborating indicators beyond a single trigger.

## 7.3 Safe Response Design

The system is intentionally positioned as a detection and triage aid rather than an autonomous enforcement engine. Automated response integrations should default to reversible actions such as temporary rate limits, challenge flows, or staged quarantine policies instead of permanent bans. This design reduces the chance of prolonged service disruption caused by misclassification while preserving rapid containment capability when alerts are strongly corroborated.

## 7.4 Security of the Monitoring Pipeline

The monitoring stack itself can become a target. Alert logs, rule files, and dashboard APIs should be treated as security-sensitive assets: integrity checks, least-privilege file permissions, and authenticated API access are essential. Tampering with rule files or suppression lists can silently blind detection, so configuration changes should be versioned, peer-reviewed, and auditable.

## 7.5 Academic and Defensive Use Statement

This work is presented to improve defensive monitoring capability in educational and resource-constrained environments. The framework is not intended to support offensive activity, unauthorized surveillance, or policy-violating monitoring. Responsible use requires explicit authorization from system owners, transparent operational policy, and compliance with applicable law and institutional ethics requirements.

## 8. CONCLUSION

This work demonstrates that practical threat hunting can be performed directly from web access logs without heavyweight infrastructure. Log Sentinel combines interpretable rule signals with per-IP behavioural context and incident correlation, producing alerts that remain explainable for analysts while meeting real-time processing constraints.

On the controlled 280-record dataset, the system achieved zero false positives, perfect recall for path traversal and scanner categories, precision of 1.0, recall of 0.5375, average latency of 0.283 ms, and peak memory of 271.17 KB. In the live containerized experiment (125 requests), it achieved precision 0.9394, recall 0.4429, F1-score 0.6019, and accuracy 0.6720. Together, these results show stable high precision with moderate recall under mixed traffic.

The implementation remains fully reproducible through the open-source codebase, Docker Compose deployment, and built-in evaluation harness. Current rule files already include expanded SQLi, XSS, and command-injection coverage, and the remaining engineering focus is improved detection of heavily obfuscated payload variants while preserving low false-positive operation.

From an operational perspective, the most relevant outcome is consistent signal quality under constrained deployment conditions: alerts remain explainable, resource usage stays bounded, and the platform scales from a minimal sentinel-only deployment to a richer Portal + observability workflow without architectural changes.

## REFERENCES

- [1] Bala Krishna Sanneboyain, "Behavioral Threat Hunting (Log Sentinel) Repository," GitHub, 2026. [Online]. Available: <https://github.com/BalaKrishnaS7/Behavioral-Threat-Hunting>.
- [2] NGINX, Inc., "Module ngx http log module," 2026. [Online]. Available: [https://nginx.org/en/docs/http/ngx\\_http\\_log\\_module.html](https://nginx.org/en/docs/http/ngx_http_log_module.html).
- [3] Docker, Inc., "Docker documentation," 2026. [Online]. Available: <https://docs.docker.com>.
- [4] FastAPI, "FastAPI documentation," 2026. [Online]. Available: <https://fastapi.tiangolo.com/>.
- [5] Vercel, "Next.js documentation," 2026. [Online]. Available: <https://nextjs.org/docs>.
- [6] PostgreSQL Global Development Group, "PostgreSQL documentation," 2026. [Online]. Available: <https://www.postgresql.org/docs/>.
- [7] Redis Ltd., "Redis documentation," 2026. [Online]. Available: <https://redis.io/docs/>.
- [8] Prometheus Authors, "Prometheus documentation," 2026. [Online]. Available: <https://prometheus.io/docs/>.
- [9] Grafana Labs, "Grafana documentation," 2026. [Online]. Available: <https://grafana.com/docs/grafana/latest/>.
- [10] Homarr, "Homarr documentation," 2026. [Online]. Available: <https://homarr.dev/docs/>.
- [11] D. E. Denning, "An intrusion-detection model," IEEE Trans. Softw. Eng., vol. SE-13, no. 2, pp. 222–232, Feb. 1987.
- [12] M. Roesch, "Snort – Lightweight intrusion detection for networks," in Proc. 13th USENIX Conf. Syst. Admin. (LISA), Seattle, WA, USA, 1999, pp. 229–238.
- [13] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Comput. Surv., vol. 41, no. 3, Art. no. 15, Jul. 2009.
- [14] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," ACM Trans. Inf. Syst. Secur., vol. 3, no. 3, pp. 186–205, Aug. 2000.
- [15] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (IDPS)," NIST Special Publication 800-94, Feb. 2007. [Online]. Available: <https://csrc.nist.gov/publications/detail/sp/800-94/final>.
- [16] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic syntax," RFC 3986, Jan. 2005. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc3986>.
- [17] MITRE, "ATT&CK Framework," 2025. [Online]. Available: <https://attack.mitre.org>.
- [18] OWASP Foundation, "OWASP Top 10:2021 web application security risks," 2021. [Online]. Available: <https://owasp.org/Top10/2021/>.
- [19] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in Proc. IEEE Symp. Security and Privacy, Oakland, CA, USA, May 2010, pp. 305–316.
- [20] Verizon, "2023 Data Breach Investigations Report," Verizon Business, 2023. [Online]. Available: <https://www.verizon.com/business/resources/reports/dbir/>.