



Machine Learning-Based Vulnerability Detection and Real-Time Threat Mitigation in TCP/IP-Based Web Applications

¹Anjali Sahu, ²Dr. Ravindra Kumar Tiwari

¹Research Scholar, ²Professor,

¹Department of Computer Application,

¹LNCT University Bhopal (M.P.), India

Abstract: The increasing reliance on TCP/IP-based web applications for financial transactions, healthcare, cloud computing, and enterprise communication has expanded the cyberattack surface. Conventional security mechanisms, including signature-based intrusion detection, static filtering, and web application firewalls, remain essential but may struggle with novel attacks, evolving traffic patterns, protocol anomalies, and legitimate-looking malicious requests.

This paper proposes an integrated machine-learning framework for vulnerability detection and real-time threat mitigation. It combines network-flow analysis, HTTP characteristics, application security signals, supervised classification, anomaly detection, risk scoring, and policy-controlled responses. Random Forest and gradient-boosting classifiers provide computationally efficient baselines, while optional deep learning supports sequential traffic analysis. An adaptive response engine selects proportionate actions, including alerts, rate limiting, session challenges, temporary blocking, and human escalation.

The evaluation methodology incorporates time-aware and group-aware data partitioning, class-imbalance handling, threshold calibration, and metrics such as precision, recall, F1-score, false-positive rate, detection latency, and mitigation effectiveness. Public intrusion-detection datasets can support preliminary network-level evaluation; dedicated web-application testing remains necessary to assess vulnerability-specific performance. The proposed framework jointly considers detection accuracy, operational latency, false-positive costs, and safe mitigation. Without experimental results, its effectiveness remains a testable research hypothesis rather than a demonstrated improvement.

IndexTerms - Machine learning; web application security; TCP/IP; vulnerability detection; intrusion detection; threat mitigation; Random Forest; gradient boosting; anomaly detection; web application firewall; cybersecurity.

1. Introduction

1.1 Background

Web applications are increasingly exposed to distributed and automated attacks because they operate across public networks, integrate multiple services, and process user-controlled data. E-commerce platforms, online banking systems, cloud dashboards, educational portals, and enterprise APIs depend on reliable communication between clients, application servers, databases, and external services. The Transmission Control Protocol/Internet Protocol (TCP/IP) suite provides the communication foundation for many such systems, while HTTP and HTTPS enable application-level exchanges.

Security failures can arise at different layers. At the network and transport layers, adversaries may exploit abnormal connection patterns, resource exhaustion, scanning activity, or distributed denial-of-service (DDoS) behaviour. At the application layer, weaknesses may include injection, broken access control, authentication failures, insecure configuration, and server-side request forgery. The OWASP Top 10 provides a recognized classification of major web application security risks; its 2025 edition should be consulted when mapping a new study to current industry priorities (OWASP Foundation, 2025). These risks cannot all be identified reliably from TCP flags, packet sizes, or flow durations alone.

Traditional security controls typically combine signatures, access-control policies, rate limits, protocol validation, and known malicious indicators. Such mechanisms are valuable because they are interpretable and can block established attack patterns with predictable behaviour. However, attackers can alter payload encoding, distribute activity across source addresses, exploit legitimate application functionality, and modify request rates to avoid simple detection rules. Static rules may also generate false positives when legitimate traffic resembles malicious activity.

Machine learning (ML) offers an additional analytical capability. Supervised models learn statistical patterns from labelled traffic, while unsupervised or semi-supervised methods identify deviations from established behaviour. Ensemble algorithms can combine heterogeneous features, and sequential models can represent temporal patterns in network events. Nevertheless, ML is not inherently secure or universally superior to conventional detection. Its performance depends on training-data quality, class balance, feature design, deployment conditions, and the representativeness of evaluation data.

The central challenge is therefore not merely to classify traffic as malicious or benign. A deployable system must identify suspicious activity, estimate risk, select a suitable mitigation action, and assess whether that action reduces harm without unnecessarily disrupting legitimate users.

1.2 Research problem

Many intrusion-detection studies evaluate models using offline datasets and classification metrics. Such evaluations can demonstrate discrimination between labelled classes, but they do not necessarily establish that a model detects application vulnerabilities, operates within a real-time latency budget, or mitigates attacks safely in a live environment.

Three distinctions are especially important.

First, **vulnerability detection and attack detection are different tasks**. A vulnerability is a weakness in code, configuration, or system design. An attack detector identifies activity that may exploit such a weakness. A vulnerable endpoint can remain unexploited during traffic observation, while a malicious request may be blocked before exploitation occurs. Consequently, network traffic classification cannot independently establish that a software vulnerability exists.

Second, **network-level and application-level evidence are complementary rather than interchangeable**. Flow features can reveal unusual connection behaviour, whereas HTTP methods, routes, response codes, authentication context, and application logs may reveal suspicious application interactions. Encrypted traffic limits direct payload visibility unless inspection occurs at an authorized TLS termination point or an application instrumentation layer.

Third, **detection does not guarantee successful mitigation**. A model may classify an event correctly but still fail operationally if the response arrives too late, targets the wrong session, blocks shared infrastructure, or creates an availability problem.

This research addresses these challenges through a layered framework that combines ML-based classification, anomaly scoring, application context, risk-aware policy selection, and feedback-based evaluation.

1.3 Research objectives

The objectives are:

1. To design a layered framework for detecting suspicious network and web-application activity using complementary TCP/IP, HTTP, and application telemetry.
2. To develop a model-selection and feature-engineering methodology suitable for supervised classification and anomaly detection.
3. To formulate a risk-scoring mechanism that combines model confidence, behavioural deviation, asset criticality, and evidence quality.
4. To define real-time mitigation procedures that balance attack containment with service availability and false-positive costs.
5. To establish a reproducible experimental protocol for evaluating detection quality, computational overhead, latency, and mitigation effectiveness.
6. To identify the limitations of public datasets and the additional evidence needed to substantiate vulnerability-specific and deployment-level claims.

1.4 Research questions

RQ1: How can network-flow features and application-level telemetry be combined to improve the contextual interpretation of suspicious web requests?

RQ2: How do Random Forest, gradient boosting, and an optional sequential model compare under time-aware and group-aware evaluation?

RQ3: How does risk-calibrated mitigation affect false-positive costs, detection-to-response latency, and service availability?

RQ4: To what extent do models trained on public network datasets generalize to contemporary web applications and previously unseen attack patterns?

1.5 Research contributions

The proposed contributions are fourfold. First, the framework separates network anomaly detection, application attack detection, and vulnerability assessment into distinct but interoperable components. Second, it introduces a risk-aware response policy that avoids treating every positive classification as justification for immediate blocking. Third, it defines an evaluation methodology that includes temporal generalization, operational latency, calibration, and mitigation outcomes. Fourth, it proposes an audit and feedback mechanism for detecting model drift, investigating false positives, and safely updating security policies.

These contributions are architectural and methodological. Their effectiveness must be established through implementation and empirical testing before they can be interpreted as demonstrated performance gains.

2. Literature Review

2.1 Conventional intrusion detection and prevention

Intrusion detection systems monitor events to identify possible malicious activity, while intrusion prevention systems additionally attempt to stop or contain detected incidents. NIST Special Publication 800-94 describes major intrusion-detection and prevention approaches and provides guidance on deployment, configuration, monitoring, and maintenance (Scarfone & Mell, 2007). Signature-based systems are effective when an event matches a known rule or indicator. Their limitations include dependence on maintained signatures and reduced sensitivity to genuinely novel patterns.

Anomaly-based systems establish a baseline of normal activity and identify deviations. Such deviations may indicate attacks, but they can also result from software updates, traffic surges, unusual business activity, or changes in client behaviour. Hybrid systems combine signatures, anomaly detection, contextual rules, and analyst review. The proposed framework follows this hybrid principle rather than replacing existing security controls with a single ML model.

2.2 Machine learning for network intrusion detection

ML-based intrusion detection has received extensive research attention. Sommer and Paxson (2010) explain why applying ML to operational network intrusion detection presents difficulties beyond standard classification, including the problem of generalizing from controlled training data to changing real-world environments. Buczak and Guven (2016) review data-mining and ML methods for intrusion detection, highlighting the importance of features, datasets, and evaluation design.

Random Forest is useful as a baseline because it can model nonlinear feature interactions, accommodate heterogeneous tabular features, and provide feature-importance estimates. Gradient-boosting algorithms can offer strong predictive performance on structured data, although their suitability depends on tuning, data quality, and computational constraints. Support Vector Machines can be useful for selected datasets but may become expensive as the number of observations increases. Neural networks can learn complex representations but typically require careful regularization, sufficient training data, and stronger controls against overfitting.

A survey or benchmark that reports high classification accuracy should be interpreted in the context of its dataset and split strategy. If benign traffic dominates, accuracy may remain high despite poor detection of rare attacks. If training and testing records contain

highly similar flows from the same capture session, a model may exploit dataset-specific patterns rather than learn transferable indicators.

2.3 Deep learning and temporal analysis

Deep-learning methods can model sequences of events, relationships among features, and complex representations. Recurrent neural networks, temporal convolutional networks, and transformer-based architectures are possible candidates when traffic is represented as ordered events or windows. However, their additional complexity does not guarantee better detection, especially where the dataset is small, labels are noisy, or a simpler classifier already captures the relevant patterns.

For real-time deployment, the computational cost of feature extraction and inference must be assessed alongside predictive performance. A complex model that marginally improves recall but exceeds the response-latency budget may be unsuitable for inline enforcement.

2.4 Web application security and vulnerability assessment

The OWASP Top 10 identifies major classes of web application risk, including broken access control, injection, authentication failures, security misconfiguration, and server-side request forgery. The classification is useful for organizing application-security testing, but it is not a complete vulnerability taxonomy or a replacement for a formal testing methodology.

2.5 Public datasets and generalization

The CIC-IDS2017 dataset introduced by Sharafaldin, Lashkari, and Ghorbani (2018) contains labelled network flows representing benign activity and multiple attack categories. It is a useful resource for benchmarking network intrusion-detection approaches. However, a flow-based dataset does not automatically provide reliable labels for source-code vulnerabilities, application authorization defects, or the exploitability of individual web endpoints.

Datasets collected in controlled environments may also exhibit temporal, host-specific, and scenario-specific patterns. Gharib et al. (2016) discuss the importance of rigorous evaluation of intrusion-detection datasets, while subsequent studies have raised concerns about dataset artefacts and the comparability of results. These concerns motivate evaluation using grouped splits, temporal holdouts, and independent application tests.

2.6 Research gap

The literature supports ML as a potentially useful component of intrusion detection, but four practical gaps remain relevant to the present study:

- The distinction between network attack classification and actual vulnerability identification is not always made explicit.
- Offline classification metrics do not adequately represent the cost and timing of operational mitigation.
- Evaluation may be overly optimistic when correlated records appear in both training and testing sets.
- Automated blocking can introduce its own risks, including false-positive denial of service, collateral blocking, and instability during traffic spikes.

3. Proposed Research Framework

3.1 System architecture

The proposed system consists of seven components: telemetry collection, feature extraction, preprocessing, ML inference, contextual risk assessment, mitigation orchestration, and feedback monitoring.

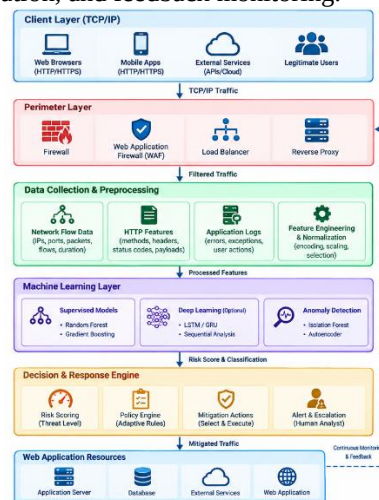


Figure 1. Proposed architecture of the ML-based web application security framework

Source: Authors' proposed conceptual architecture; to be converted into a publication-quality vector figure before submission.

The collection layer receives flow records from network sensors, HTTP metadata from an authorized reverse proxy or web application firewall, and relevant application logs. Data collection should follow applicable privacy requirements and minimize unnecessary retention of personal or sensitive information.

The feature extraction layer generates features over request, session, connection, and time-window levels. The preprocessing layer validates schemas, handles missing values, applies training-derived transformations, and rejects malformed or out-of-range inputs. The detection layer combines supervised classifiers with anomaly scoring and established security rules. The risk assessment layer integrates model output with application context. Finally, the response engine applies preapproved policies, records decisions, and observes the effect of each action.

3.2 Threat model

The framework considers external and internal sources of suspicious activity, including:

1. Network scanning and abnormal connection establishment.
2. Brute-force or credential-stuffing behaviour.
3. Request floods and resource-exhaustion attempts.
4. Injection-like HTTP request patterns.
5. Suspicious authentication and session behaviour.
6. Unexpected request rates, methods, routes, or response patterns.

7. Anomalous access to sensitive application resources.
8. Previously unseen combinations of network and application indicators.

The system does not assume that all attacks are externally originated or that IP addresses uniquely identify actors. Shared proxies, carrier-grade NAT, VPNs, and distributed botnets complicate attribution. IP-based blocking must therefore be time-limited and context-sensitive.

The framework assumes that telemetry sources are sufficiently trustworthy, clocks are synchronized within a known tolerance, and response actions are enforced by a protected control plane. If logs are incomplete, features are manipulated, or the control plane is compromised, detection confidence and mitigation reliability may degrade.

3.3 Data sources and labels

The proposed evaluation uses two complementary data categories.

Network-level data: Public intrusion-detection datasets such as CIC-IDS2017 can be used to assess discrimination between benign and labelled malicious flows. Such experiments should retain original source metadata for grouping and should not imply that flow labels identify software vulnerabilities.

Application-level data: An isolated, authorized test environment should contain deliberately vulnerable demonstration applications, benign usage scenarios, and controlled attack simulations. Each test case should be labelled according to the verified behaviour, such as successful or unsuccessful exploitation, detected attack attempt, unauthorized access, or service degradation. A vulnerability finding should be confirmed using an appropriate security-testing procedure rather than inferred solely from the model score.

Labels should distinguish at least four conditions: benign activity, suspicious but unconfirmed activity, confirmed malicious activity, and confirmed vulnerability evidence. Ambiguous observations should not be silently forced into a binary label.

3.4 Feature engineering

Feature engineering is central to the proposed framework. Features should be extracted from information that the deployment environment is authorized to inspect.

Table 1. Proposed feature groups and their security relevance

Feature group	Example features	Primary analytical purpose
TCP/IP flow	Flow duration, packet counts, byte counts, packet rates, connection failures	Network anomaly and resource-exhaustion detection
TCP behaviour	SYN/ACK/RST counts, retransmission indicators where available, connection establishment failures	Suspicious connection patterns
Temporal	Inter-arrival time, request rate, burstiness, session frequency	Brute-force, scanning and flooding indicators
HTTP metadata	Method, status code, route template, request size, response size, content type	Application request and response anomalies
Session and identity	Authentication failures, session changes, privilege transitions, request sequences	Authentication and access-control risk signals
Application context	Endpoint sensitivity, operation type, authorization outcome, error class	Contextual risk assessment
Historical behaviour	Baseline deviation, repeated failures, endpoint-specific rate changes	Behavioural anomaly detection
Operational context	Service load, maintenance window, current incident state	Safer mitigation and false-positive reduction

Raw URLs should be normalized into route templates when possible, for example, `/users/{id}` rather than storing every identifier. Sensitive query values, authorization headers, passwords, and session tokens should not be used as model inputs unless a narrowly justified and appropriately protected use case requires them. In most deployments, these values should be excluded or irreversibly transformed.

HTTP payload inspection should occur only where authorized. For HTTPS, a network sensor outside the TLS termination point generally cannot observe encrypted application payloads. The design should therefore combine network metadata with application-side telemetry rather than claim universal visibility into encrypted requests.

3.5 Data preprocessing

The preprocessing pipeline should be fitted on training data only. Numerical features can be imputed using training-set statistics, while categorical features can be encoded using a fixed training-derived vocabulary. Unknown categories should be handled consistently during inference.

Highly skewed variables, including flow duration and request counts, may require transformations such as $\log(1+x)$. Extreme values should be investigated before removal because they may represent genuine attacks. Feature selection should be performed within the training process to avoid leakage from the test set.

Class imbalance should be addressed using class-weighted loss, carefully selected resampling methods, or threshold adjustment. If synthetic oversampling is used, it must be applied only to training data. Oversampling correlated flows before grouping can produce misleadingly optimistic results.

3.6 Supervised detection model

The primary model is a Random Forest classifier, with gradient boosting as a competing candidate. The models estimate the probability of an event belonging to a malicious class:

$$\hat{p}(y=1|\mathbf{x}) = f_{\theta}(\mathbf{x})$$

where $\mathbf{x}$ denotes the feature vector, $(y=1)$ represents the target malicious class, and θ denotes learned model parameters.

For a Random Forest with (M) trees, the estimated class probability can be expressed as:

$$\hat{p}(y=1|\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M p_m(y=1|\mathbf{x})$$

where (p_m) is the class-probability estimate from tree (m) . Probability calibration should be evaluated separately because ensemble outputs are not guaranteed to be well calibrated.

The supervised model is intended to learn patterns from labelled examples. Its output should not be interpreted as a direct probability that a software vulnerability exists. A high score indicates similarity to the model's learned malicious patterns under its training and calibration assumptions.

3.7 Anomaly detection

A complementary anomaly detector identifies activity that differs from an established baseline. An Isolation Forest or another suitable unsupervised method can generate an anomaly score ($a(x)$). The score should be normalized or calibrated using held-out benign and representative attack data before it is combined with supervised predictions.

Anomaly detection is especially relevant when labels are incomplete or when behaviour changes. However, unusual activity is not necessarily malicious. New product launches, flash sales, software deployments, and legitimate usage surges can produce substantial deviations. Consequently, anomaly scores should initially trigger monitoring or proportionate challenges rather than automatic permanent blocking.

3.8 Contextual risk scoring

Let $p(x)$ be the calibrated supervised-model score, $a(x)$ the normalized anomaly score, $c(x)$ an application-criticality factor, and $e(x)$ the strength of corroborating evidence. A normalized risk score can be defined as:

$$R(x) = 100 \left[w_p p(x) + w_a a(x) + w_c c(x) + w_e e(x) \right]$$

subject to:

$$w_p + w_a + w_c + w_e = 1, \quad w_i \geq 0$$

Each component must be scaled to $([0,1])$. The weights should be selected using training and validation data, operational risk analysis, and cost-sensitive evaluation. Asset criticality and evidence strength should be defined through documented policies, not improvised after observing test results.

The score supports prioritization; it is not a universal probability of compromise. Calibration and uncertainty must be considered before enforcement decisions.

Table 2. Proposed risk levels and response policy

Risk score	Illustrative level	Initial response	Escalation condition
0–24	Low	Log and monitor	Repeated deviation or corroborating evidence
25–49	Moderate	Alert and apply low-impact rate controls where justified	Continued suspicious behaviour
50–74	High	Rate limit, step-up verification, or temporary session restriction	Confirmed malicious evidence or persistent high risk
75–100	Critical	Apply a preapproved short-duration block or targeted containment	Human review and incident-response escalation

3.9 Real-time mitigation engine

The mitigation engine maps risk and evidence to a permitted response. It should apply the least disruptive action likely to reduce the identified threat.

- **Alerting:** Notify security monitoring systems when activity is suspicious but insufficiently verified.
- **Rate limiting:** Restrict request frequency using endpoint, session, or authenticated identity context.
- **Step-up verification:** Require additional authentication or a challenge for risky operations.
- **Temporary blocking:** Apply short-lived, narrowly scoped blocks when evidence and risk justify intervention.
- **Application-level containment:** Restrict access to a sensitive operation or session when authorization or behavioural indicators support the action.
- **Human escalation:** Require analyst approval for high-impact actions, ambiguous evidence, and actions that could affect many legitimate users.

Permanent blocking should not be the default outcome of a single model prediction. Shared IP addresses, reverse proxies, and cloud gateways may serve many unrelated users. Mitigation should include expiration times, rollback mechanisms, allowlisted critical services, audit logs, and emergency disablement.

3.10 Detection-to-response workflow



Figure 2. Operational workflow for real-time detection and mitigation

The processing path should be asynchronous where possible. Feature extraction and model inference may run inline only when the latency budget permits; expensive investigations should be moved to an event-processing or analyst workflow. A fail-safe design must define behaviour when the model service becomes unavailable. For most general-purpose web applications, loss of the ML service should not automatically block all users; existing deterministic controls should remain active.

3.11 Model maintenance and governance

The model lifecycle should include version control, dataset provenance, validation reports, deployment approvals, monitoring, rollback, and periodic retraining. Changes in endpoint mix, user behaviour, protocol distribution, or attack patterns can cause distribution drift. Drift should trigger investigation, not automatic retraining on unverified traffic labels.

Training data should be screened for poisoning, duplicated records, incorrect labels, and accidental inclusion of identifiers that allow the model to memorize capture-specific patterns. Analyst feedback must be reviewed before becoming a training label. Security and operations teams should retain authority to suspend automated actions if the system produces unexpected blocking or excessive alert volume.

4. Mathematical Formulation and Evaluation Metrics

4.1 Binary Classification

Binary classification performance is evaluated using true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Key metrics are:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F_1 = 2 \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \quad \text{FPR} = \frac{FP}{FP + TN}$$

Precision indicates the reliability of positive predictions, while recall measures detected positive events. F1 balances both, and FPR quantifies benign events incorrectly flagged. Accuracy should be supplemented with class-specific metrics, confusion matrices, precision–recall curves, and uncertainty intervals. Macro-averaging treats attack categories equally, whereas prevalence-weighted metrics reflect their observed distribution.

4.2 Threshold Selection

The decision threshold should balance missed attacks against false alarms. The expected classification cost is:

$$C(t) = C_{FN} FN(t) + C_{FP} FP(t)$$

where t is the decision threshold and C_{FN} and C_{FP} represent the costs of false negatives and false positives. Select thresholds using validation data and a documented cost model, reserving the test set for final evaluation. High-impact mitigation may also require confidence checks or corroborating evidence.

4.3 Latency and Throughput

End-to-end latency is measured from event availability at the sensor to the mitigation decision, or to confirmed enforcement when applicable:

$$T_{\text{total}} = T_{\text{collect}} + T_{\text{feature}} + T_{\text{infer}} + T_{\text{policy}} + T_{\text{enforce}}$$

Measure each component separately and report median, 95th-percentile, and 99th-percentile latency. Evaluate throughput under increasing request rates and representative concurrent workloads.

4.4 Mitigation Effectiveness

Mitigation must be assessed independently of classification performance. Authorized tests should measure attack completion, resource consumption, service availability, and collateral impact before and after enforcement. Containment effectiveness can be calculated as:

$$E_{\text{contain}} = \frac{A_{\text{baseline}} - A_{\text{mitigated}}}{A_{\text{baseline}}} \times 100\%$$

Here, A_{baseline} and $A_{\text{mitigated}}$ represent harmful activity without and with mitigation, respectively. Comparisons require consistent attack conditions, observation periods, and workloads. Report service availability and false-blocking rates alongside containment effectiveness.

5. Experimental Methodology

5.1 Research Design

The study adopts a controlled comparative design comprising offline classification and authorized application testing. The first stage evaluates detection performance, while the second measures real-time response and mitigation effectiveness. Testing must use isolated environments, synthetic accounts, controlled attack scripts, and documented rate limits. Third-party systems must not be tested without explicit authorization.

5.2 Dataset Selection

CIC-IDS2017 is proposed for network-flow benchmarking, supplemented by application-level test cases. The study should document dataset versions, download dates, label mappings, excluded features, missing values, and filtering procedures. Dataset limitations must be acknowledged because network-flow records alone cannot verify application vulnerabilities or successful exploitation.

5.3 Data Splitting and Leakage Prevention

Grouped and temporal data partitions should reduce information leakage and assess generalization. The protocol reserves an independent test set, fits preprocessing and feature selection on training data, tunes parameters using validation data, and evaluates the locked pipeline on the test set. Random splitting may serve as a supplementary benchmark but should not replace leakage-resistant evaluation.

5.4 Candidate Models

Four configurations are proposed:

Model A: Rule-based baseline using predefined security rules.

Model B: Random Forest classifier for structured network and application features.

Model C: Gradient-boosting classifier with comparable tuning procedures.

Model D: Hybrid model combining calibrated predictions, anomaly scores, application context, and policy constraints.

All models should use comparable evaluation conditions. Training time, model size, inference cost, and operational dependencies should also be recorded.

5.5 Hyperparameter Optimization

Hyperparameters should be tuned using grouped cross-validation or equivalent leakage-resistant procedures. Model selection should consider macro-F1, minority-class recall, false-positive rate, and latency. Search settings, random seeds, software versions, hardware, and stopping criteria must be documented. Test-set results should remain unseen until final evaluation.

5.6 Mitigation Evaluation

Controlled application tests should include normal browsing, repeated authentication failures, scanning, request bursts, benign traffic surges, authorized injection tests, unauthorized access attempts, model-service failures, and policy recovery. Each scenario should be repeated under consistent conditions. Measurements should include detection and enforcement latency, attack completion, legitimate-request success, false blocking, and recovery time. Attempted attacks must be distinguished from confirmed exploits.

5.7 Statistical Analysis

Results should include appropriate confidence intervals and descriptive statistics, such as means and standard deviations or medians and interquartile ranges. Paired comparisons should be used where justified by the experimental design. Statistical significance must be interpreted alongside practical trade-offs between detection performance, latency, and false-positive costs.

5.8 Reproducibility and Ethical Safeguards

The study should document datasets, code versions, configurations, feature definitions, labels, data splits, random seeds, and evaluation scripts. Sensitive data must be minimized and appropriately protected. Experiments should use authorized environments, reversible mitigation policies, and procedures to prevent accidental disruption while complying with applicable legal, ethical, and institutional requirements.

6. Results and Reporting Framework

6.1 Status of empirical results

The framework is technically specified, but no completed model training, independent test execution, or live mitigation measurement is provided with this manuscript. Therefore, numerical claims about accuracy, recall, F1-score, detection latency, or attack-containment effectiveness cannot responsibly be reported as findings. The following tables define the required reporting format for the completed study.

6.2 Classification performance

Table 3. Template for comparative model evaluation

Model	Precision	Recall	Macro-F1	False-positive rate	PR-AUC
Rule-based baseline	To measure	To measure	To measure	To measure	N/A where no score is available
Random Forest	To measure	To measure	To measure	To measure	To measure
Gradient boosting	To measure	To measure	To measure	To measure	To measure
Hybrid model	To measure	To measure	To measure	To measure	To measure

The completed analysis should identify whether improvements are consistent across attack classes. If the hybrid model increases recall but also substantially increases false positives, the trade-off should be quantified rather than summarized as unconditional superiority. Results should also be presented by class and, where possible, by capture scenario.

6.3 Computational and operational performance

Table 4. Template for real-time processing evaluation

Configuration	Median latency	95th-percentile latency	99th-percentile latency	Throughput	Memory usage
Rule-based baseline	To measure	To measure	To measure	To measure	To measure
Random Forest	To measure	To measure	To measure	To measure	To measure
Gradient boosting	To measure	To measure	To measure	To measure	To measure
Hybrid model	To measure	To measure	To measure	To measure	To measure

The study should examine performance at multiple request rates and concurrency levels. If queueing delay rises sharply under load, the system may not meet its operational objective even if its isolated inference time is low. Hardware, software versions, batching strategy, and the number of worker processes must be documented.

6.4 Mitigation outcomes

Table 5. Template for controlled attack-containment evaluation

Test scenario	Detection rate	Attack completion after response	Legitimate request success rate	False-blocking events	Recovery time
Authentication abuse	To measure	To measure	To measure	To measure	To measure
Request flooding	To measure	To measure	To measure	To measure	To measure
Injection test cases	To measure	To measure	To measure	To measure	To measure
Suspicious access sequence	To measure	To measure	To measure	To measure	To measure
Benign traffic surge	To measure	Not applicable	To measure	To measure	To measure

The results should separate preventive containment from vulnerability remediation. A WAF rule may prevent a particular exploit attempt without correcting vulnerable application code. Conversely, a code fix may remove the weakness even if a traffic classifier continues to flag unusual requests.

6.5 Ablation analysis

An ablation study should evaluate the contribution of each framework component. The full system should be compared against versions without application context, without anomaly scoring, without calibration, and without the mitigation policy layer.

Table 6. Proposed ablation-study design

Configuration	Supervised ML	Anomaly score	Application context	Risk-calibrated response
A: ML only	Yes	No	No	No
B: ML + anomaly detection	Yes	Yes	No	No
C: Contextual hybrid	Yes	Yes	Yes	No
D: Complete framework	Yes	Yes	Yes	Yes

The ablation study should establish whether each component provides a measurable benefit. The mitigation layer should be evaluated primarily on operational outcomes, while the detection components should be assessed on classification and calibration metrics.

6.6 Interpretation of anticipated outcomes

The proposed framework yields several testable hypotheses.

H1: Combining network and application telemetry will improve detection of selected web-attack scenarios relative to a flow-only model when the additional features contain reliable, nonredundant information.

H2: Group-aware and temporal evaluation will produce a more realistic estimate of generalization than a random row-level split.

H3: Risk-calibrated mitigation will reduce unnecessary blocking compared with immediate blocking on every positive prediction, although it may delay intervention for some attacks.

H4: A hybrid model may improve sensitivity to previously unseen behaviour, but its false-positive burden will depend on the quality of the benign baseline and the calibration of anomaly scores.

These hypotheses are not findings. They should be accepted, rejected, or qualified based on measured results and uncertainty estimates.

7. Discussion

7.1 Implications for Web Application Security

ML-based detection should complement, not replace, secure coding, access-control testing, dependency management, and vulnerability assessment. Runtime models can prioritize suspicious activity, but traffic patterns alone cannot reliably establish vulnerabilities. Therefore, ML detection should operate alongside static analysis, dynamic testing, configuration assessment, and code remediation.

7.2 Balancing Recall and False Positives

Aggressive detection thresholds may disrupt legitimate users and overwhelm security teams, whereas missed attacks can expose sensitive data. Risk-based policies should match responses to evidence confidence and potential impact. Alerts and monitoring suit uncertain cases, while stronger actions require more reliable evidence. Risk-score weights and thresholds should be documented, validated, and periodically reviewed according to application requirements.

7.3 TCP/IP Visibility and Encrypted Traffic

TCP/IP metadata helps identify abnormal connections and traffic patterns, but encryption limits payload visibility. Flow-based models alone cannot reliably detect every application-layer attack. Combining network telemetry with authorized reverse-proxy, firewall, identity, and application logs improves contextual analysis. Recording feature provenance also supports transparent investigations while minimizing sensitive-data collection.

7.4 Generalization and Concept Drift

Changes in traffic, application routes, users, and attack strategies can reduce model performance. Monitoring should track score distributions, alert volumes, confirmed incidents, and unseen feature values. Retraining should use reviewed data rather than automatic responses to every statistical change. Temporal testing and independent application evaluation are necessary to assess generalization.

7.5 Adversarial Robustness

Attackers may manipulate request timing, source distribution, and feature patterns to evade detection. Training-data contamination is another concern. Input validation, protected model artifacts, controlled retraining, and independent security testing are therefore essential. Explainability can support investigation but does not guarantee causal understanding or robustness.

7.6 Operational Resilience

ML services and feature pipelines can fail or generate inappropriate responses. Reliable deployment requires bounded timeouts, deterministic fallback rules, audit trails, policy versioning, rollback, and emergency disablement. Existing security controls should remain functional when ML inference becomes unavailable, and critical mitigation policies should be tested before production deployment.

7.7 Comparison with Conventional Approaches

Signature-based and deterministic rules offer predictable, interpretable behaviour, whereas ML can identify complex feature combinations. However, ML introduces additional data, maintenance, and monitoring requirements. The preferred approach should be determined through comparisons with conventional baselines using detection performance, latency, false positives, availability, auditability, and maintenance costs—not offline accuracy alone.

8. Limitations and Threats to Validity

The proposed framework has several limitations.

Dataset validity: Public datasets may not represent contemporary applications or diverse attack scenarios. **Label validity:** Network labels do not necessarily confirm actual vulnerabilities or successful exploitation. **External validity:** Results from one environment may not generalize to others.

Temporal validity: Group-aware and time-based testing reduces, but cannot eliminate, data leakage and distribution shift.

Measurement validity: Latency depends on hardware, traffic load, sensor placement, and measurement boundaries; inference time alone does not establish real-time performance.

Mitigation validity: Blocking attacks does not remediate underlying vulnerabilities, and containment must be evaluated separately from service availability and vulnerability remediation.

Reproducibility: Reliable replication requires documented preprocessing, software versions, hyperparameters, random seeds, and hardware.

Operational and ethical constraints: Experiments must protect personal data, prevent service disruption, and comply with applicable legal requirements. Initial testing should use authorized, isolated environments and minimized telemetry. These limitations define the evidence required for responsible validation.

9. Future Research Directions

Future research should evaluate the framework across diverse web applications and datasets to assess generalization under changing traffic patterns and authentication mechanisms. Cost-sensitive learning could incorporate attack severity and false-positive consequences, while calibrated uncertainty estimation could direct ambiguous events to monitoring or analyst review. Sequential models may improve detection of multi-stage attacks, provided their latency and computational costs remain acceptable. Federated and privacy-preserving learning could enable collaborative detection without exchanging raw traffic, subject to safeguards against model poisoning and privacy risks. Explainability should be assessed through analyst efficiency, investigation time, and decision consistency. Finally, a standardized mitigation benchmark should jointly measure attack containment, legitimate-request success, false-blocking rates, latency percentiles, recovery time, and resource overhead to enable meaningful comparisons beyond classification metrics.

10. Conclusion

This paper presents a proposed machine-learning-based framework for vulnerability-related threat detection and real-time mitigation in TCP/IP-based web applications. The framework integrates network-flow analysis, HTTP and application telemetry, supervised classification, anomaly scoring, contextual risk assessment, policy-controlled responses, and operational feedback.

The study emphasizes that vulnerability identification, attack detection, and attack mitigation are distinct tasks. Network-flow classifiers can identify suspicious patterns but cannot independently establish the presence or exploitability of a software vulnerability. Application-level evidence, controlled testing, and appropriate security verification are required for vulnerability-specific conclusions.

The proposed evaluation methodology combines leakage-resistant data splitting, competing ML baselines, class-sensitive metrics, latency measurement, and controlled mitigation experiments. It also addresses false-positive costs, model drift, encrypted traffic, adversarial adaptation, and safe operational fallback. These elements provide a practical basis for evaluating whether ML adds value beyond conventional security controls.

No measured results have been supplied in this manuscript; therefore, the proposed hypotheses remain to be tested. Future implementation should report verified model performance, confidence intervals, end-to-end mitigation latency, false-blocking rates, and attack-containment outcomes using documented datasets and authorized test environments. Only after such evaluation can the system's effectiveness, generalization, and suitability for publication be substantiated.

References

- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. <https://doi.org/10.1007/978-0-387-45528-0>
- Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32. <https://doi.org/10.1023/A:1010933404324>
- Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153–1176. <https://doi.org/10.1109/COMST.2015.2494502>
- Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3), Article 15. <https://doi.org/10.1145/1541880.1541882>
- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). <https://doi.org/10.1145/2939672.2939785>
- Dua, D., & Graff, C. (2019). *UCI Machine Learning Repository*. University of California, Irvine. <https://archive.ics.uci.edu/>
- European Union Agency for Cybersecurity. (2023). *ENISA Threat Landscape 2023*. ENISA. <https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023>
- Gharib, A., Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2016). An evaluation framework for intrusion detection dataset. In *2016 International Conference on Information Science and Security (ICISS)*. IEEE. <https://doi.org/10.1109/ICISSEC.2016.7885840>
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. <https://www.deeplearningbook.org/>
- Gu, G., Fogla, P., Dagon, D., Lee, W., & Skorić, B. (2008). Measuring intrusion detection capability: An information-theoretic approach. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*. ACM. Verify final bibliographic details against the publisher record before submission.
- He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 1263–1284. <https://doi.org/10.1109/TKDE.2008.239>
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity*, 2, Article 20. <https://doi.org/10.1186/s42400-019-0038-7>

14. Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyszogrod, D., Cunningham, R. K., & Zissman, M. A. (2000). Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*. IEEE. Verify the exact proceedings metadata before submission.
15. Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, 30. <https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html>
16. MITRE. (2025). *Common Weakness Enumeration (CWE)*. <https://cwe.mitre.org/>
17. MITRE. (2025). *MITRE ATT&CK*. <https://attack.mitre.org/>
18. National Institute of Standards and Technology. (2007). *Guide to Intrusion Detection and Prevention Systems (IDPS)* (Special Publication 800-94). U.S. Department of Commerce. <https://doi.org/10.6028/NIST.SP.800-94>
19. National Institute of Standards and Technology. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)* (NIST AI 100-1). <https://doi.org/10.6028/NIST.AI.100-1>
20. National Institute of Standards and Technology. (2023). *Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations* (NIST AI 100-2e2023). <https://doi.org/10.6028/NIST.AI.100-2e2023>
21. OWASP Foundation. (2021). *OWASP Top 10:2021*. <https://owasp.org/Top10/2021/>
22. OWASP Foundation. (2025). *OWASP Top 10:2025*. <https://owasp.org/www-project-top-ten/>
23. OWASP Foundation. (n.d.). *OWASP Application Security Verification Standard (ASVS)*. <https://owasp.org/www-project-application-security-verification-standard/>
24. OWASP Foundation. (n.d.). *OWASP Web Security Testing Guide*. <https://owasp.org/www-project-web-security-testing-guide/>
25. Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.
26. Scarfone, K., & Mell, P. (2007). *Guide to Intrusion Detection and Prevention Systems (IDPS)* (NIST Special Publication 800-94). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-94>
27. Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)* (pp. 108–116). SCITEPRESS. <https://doi.org/10.5220/0006639801080116>
28. Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy* (pp. 305–316). IEEE. <https://doi.org/10.1109/SP.2010.25>
29. Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. IEEE. <https://doi.org/10.1109/CISDA.2009.5356528>
30. Tian, Y., Pei, K., Jana, S., & Ray, B. (2018). DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering* (pp. 303–314). ACM. <https://doi.org/10.1145/3180155.3180220>
31. van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, 2579–2605. <https://www.jmlr.org/papers/v9/vandermaaten08a.html>
32. Zhang, J., & Zulkernine, M. (2006). Anomaly based network intrusion detection with unsupervised outlier detection. In *2006 IEEE International Conference on Communications* (Vol. 5, pp. 2388–2393). IEEE. Verify the final publication metadata and DOI against the publisher's record before submission.