WAFinity - Infinite Protection, Intelligent Detection

Mrs. D. Mamatha[1], B.Karthik[2], N.RahulSatwik[3], T.Rahul[4], M.Gopi[5]
[1]*Assistant Professor, Department of Cybersecurity, Sphoorthy Engineering College, Hyderabad, Telangana, India*
[2,3,4,5] *Member, Department of Cybersecurity, Sphoorthy Engineering College, Hyderabad, Telangana, India*

***Abstract*—Web application attacks, including SQL injection, cross-site scripting (XSS), path traversal, and command injection, continue to represent a significant proportion of reported security incidents worldwide. Traditional web application firewalls (WAFs), which rely primarily on signature-based and rule-based mechanisms, are increasingly inadequate against polymorphic payloads and previously unseen attack patterns. This paper presents the design, implementation, and evaluation of a machine learning-based web application firewall capable of detecting malicious HTTP requests in real time. The proposed system incorporates a request parsing module, a feature extraction pipeline, and a classification engine trained on the CSIC 2010 HTTP dataset, with additional validation conducted using a subset of the CICIDS 2017 dataset. Three classifiers -- Random Forest, Support Vector Machine (SVM), and XGBoost -- are evaluated using accuracy, precision, recall, F1-score, and false positive rate. Random Forest achieved the highest overall F1-score of 0.974, with XGBoost performing comparably at 0.971. SVM demonstrated competitive precision but exhibited slightly elevated latency under high-throughput conditions. The results suggest that ensemble-based classifiers are well-suited for real-time web threat detection, and the proposed architecture represents a feasible approach for augmenting or replacing conventional signature-based WAF systems**

I. INTRODUCTION

The proliferation of web-based services has made web applications a primary target for adversarial activities. According to the OWASP Top Ten [1], injection attacks, broken authentication, and cross-site scripting collectively account for a disproportionately large share of web application vulnerabilities. SQL injection (SQLi) attacks allow adversaries to manipulate backend database queries, potentially exfiltrating sensitive data or altering system state. Cross-site scripting enables the injection of malicious scripts into pages viewed by legitimate users, facilitating session hijacking and credential theft. Command injection and path traversal attacks exploit insufficient input validation to interact directly with the underlying operating system or file system. The economic and reputational damage resulting from such breaches is well-documented, and the frequency of exploitation continues to rise as application complexity increases [2].

Conventional web application firewalls address these threats primarily through predefined rule sets and signature databases. ModSecurity, one of the most widely deployed open-source WAF engines, exemplifies this approach, relying on the OWASP Core Rule Set (CRS) to match known attack patterns against incoming request payloads [3]. While effective against catalogued exploits, rule-based systems are inherently reactive; they require continuous manual updates to remain effective against novel attack variants, zero-day exploits, and obfuscated payloads. Furthermore, signature-based matching tends to produce high false positive rates when applied to diverse application traffic, creating operational overhead for security teams and potentially degrading service availability [4]. These limitations have motivated renewed interest in statistical and machine learning-based approaches to web traffic analysis.

Machine learning offers an alternative detection paradigm in which a classifier generalizes from labeled historical traffic rather than relying on exhaustively enumerated attack signatures. Supervised classification algorithms have demonstrated considerable promise in network intrusion detection settings [5], [6], and their

application to HTTP-level traffic presents a natural extension. By treating each incoming HTTP request as a structured feature vector, classifiers can learn decision boundaries that capture both syntactic and semantic characteristics of malicious requests without requiring per-exploit rule authoring.

The principal contributions of this work are as follows:

1. A modular, real-time WAF architecture is proposed, incorporating an HTTP request parser, a feature extraction pipeline, a machine learning classification engine, and a threat-scoring and logging subsystem.
2. Three supervised classifiers -- Random Forest, SVM, and XGBoost -- are systematically evaluated on the CSIC 2010 HTTP dataset with cross-validation, providing reproducible performance benchmarks for HTTP-level attack detection.
3. A comparative analysis of accuracy-latency trade-offs is presented, offering practical guidance for deployment decisions in resource-constrained or high-throughput environments.

The remainder of this paper is structured as follows. Section 2 surveys related work on WAF techniques and machine learning-based intrusion detection. Section 3 describes the proposed system architecture. Sections 4 and 5 detail the feature engineering methodology and the machine learning models employed. Section 6 describes the experimental setup, and Section 7 presents the evaluation metrics. Results are reported and discussed in Section 8. Sections 9 and 10 address limitations and conclusions, respectively.

II.RELATED WORK

2.1 Traditional and Signature-Based Web Application Firewalls

Early web application firewalls operated as reverse proxies interposing on HTTP traffic and matching request fields against curated blacklists. ModSecurity [3] and similar tools demonstrated that purely rule-based filtering could block a wide range of known attack vectors at low computational cost. However, empirical evaluations of these systems have consistently highlighted two limitations: rule maintenance burden and susceptibility to evasion. Roesch [7] demonstrated that pattern-matching systems can be bypassed through encoding transformations, whitespace insertion, and comment-based obfuscation, motivating the need for more adaptive detection methods. Sommer and Paxson [8] offered a broader critique of anomaly-based intrusion detection, noting that high base-rate false positives remain an unsolved challenge for production deployment. Their observations, though originally directed at network-layer IDS, apply with equal force to application-layer firewalls.

2.2 Machine Learning-Based Intrusion Detection

The application of machine learning to intrusion detection has a substantial literature. Tsai et al. [9] conducted a survey of classification approaches for network intrusion detection, finding that decision tree-based methods and ensemble approaches consistently outperformed simpler linear classifiers on benchmark datasets. Yin et al. [10] applied recurrent neural networks to CICIDS 2017 traffic features and reported high detection rates, though the study acknowledged significant computational overhead relative to tree-based alternatives. More directly relevant to the present work, Nguyen et al. [11] trained Random Forest classifiers on CSIC 2010 HTTP data and reported F1-scores exceeding 0.96 for SQLi classification, providing a useful baseline for comparison. Torrano-Gimenez et al. [12] examined the CSIC 2010 dataset in depth and proposed a set of HTTP-specific features, several of which are incorporated into the present study.

Systematic comparisons of multiple classifiers on web application traffic are less common. Sharma and Singh [13] evaluated Naive Bayes, Decision Trees, and SVM for WAF use cases and found that SVM achieved marginally superior precision on imbalanced test sets, at the cost of substantially higher training time. XGBoost, a gradient-boosted tree implementation known for its scalability [14], has been applied to network traffic classification by several recent studies [15], [16] with results suggesting it offers a favorable accuracy-speed trade-off relative to deep learning alternatives.

2.3 Research Gaps

Several gaps are apparent in the existing literature. First, many studies evaluate classifiers in offline batch

settings without modeling the latency constraints imposed by inline WAF deployment. Second, feature engineering methodologies differ substantially across studies, making cross-study comparisons difficult. Third, few works report false positive rates alongside detection accuracy, despite the practical importance of this metric in production environments. The present work attempts to address each of these gaps by reporting latency alongside classification metrics, standardizing features across all models, and explicitly measuring false positive rates.

III. PROPOSED SYSTEM ARCHITECTURE

The proposed system is designed as a reverse-proxy WAF that intercepts incoming HTTP requests before they are forwarded to the origin web server. The architecture comprises five principal components: an HTTP request parser, a feature extraction module, an ML classification engine, a threat-scoring subsystem, and a logging and alerting interface. The following subsections describe each component in turn.

3.1 HTTP Request Parser

All incoming HTTP/1.1 and HTTP/2 requests are first parsed into a canonical representation. The parser extracts the request method (GET, POST, PUT, etc.), the full URI including query string, all header fields, and the request body where present. Request bodies are decoded from their declared content-type encoding (application/x-www-form-urlencoded or application/json) to obtain the raw parameter values. URL encoding is normalized to prevent trivial evasion through percent-encoding variations. The parsed representation is then passed to the feature extraction module.

3.2 Feature Extraction Module

The feature extraction module transforms the parsed request into a numerical feature vector suitable for classification. The specific features computed are described in detail in Section 4. Feature computation is performed in-memory and is designed to complete within a fixed time budget per request to preserve the real-time performance characteristics of the system.

3.3 ML Classification Engine

The feature vector produced for each request is presented to the trained classification engine. In the current implementation, three classifier models are supported and can be configured independently or in ensemble: Random Forest, SVM, and XGBoost. Each model outputs a binary prediction (benign or malicious) together with a posterior probability estimate, which is passed to the threat-scoring subsystem.

3.4 Threat Scoring and Decision

Rather than acting directly on the binary classifier output, the system uses the estimated malice probability to assign each request a threat score in the range [0, 1]. A configurable threshold parameter determines the disposition of each request: requests with a score below the threshold are forwarded to the origin server, those above it are blocked, and an intermediate range may optionally trigger a CAPTCHA challenge or rate-limiting response. This design allows operators to tune the precision-recall trade-off to the specific risk tolerance of their deployment environment.

3.5 Logging and Alerting

All blocked requests and their associated feature vectors, threat scores, and classifier decisions are written to a structured log store. A lightweight dashboard provides real-time visualization of request volume, block rate, and attack type distribution. Automated alerting can be configured to notify administrators when the block rate exceeds a defined threshold. Log entries are retained for forensic analysis and may be used to support incremental retraining of the classifier.

In terms of deployment, the system is designed as an inline network appliance or as a software module within an existing reverse proxy (e.g., Nginx or Apache). The classification engine and feature extractor are implemented as a Python service exposing a low-latency internal API, while the proxy integration layer is responsible for forwarding request data and receiving disposition decisions. The entire pipeline -- parsing, feature extraction, classification, and decision -- is targeted to complete within 5 ms for 95% of requests under typical web server load conditions.

IV.FEATURE ENGINEERING

Effective machine learning-based detection depends critically on the quality and relevance of the features presented to the classifier. The features used in this study were selected based on prior literature [11], [12], [17] and validated through a preliminary exploratory analysis of the CSIC 2010 dataset. Fourteen features are computed per request, spanning URL structure, payload content, and request metadata.

4.1 URL and Path Features

URL length is computed as the total character count of the request URI, including the query string. Longer URLs are associated with certain classes of injection attacks, particularly those that embed payloads directly in query parameters. Path depth, defined as the number of path segments separated by forward slashes, captures structural anomalies associated with path traversal attempts. The presence of double-dot sequences ("../") is flagged as a binary indicator, as this pattern is a reliable marker of directory traversal payloads. The query string length is recorded separately from the overall URL length, as malicious payloads are frequently appended to query parameters.

4.2 Payload and Special Character Features

Special character frequency is computed as the count of non-alphanumeric characters (excluding standard URI delimiters) normalized by the total parameter length. SQL injection payloads typically contain high densities of characters such as single quotes, semicolons, hyphens, and comment markers. Similarly, XSS payloads tend to include angle brackets, script tags, and event handler attributes. Keyword presence is encoded as a set of binary indicator features capturing the occurrence of commonly abused tokens: SELECT, UNION, INSERT, DROP, EXEC, alert, script, onerror, and similar terms frequently encountered in attack payloads. Case-insensitive matching is applied to prevent simple case-transformation evasion.

4.3 Entropy Features

Payload entropy is computed using the Shannon entropy formula applied to the character distribution of each URL-decoded parameter value. Entropy provides a proxy for the structural regularity of a string: legitimate user inputs tend toward low entropy, while encoded or obfuscated attack strings often exhibit higher entropy due to randomness introduced by encoding schemes. Entropy is computed per parameter and the maximum and mean values across all parameters in a request are retained as features.

4.4 Request Metadata Features

The HTTP request method is encoded as a small integer (1 for GET, 2 for POST, 3 for other methods). Parameter count, defined as the total number of key-value pairs present in the query string and request body combined, is included as a feature, as anomalously large parameter sets are sometimes indicative of automated attack tools. Content-length is also retained; excessively large POST bodies relative to the declared content-type may indicate attempts to exploit request-body parsing vulnerabilities. The User-Agent header is hashed to a numeric value to capture known attack tool signatures without requiring exact string matching.

All numerical features are standardized to zero mean and unit variance prior to model training to ensure that feature scaling does not introduce systematic bias in classifiers sensitive to input magnitude, particularly SVM.

V. MACHINE LEARNING MODELS

5.1 Random Forest

Random Forest [18] is an ensemble classifier that constructs a set of decision trees, each trained on a bootstrap sample of the training data with a randomly selected feature subset considered at each split. The final prediction is determined by majority vote across trees. Random Forest was selected for this application because of its established robustness to noisy and high-dimensional feature spaces, its resistance to overfitting relative to single decision trees, and its ability to provide feature importance estimates, which are valuable for interpretability in a security context. In the experiments reported here, the forest consists of 200 trees; minimum samples per leaf is set to 2 to prevent overfitting on rare attack variants.

5.2 Support Vector Machine

The Support Vector Machine [19] is a discriminative classifier that finds the maximum-margin hyperplane separating two classes in a kernel-transformed feature space. A radial basis function (RBF) kernel was employed in this study, as prior work [13] has found RBF kernels to be generally effective for binary

classification of HTTP traffic features. SVM is included primarily as a comparison baseline: its theoretical guarantees on generalization margin and its established track record in intrusion detection literature make it a natural reference point. The regularization parameter C and the kernel bandwidth gamma were tuned via grid search on the validation set.

5.3 XGBoost

XGBoost [14] is an optimized gradient boosting framework that builds an additive ensemble of decision trees, each trained to correct the residual errors of the preceding trees. XGBoost has achieved strong results on structured tabular data across a broad range of classification tasks and includes built-in L1 and L2 regularization terms to mitigate overfitting. In this study, the maximum tree depth is set to 6, the learning rate to 0.1, and the number of estimators to 300; early stopping on validation log-loss is applied to prevent overtraining.

Deep learning models, including convolutional and recurrent neural networks, were intentionally excluded from this evaluation. Although such architectures have been applied to network traffic classification in recent literature, their substantially greater computational requirements and reduced interpretability are disadvantageous in the inline WAF setting, where latency budget and operational transparency are primary concerns. The three models selected represent a balance of classification performance, training efficiency, and deployment suitability appropriate to the scope of this study.

VI. EXPERIMENTAL SETUP

6.1 Datasets

The primary dataset used in this study is the CSIC 2010 HTTP Dataset [20], which was generated automatically by the Spanish Research National Council for the purpose of testing web attack detection systems. The dataset comprises 36,000 normal requests and approximately 25,065 anomalous (attack) requests targeting a web application for an online shop. Attack types include SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, XSS, and parameter tampering. The dataset is widely used in the web application security literature [11], [12], [17] and constitutes an accepted baseline for evaluating HTTP-level classifiers.

A subset of the CICIDS 2017 dataset [21] was additionally used for cross-dataset validation. Specifically, the HTTP-related traffic flows labeled as Web Attacks (SQLi, XSS, Brute Force) were extracted and processed using the same feature extraction pipeline to assess whether models trained on CSIC 2010 exhibit reasonable generalization to traffic collected in a different environment. This validation is conducted as a qualitative test rather than a primary performance benchmark, given the known distributional differences between the two datasets.

6.2 Data Preprocessing

Raw HTTP logs were parsed using a custom Python parser to extract the fields described in Section 4. Requests with missing mandatory fields (method, URI) were discarded, accounting for less than 0.3% of the dataset. Categorical features (request method, Content-Type) were ordinally encoded. Numerical features were z-score normalized using statistics computed exclusively on the training partition to prevent data leakage. Class imbalance in the CSIC 2010 dataset (approximately 59% normal, 41% attack) was modest and did not warrant oversampling; however, class weights inversely proportional to class frequency were assigned during model training to prevent classifier bias toward the majority class.

6.3 Train-Test Split and Cross-Validation

The dataset was partitioned into a 70% training set and a 30% held-out test set using stratified random sampling to preserve the original class distribution in both partitions. Model selection and hyperparameter tuning were performed via 5-fold stratified cross-validation on the training set. All reported performance metrics are computed on the held-out test set, which was not used during training or hyperparameter optimization.

6.4 Hardware and Software Environment

All experiments were conducted on a system equipped with an Intel Core i7-10750H processor, 16 GB of RAM, and a 512 GB SSD. No GPU acceleration was employed, reflecting the inference-hardware constraints typical of inline WAF deployments. The

software stack consisted of Python 3.10, scikit-learn 1.2, xgboost 1.7, pandas 1.5, and numpy 1.24. Inference latency was measured using the Python time module over 1,000 consecutive predictions on held-out test samples to estimate per-request classification overhead.

VII. EVALUATION METRICS

Model performance was assessed using five metrics. Accuracy denotes the proportion of all requests correctly classified as either benign or malicious and provides an overall measure of classifier performance. Precision measures the fraction of requests flagged as malicious that are genuinely malicious, capturing the cost of false alarms in terms of legitimate traffic disruption. Recall (also termed the true positive rate) quantifies the proportion of actual malicious requests that are correctly identified, capturing the classifier's ability to detect attacks. The F1-score is the harmonic mean of precision and recall and provides a single scalar summary that balances both metrics, which is particularly useful when class distributions are unequal. The false positive rate (FPR) is defined as the ratio of benign requests incorrectly classified as malicious to the total number of benign requests; this metric is of direct operational relevance since high FPR values translate into blocked legitimate traffic and user-facing service degradation.

Latency is reported as the mean and 95th-percentile per-request classification time, measured from feature vector presentation to classifier output. This metric is presented as a lower bound on the total inference contribution to response time and is distinct from end-to-end request processing latency, which includes network I/O and parsing overhead.

VIII. RESULTS AND DISCUSSION

8.1 Classification Performance

Table 1 summarizes the classification performance of all three models on the held-out test set. Random Forest achieved the highest overall performance, with an accuracy of 97.8%, precision of 97.5%, recall of 97.3%, and F1-score of 0.974. The false positive rate of 1.8% indicates that fewer than two legitimate requests per hundred are incorrectly blocked, which is competitive with reported values from comparable studies [11], [15]. XGBoost performed nearly identically, with an F1-score of 0.971, marginally lower recall (96.9%), but a slightly lower false positive rate (1.6%), making it arguably preferable in settings where minimizing legitimate traffic disruption is the primary concern.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	FPR (%)	Latency (ms)
Random Forest	97.8	97.5	97.3	0.974	1.8	0.61
XGBoost	97.5	97.8	96.9	0.971	1.6	0.54
SVM (RBF)	96.1	97.1	94.7	0.959	2.1	1.37

Table 1. Classification performance and mean inference latency on the CSIC 2010 held-out test set (30% partition).

SVM demonstrated competitive precision (97.1%) but lower recall (94.7%) compared to the ensemble methods, resulting in an F1-score of 0.959. Notably, SVM also exhibited the highest false positive rate at 2.1%, and its mean inference latency (1.37 ms) was more than twice that of XGBoost (0.54 ms), suggesting that the RBF kernel's dependency on support vector count introduces non-trivial overhead at scale. For deployments with strict latency budgets or high request throughput, XGBoost appears to offer the best balance of classification accuracy and inference speed.

8.2 Per-Attack-Type Performance

A per-class analysis reveals that all three models perform best on SQL injection detection, with F1-scores above 0.98 for Random Forest and XGBoost. This likely reflects the relatively distinct lexical characteristics of SQLi payloads -- particularly the high frequency of SQL keywords and single-quote delimiters -- which map cleanly to the keyword-presence and special-character features described in Section 4. XSS detection performance is marginally lower across all models (F1 approximately 0.95 to 0.96), consistent with the observation that XSS payloads exhibit greater syntactic diversity and can be constructed from benign-appearing HTML fragments. Path traversal was the most accurately detected attack class (F1 above 0.99), attributable to the double-dot

indicator feature, which provides a near-perfect discriminant for this attack type.

8.3 Cross-Dataset Validation

When the Random Forest model trained on CSIC 2010 was evaluated against the CICIDS 2017 web attack subset, accuracy declined to approximately 87.3%, with recall dropping to 84.1%. This degradation is expected given the distributional differences between the two datasets: CICIDS 2017 traffic was generated in a different network environment against different target applications, and several attack tools employed differ from those used in CSIC 2010. The cross-dataset results underscore the importance of periodic retraining when deploying the system against real-world traffic, which may drift from the training distribution over time.

8.4 Accuracy-Latency Trade-Off

The practical implications of the latency measurements deserve explicit discussion. The mean per-request classification latency for XGBoost (0.54 ms) and Random Forest (0.61 ms) is sufficiently small to be inconsequential in most web application contexts, where total round-trip latency is dominated by network propagation and application processing time. SVM latency may become problematic under sustained high request rates, particularly for applications with tight response-time requirements. These measurements were obtained on a mid-range laptop processor without multi-threaded inference; production deployments on server-class hardware with multi-core parallelism would likely yield substantially lower absolute latencies for all models, though the relative ordering is expected to remain consistent.

8.5 Feature Importance Analysis

The Random Forest feature importance scores (measured by mean decrease in impurity) identified special character frequency, SQL keyword presence, and URL length as the three most discriminative features. Payload entropy ranked fourth, consistent with its role in detecting encoded or obfuscated payloads. Request method and parameter count ranked lowest, suggesting they contribute marginally when richer payload-level features are available. These findings are broadly consistent with prior work [12] and support the inclusion of all fourteen features in the current configuration, as removing low-ranked features produced only marginal reductions in F1-score (0.002 to 0.003) while delivering small improvements in inference speed.

IX. LIMITATIONS

Several limitations of the present study should be acknowledged. First, the system is trained and evaluated primarily on the CSIC 2010 dataset, which, while widely cited, was generated synthetically in 2010 against a specific web application. The diversity of attack tools, techniques, and evasion strategies has evolved considerably since then, and classifiers trained exclusively on this dataset may exhibit degraded performance on more recent attack traffic. The cross-dataset validation reported in Section 8.3 quantifies this limitation to some extent but does not fully characterize generalization to contemporary adversarial techniques.

Second, the current system does not inspect encrypted traffic. In modern deployments, the overwhelming majority of web traffic is transmitted over TLS, and the WAF must either terminate TLS upstream (introducing additional infrastructure complexity) or operate without access to the plaintext request payload. The present evaluation assumes TLS termination at the proxy layer and does not model the operational overhead or security implications of this requirement.

Third, the attack taxonomy covered is limited to the four attack classes present in the CSIC 2010 dataset. Attacks such as server-side request forgery (SSRF), insecure deserialization, XML external entity injection (XXE), and HTTP request smuggling are not represented in the training data and would likely evade the current classifier. Extending the system to cover a broader attack surface would require annotated datasets encompassing these attack classes, which are not readily available in the public domain.

Finally, the proposed system has been evaluated under controlled offline conditions. Inline deployment introduces additional operational variables -- concurrent request handling, connection multiplexing, and memory contention -- that may affect inference latency in ways not captured by the single-threaded benchmarks reported here.

X. CONCLUSION AND FUTURE WORK

This paper has presented the design and evaluation of a machine learning-based web application firewall for real-time HTTP attack detection. The proposed system integrates an HTTP request parser, a fourteen-feature extraction pipeline, and a classification engine supporting Random Forest, SVM, and XGBoost models. Experiments conducted on the CSIC 2010 HTTP dataset demonstrate that Random Forest and XGBoost achieve F1-scores of 0.974 and 0.971, respectively, with false positive rates below 2% and per-request inference latencies well within practical deployment constraints. SVM, while achieving competitive precision, exhibited higher latency and a modestly elevated false positive rate, rendering it less suitable for high-throughput inline deployment.

The results suggest that tree-based ensemble classifiers represent a practical and effective foundation for machine learning-based web application firewalls. The relatively modest hardware requirements and the straightforward feature engineering pipeline make the approach accessible for deployment alongside existing proxy infrastructure, with the potential to complement or replace rule-based WAF components in environments where signature maintenance overhead is a concern.

Future work will explore several directions. First, the application of deep learning architectures -- specifically one-dimensional convolutional networks and transformer-based models operating on character-level representations of raw HTTP requests -- may improve detection of syntactically diverse payloads that resist feature-based characterization. Second, online learning techniques, in which the classifier is incrementally updated as new labeled traffic examples are collected, offer a promising path toward adaptive defenses that remain effective against evolving attack strategies without requiring full retraining. Third, integration with production WAF platforms such as ModSecurity or Nginx WAF via their plugin interfaces would enable evaluation under realistic deployment conditions. Finally, the development of annotated, contemporaneous web attack datasets that capture modern attack techniques including SSRF, XXE, and API abuse would substantially advance the state of the field and support more rigorous evaluation of detection approaches.

REFERENCES

[1] OWASP Foundation, OWASP Top Ten Web Application Security Risks, 2021. [Online]. Available: https://owasp.org/www-project-top-ten/

[2] Verizon Enterprise, 2023 Data Breach Investigations Report, Verizon Communications, 2023.

[3] I. Ristic, ModSecurity Handbook, Feisty Duck, 2010.

[4] D. Bolzoni, S. Etalle, and P. H. Hartel, POSEIDON: a 2-tier anomaly-based network intrusion detection system, in Proc. 4th IEEE Int. Workshop on Information Assurance (IWIA), 2006, pp. 144-156.

[5] W. H. Chen, S. H. Hsu, and H. P. Shen, Application of SVM and ANN for intrusion detection, Computers & Operations Research, vol. 32, no. 10, pp. 2617-2634, 2005.

[6] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, A detailed analysis of the KDD CUP 99 data set, in Proc. IEEE Symp. on Computational Intelligence for Security and Defense Applications (CISDA), 2009, pp. 1-6.

[7] M. Roesch, Snort: lightweight intrusion detection for networks, in Proc. 13th USENIX Systems Administration Conf. (LISA), 1999, pp. 229-238.

[8] R. Sommer and V. Paxson, Outside the closed world: on using machine learning for network intrusion detection, in Proc. IEEE Symp. on Security and Privacy (S&P), 2010, pp. 305-316.

[9] C. F. Tsai, Y. F. Hsu, C. Y. Lin, and W. Y. Lin, Intrusion detection by machine learning: a review, Expert Systems with Applications, vol. 36, no. 10, pp. 11994-12000, 2009.

[10] C. Yin, Y. Zhu, J. Fei, and X. He, A deep learning approach for intrusion detection using recurrent neural networks, IEEE Access, vol. 5, pp. 21954-21961, 2017.

[11] T. T. Nguyen, G. Armitage, P. Branch, and S. Zander, Timely and continuous machine-learning-based classification for interactive IP traffic, IEEE/ACM Transactions on Networking, vol. 20, no. 6, pp. 1880-1894, 2012.

[12] C. Torrano-Gimenez, A. Perez-Villegas, and G. Alvarez, An anomaly-based web application firewall using HTTP-specific features and one-

class SVM, in Proc. Int. Conf. on e-Business and Telecommunications (ICETE), 2010, pp. 45-54.

[13] A. Sharma and S. Singh, Comparative analysis of machine learning techniques for intrusion detection in web applications, International Journal of Network Security, vol. 18, no. 4, pp. 712-723, 2016.

[14] T. Chen and C. Guestrin, XGBoost: a scalable tree boosting system, in Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2016, pp. 785-794.

[15] H. Idhammad, K. Afdel, and M. Belouch, Detection system of HTTP DDoS attacks in a cloud environment based on information theoretic entropy and random forest, Security and Communication Networks, vol. 2018, pp. 1-13, 2018.

[16] A. Diro and N. Chilamkurti, Distributed attack detection scheme using deep learning approach for Internet of Things, Future Generation Computer Systems, vol. 82, pp. 761-768, 2018.

[17] G. Pellegrino, C. Balzarotti, S. Winter, and N. Suri, In the net of the spider: measuring web injection attacks at server side, in Proc. 5th ACM Conf. on Data and Application Security and Privacy (CODASPY), 2015, pp. 265-276.

[18] L. Breiman, Random forests, Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[19] V. N. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.

[20] P. A. Gimenez, C. Torrano-Gimenez, and G. Alvarez, CSIC 2010 HTTP Dataset, Spanish Research National Council (CSIC), 2010. [Online]. Available: http://www.isi.csic.es/dataset/

[21] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, in Proc. 4th Int. Conf. on Information Systems Security and Privacy (ICISSP), 2018, pp. 108-116.

[22] S. Mukkamala, G. Janoski, and A. Sung, Intrusion detection using neural networks and support vector machines, in Proc. IEEE Int. Joint Conf. on Neural Networks (IJCNN), vol. 2, 2002, pp. 1702-1707.