

# Automated Vulnerability Detection and Classification using Adaptive Machine Learning

Ms N. Kameshwari, Assistant Professor  
Bharath Institute of Higher Education and Research, Chennai, India.  
kameshwari.cse@bharathuniv.ac.in

M. Manasa Department of  
Computer Science and  
Engineering  
Bharath Institute of Higher  
Education and Research,  
Chennai, India.  
manasamandadi21@gmail.com

A.S. Lishma Department  
of Computer Science and  
Engineering  
Bharath Institute of Higher  
Education and Research,  
Chennai, India  
lishmalishu08786@gmail.com

**Abstract**—Web applications are an essential component of digital services but are increasingly vulnerable to security threats, including SQL Injection, Cross-Site Scripting, and Cross-Site Request Forgery. Manual vulnerability detection is a tedious task and susceptible to errors, necessitating automation and scalable approaches. This proposed work presents an Automated Web Application Vulnerability Detection model, which utilizes web crawling, intelligent fuzzing, and an Adapted Random Forest (ARF) algorithm to detect and rank vulnerabilities in real time. The suggested model extracts variables (features) such as payload length, frequency of special character, HTTP response parameters, and response variability, and computes severity scores to guide remediation. The model utilizes an input dataset containing HTTP requests and responses, including benign and malicious payloads targeting OWASP Top 10 vulnerabilities, and builds a web application, allowing user to view the output. Evaluation on this dataset shows enhanced performance over baseline models, achieving high accuracy, precision, recall, and F1-score. The framework adapts to obfuscated and zero-day payloads, ensuring robust detection of evolving threats. By combining automation, feature-driven adaptive RF algorithm, and severity-based reporting, the proposed model provides an efficient, and actionable solution for securing modern web applications.

**Index Terms**—Web Application Security; Automated Vulnerability Detection; OWASP Top 10; Adaptive Random Forest; Fuzzing

## I. INTRODUCTION

Web applications play an important role in modern digital services across sectors such as finance, healthcare, education, and retail. Their widespread use, however, makes them frequent targets for attacks like SQL Injection (SQLi), Cross-Site Scripting (XSS), Command Injection, and CSRF, all of which are highlighted in the OWASP Top 10. Traditional security methods such as manual penetration testing and signature-based scanners are slow, error-prone, and struggle to detect obfuscated or zero-day attacks. As web application breaches continue to rise, there is a growing need for automated and scalable vulnerability detection techniques.

To address these challenges, the proposed system combines intelligent web crawling, fuzzing, and an Adaptive Random Forest (ARF) classifier into a unified automated framework. The model identifies input fields, injects OWASP-based attack payloads, and analyzes responses to detect potential vulnerabilities. ARF improves detection accuracy and assigns

severity levels such as low, high, and critical, helping analysts prioritize remediation. The system is adaptable to real-time scenarios and can detect complex, obfuscated, and zero-day attack patterns, making it suitable for modern web security needs.

### A. Features of the Proposed Vulnerability Detection Model

The proposed model includes several features that support efficient detection of web application vulnerabilities. Automated crawling identifies input fields and attack vectors without manual effort, while intelligent fuzzing injects OWASP-based payloads to simulate real attack scenarios. A statistical feature extraction module then analyzes request-response data, including payload length, status codes, error messages, special characters, and pattern indicators. These features are classified using the Adaptive Random Forest (ARF) model, which predicts both the type and severity of vulnerabilities with improved accuracy and fewer false positives. Severity scoring helps analysts prioritize threats by categorizing vulnerabilities as low, high, or critical, and results are presented in a clear Excel report that highlights vulnerable fields, executed payloads, and risk levels. The model is also scalable, making it suitable for both small applications and large enterprise systems.

### B. Key Benefits of the Proposed Vulnerability Detection Model

The proposed model offers several key benefits for web security by dynamically identifying vulnerabilities before they can be exploited, reducing the risk of data breaches. Through automated crawling, intelligent fuzzing, statistical feature extraction, and the ARF algorithm, it removes the need for manual penetration testing and delivers a more efficient and consistent evaluation process. Its severity-based scoring method helps prioritize critical issues, while its ability to detect obfuscated payloads and zero-day attack patterns enhances overall protection. The system also generates clear reports, enabling security analysts to make informed decisions and strengthen web application security.

### C. Research Contributions of the Proposed Vulnerability Detection Model

The proposed work provides major contributions to web application security:

- **Integrated Automated Framework:** Combines web crawling, intelligent fuzzing, and the ARF classifier into a single automated system for reliable vulnerability detection.
- **OWASP Top 10 Payloads:** Uses standardized OWASP payloads to simulate real-world attacks and ensure broad vulnerability coverage.
- **Advanced Feature Engineering:** Extracts key features such as payload length, response codes, error messages, and special characters to improve classification accuracy.
- **Robust ARF Classifier:** Employs Adaptive Random Forest to accurately detect vulnerabilities while minimizing false positives.
- **Severity Scoring and Risk Prioritization:** Assigns Low, Medium, or High severity levels to support efficient remediation.
- **Comprehensive Reporting:** Generates clear reports showing vulnerable fields, executed payloads, and severity levels for technical and managerial use.
- **Scalability and Adaptability:** Performs consistently across small and large web applications, supporting diverse deployment environments.
- **Cost-Effective Solution:** Reduces manual effort and resource requirements, making it practical for real-world security operations.

## II. THE ROLE OF MACHINE LEARNING IN WEB VULNERABILITY DETECTION

Existing web security solutions often rely on manual testing, static analysis, and signature-based scanners. While these methods can detect known vulnerabilities, they struggle with new or evolving attack patterns and are not suitable for dynamic web applications. Earlier machine-learning approaches also faced challenges such as limited training data, high false positives, and difficulty analyzing real-time request-response behavior [3], [9].

These limitations led researchers to explore more advanced ML-based techniques. Recent work shows that intelligent fuzzing and automated classifiers improve scalability and detection accuracy, though issues with feature selection and noise handling remain [10], [11]. To address these gaps, the proposed model integrates Adaptive Random Forest with strong feature extraction to enhance vulnerability detection, classification, and severity prioritization in real time [15].

## III. STATE-OF-THE-ART IN WEB APPLICATION VULNERABILITY DETECTION

Web applications remain vulnerable to attacks such as SQL Injection, XSS, Broken Authentication, and Security Misconfiguration, which continue to appear in the OWASP Top 10 [1]. Traditional scanners based on signatures or rules often fail when attackers use obfuscation or zero-day techniques.

As a result, recent research from 2023–2025 has focused on classical machine-learning models such as Random Forest (RF), Decision Trees (DT), SVM, Logistic Regression, and Naïve Bayes, which offer lightweight and interpretable solutions compared to deep learning [2].

Pejs' and Kapui [3] evaluated several ML models, including Logistic Regression, Random Forest, Gradient Boosting, and SVM, for SQL Injection detection. Their study emphasized the importance of feature engineering (e.g., TF-IDF, skip-grams) and showed that Random Forest achieved the highest accuracy across diverse payloads. Similarly, ML-based XSS detection was explored in [5], where RF and SVM outperformed other models using simple lexical and structural features.

Qin et al. [9] improved XSS detection through multi-channel feature extraction, combining lexical, syntactic, and semantic features before classification with Random Forest. Their approach improved accuracy and reduced false positives without relying on deep learning. To enhance fuzzing, Chafjiri et al. [11] integrated ML models such as Random Forest and Decision Trees to guide payload selection, improving detection efficiency compared to purely random fuzzing.

Rathore et al. [10] proposed a combined crawling and ML framework that detects multiple OWASP vulnerabilities, showing that Random Forest consistently outperformed Naïve Bayes and Decision Trees. Broader surveys on fuzzing and web vulnerability detection also highlight the benefits of ML-assisted fuzzing and adaptive classification techniques [7], [12]. Advanced ML approaches, such as BERT-based request analysis for parameter tampering, were introduced in [13], and Random Forest has been widely recognized as a strong classifier for web traffic analysis [14]. The Adaptive Random Forest model (ARF) introduced in [15] further supports evolving data patterns, making it suitable for real-time detection systems.

Overall, recent studies show that classical ML models especially Random Forest provide accurate and scalable solutions for vulnerability detection. However, gaps remain in covering a wider range of OWASP Top 10 vulnerabilities, integrating real-time crawling, and supporting severity scoring. The proposed system addresses these limitations by combining crawling, intelligent fuzzing, Random Forest-based classification, and severity assessment to enhance automated web vulnerability detection.

### A. Issues in Existing Web Application Vulnerability Detection Systems

Existing systems exhibit several issues:

- **Dependence on Signature-Based Approaches:** Traditional scanners rely on fixed signatures to detect SQLi, XSS, and other attacks. These methods fail when attackers use obfuscated, modified, or zero-day payloads.
- **High False Positives and False Negatives:** Many scanners incorrectly flag normal inputs as malicious and also miss subtle attacks, reducing the reliability of detection results.
- **Scalability Limitations:** Modern web applications generate large and diverse traffic. Existing tools often struggle

to handle high-volume, real-time requests in enterprise environments.

- **Overfitting in ML Models:** Some machine-learning approaches overfit to specific datasets, causing poor performance when facing new or unseen attack patterns.
- **Lack of Adaptability:** Several systems are static and cannot quickly adjust to evolving attack techniques, making them less effective against sophisticated threats.
- **Limited Feature Engineering:** Many models ignore advanced features such as entropy, request–response behavior, or contextual metadata, leading to weaker detection accuracy.

The proposed model addresses these issues by integrating crawling, fuzzing, advanced feature extraction, and an Adaptive Random Forest classifier. This dynamic approach improves accuracy, reduces false positives, and provides a scalable and adaptable solution for detecting OWASP Top 10 vulnerabilities.

## IV. PROPOSED WEB APPLICATION VULNERABILITY DETECTION SYSTEM

The proposed system addresses the challenge of detecting web application vulnerabilities by combining web crawling, intelligent fuzzing, statistical feature extraction, and an Adaptive Random Forest (ARF) classifier. Unlike traditional scanners that rely on static signatures or manual testing, the system uses machine learning to analyze patterns in HTTP request–response behavior. It is capable of detecting multiple OWASP Top 10 vulnerabilities, including SQLi, XSS, Authentication Bypass, and Command Injection. The process begins with URL input, followed by automated crawling to identify input fields. Fuzzing then injects OWASP-based payloads to generate attack scenarios, and server responses are collected. A rule-based feature extraction module processes these responses and converts them into structured attributes for classification, as summarized in Table 1.

TABLE I: Features extracted from HTTP requests and responses

| Feature                     | Purpose                                                                        |
|-----------------------------|--------------------------------------------------------------------------------|
| HTTP response code          | Status code such as Eg. 200, 404, 500 to detect abnormal or error responses.   |
| Payload Length              | Size of HTTP request and response to identify unusually large or small inputs. |
| Entropy Analysis            | Measures randomness in requests/responses to detect obfuscated attacks.        |
| Keyword Presence            | Checks for SQL, JavaScript, and other attack-related keywords.                 |
| Special Character Frequency | Counts characters such as ', ", ;, ;, ; used in injection attacks.             |
| Error Message Patterns      | Parses server error messages for hints of potential vulnerabilities.           |
| Response Time               | Measures latency in server responses.                                          |

The ARF shows improved performance than conventional RF by utilizing dynamic feature weighting and adaptive bootstrap sampling for each decision tree. These features reduce

overfitting, improves generalization, and enables the model to adapt to changing payload distributions, zero-day vulnerabilities, and obfuscated attacks. Hence, ARF achieves high robustness and attained a high detection accuracy of 98.3%, but conventional RF achieves only 97%, providing a balance between model stability and accuracy. The performance of ARF is compared to other ML models and the output is presented in Table 2.

TABLE II: Web Application Vulnerability Detection Model Performance Analysis

| Model                | Acc  | Prec | Recall | F1   | Spec | ROC-AUC |
|----------------------|------|------|--------|------|------|---------|
| Proposed Adaptive RF | 98.3 | 0.99 | 0.97   | 0.98 | 0.99 | 0.98    |
| Random Forest        | 96.1 | 0.95 | 0.92   | 0.93 | 0.96 | 0.95    |
| Decision Tree        | 94.4 | 0.91 | 0.89   | 0.90 | 0.93 | 0.92    |
| Naive Bayes          | 92.2 | 0.89 | 0.86   | 0.87 | 0.91 | 0.88    |
| Neural Network       | 89.7 | 0.92 | 0.84   | 0.87 | 0.88 | 0.89    |
| LSTM                 | 87.5 | 0.85 | 0.82   | 0.83 | 0.86 | 0.85    |
| SVM                  | 90.9 | 0.88 | 0.86   | 0.87 | 0.90 | 0.88    |

The hyperparameter settings of the ARF model, shown in Table 3, are selected to achieve a balance between adaptability, and vulnerability detection accuracy. The en-semble size of 50 trees reduces computation and maintaining performance. Bootstrap resampling and Adaptive feature weighting are applied to improve model generaliza-tion to new payload patterns. The grace period and lambda value parameters are regulate tree growth and resampling behaviour, ensuring the model remains stable and responsive during vulnerability detection.

TABLE III: Hyperparameter configuration for ARF Classifier

| Parameter          | Value / Description                                       |
|--------------------|-----------------------------------------------------------|
| n_estimators       | Number of trees in the ensemble: 50                       |
| max_features       | Number of features considered for splitting a node: log2  |
| max_depth          | Max depth of each tree: 30                                |
| grace period       | Features observed per leaf before split: 100              |
| split criterion    | Function used to measure the quality of a split: Gini     |
| lambda value       | Poisson parameter for instance resampling: 6              |
| feature weighting  | To apply dynamic adaptive weighting: Enabled              |
| bootstrap sampling | To use adaptive bootstrap for resampling: True            |
| learning rate      | Controls the contribution of new data in adaptation: 0.05 |
| random state       | Seed for reproducibility: 42                              |

The Fig 1. displays ROC curves, which show the model overall ability to differentiate positives and negative samples among the compared ML models used for detect and classifying system vulnerabilities. The ROC plot illustrates model performance by plotting True Positive Rate (TPR) against False Positive Rate (FPR), evaluating how well the ARF identifies vulnerabilities. The ARF achieved best performance with an AUC value of 0.98, followed by the Random Forest (RF) and Decision Tree with AUC of 0.95 and 0.92, respectively. The comparison plot showed that the Neural Network and SVM attained AUC 0.90 and 0.88, respectively, Naïve Bayes and LSTM attained moderate performance with AUC of 0.87 and 0.85. The higher the curve above this diagonal reference line, the better the proposed model's classification ability. The proposed ARF showed better performance than traditional ML algorithms to finding security weaknesses in web applications.

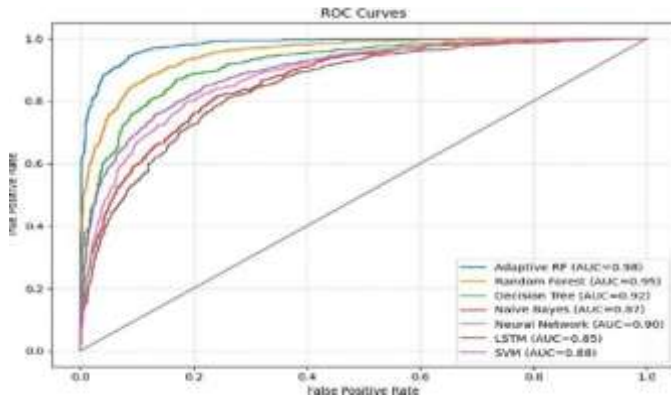


Fig. 1: ROC curve of ARF and other ML models for vulnerability detection

Figure 2 shows the training performance of different ML models in terms of accuracy and loss. The accuracy graph indicates that ARF improves steadily across epochs and reaches the highest accuracy of about 0.98, outperforming Random Forest and Decision Tree. Models such as Naïve Bayes, Neural Network, LSTM, and SVM show slower improvement. The loss graph demonstrates that ARF achieves the lowest training loss, indicating better generalization, while models like LSTM and SVM show higher loss and signs of overfitting. Overall, ARF delivers stronger and more stable performance during training compared to the other models.

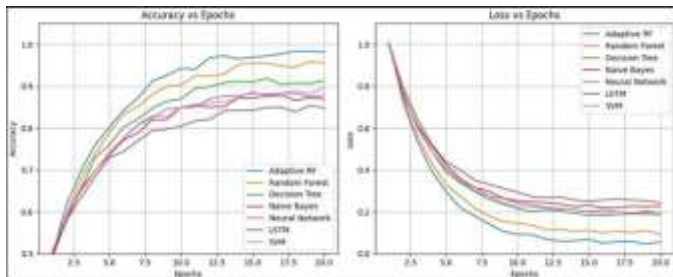


Fig. 2: Training performance of various ML classifiers: Accuracy and Loss Graph

## V. PROPOSED VULNERABILITY DETECTION SYSTEM ARCHITECTURE

Figure 3 shows the architecture of the proposed system, which integrates several automated modules for accurate vulnerability detection. The user provides a target URL, which is processed by the Flask backend. The crawler identifies input fields, and the fuzzing module injects OWASP payloads into them. Collected responses are then processed by the feature extraction module to generate attributes such as payload length, word count, and status codes. These features are classified using the ARF model, which also assigns severity

levels. The final results are presented through a simple, color-coded interface, with an option to download the report in Excel format.

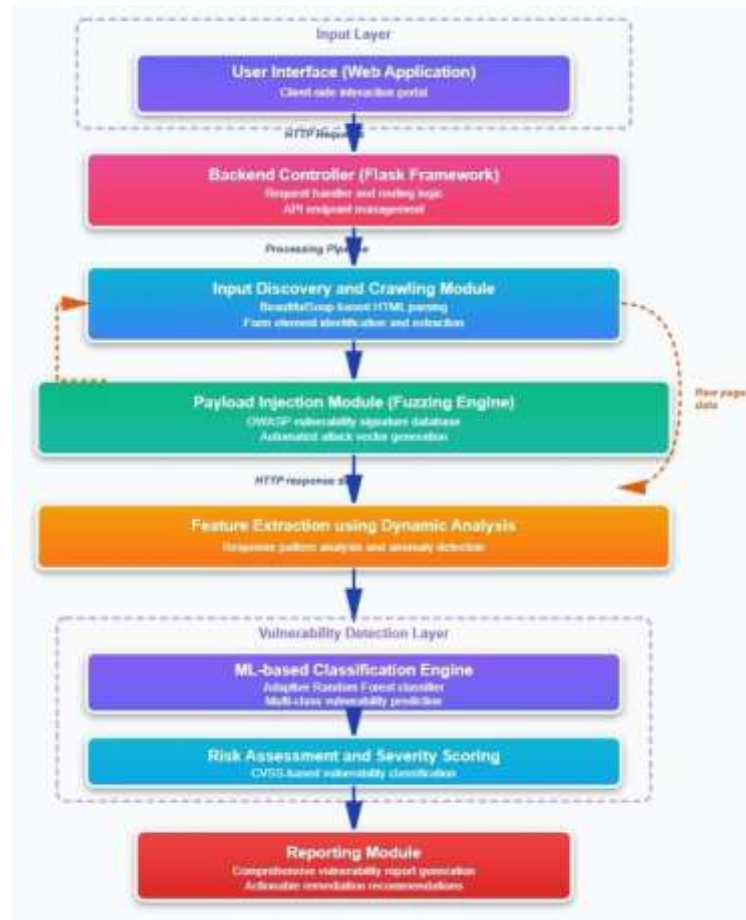


Fig. 3: Workflow of the proposed Vulnerability Detection System

Figure 4 presents the workflow of the ARF-based vulnerability detection system. The dataset includes OWASP Top 10 payloads labeled with vulnerability type and severity to ensure diverse attack coverage. After extracting features such as payload length, character patterns, keyword presence, status codes, and entropy, the data is split into 80% training and 20% testing to evaluate generalization. The ARF model then builds multiple decision trees using bootstrap samples and random feature subsets, and the final prediction is obtained through majority voting to enhance stability and reduce overfitting. The model adapts well to variations in payload structure and handles noisy or imbalanced datasets more effectively than traditional classifiers. Model performance is evaluated using accuracy, precision, recall, and F1-score to ensure reliable classification. The system outputs the detected vulnerability type and severity level, with results displayed in an Excel report for easy review and analysis.

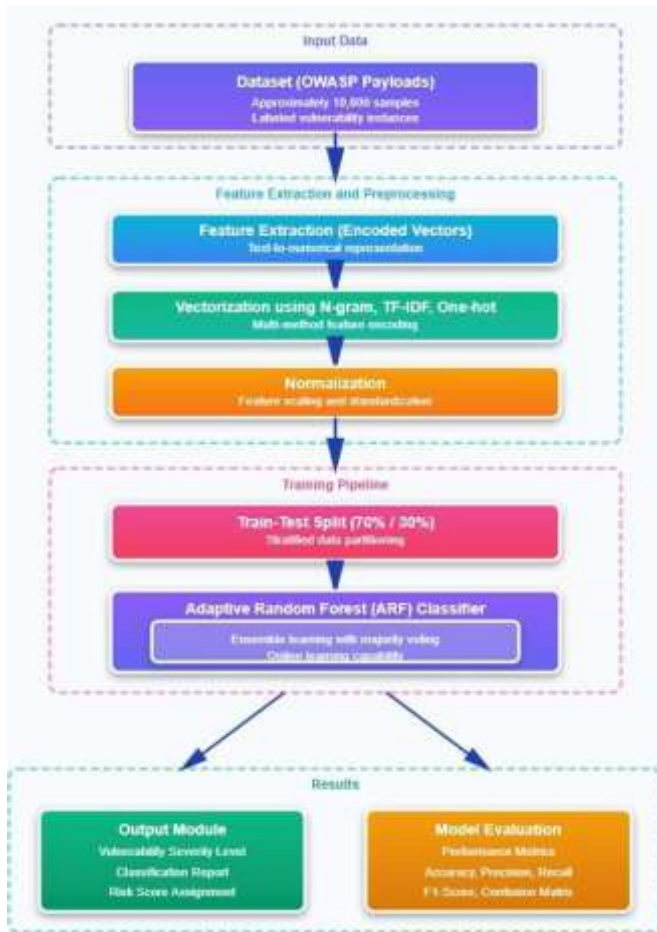


Fig. 4: Random Forest Architecture for Vulnerability Detection

## VI. METHODOLOGY

The proposed methodology follows a structured workflow for automated vulnerability detection. The process begins when the user enters a target URL, after which the crawler scans the application to identify input fields such as forms, text boxes, and search areas. In the next phase, OWASP Top 10 payloads are injected into these fields through intelligent fuzzing to simulate realistic attack attempts. The system then collects server responses including status codes, error messages, and output changes which serve as the basis for extracting meaningful features used in the detection phase.

### A. Vulnerability Assessment and Risk Score

Next, the fuzzing module injects predefined OWASP Top 10 payloads into the detected input fields to simulate real attack scenarios. The system then records all server responses, including status codes, error messages, and output changes. These responses form the raw data for feature extraction.

In the detection phase, the Random Forest classifier evaluates features such as payload length, special character count, SQL/JavaScript keywords, entropy, and HTTP status codes to predict the vulnerability type and severity. This approach

ensures an automated and scalable method for testing web application security.

The normalized risk score is calculated using:

$$\text{Risk}_{\text{norm}} = \frac{\text{Risk}_i - \text{Risk}_{\text{min}}}{\text{Risk}_{\text{max}} - \text{Risk}_{\text{min}}} \quad (1)$$

where  $\text{Risk}_i$  is the raw risk score of a request, and  $\text{Risk}_{\text{min}}$  and  $\text{Risk}_{\text{max}}$  are the minimum and maximum scores observed in the dataset.

### B. Automated Classification and Zero-Day Handling

The ARF model classifies requests as benign or malicious and further identifies the type of attack, such as SQLi, XSS, Command Injection, or File Inclusion. The system is trained with obfuscated and augmented payloads, allowing it to recognize zero-day and adaptive attacks by detecting subtle deviations in request patterns.

### C. Prioritization and Visualization

Vulnerabilities are prioritized based on their impact, giving higher importance to severe attacks like SQL Injection. The Streamlit dashboard displays request classifications, detected vulnerabilities, severity levels, and alerts, helping analysts quickly understand and act on the results.

## VII. EXPERIMENTAL RESULTS AND DISCUSSION

Figure 5 shows the scanner interface, which accepts target URLs and payload limits (default 30). Full Fuzz enables deeper form fuzzing for sites like DVWA or Juice Shop. After selecting Start Scan, the tool crawls the site, injects payloads, and detects issues such as SQL Injection, XSS, file inclusion, and insecure headers. The Download Latest Report option provides detailed findings with severity levels and fixes. All scanning must be done only on authorized test sites to stay within legal and ethical boundaries.



Fig. 5: Automated Web Vulnerability Scanner Input Interface

Figures 6 and 7 present the scan results. The system identified a high-severity Broken Authentication issue on <http://testphp.vulnweb.com/> with a confidence of 0.4067, indicating risks such as account compromise. Strengthening MFA, session handling, and password-reset controls is recommended, along with focused manual testing and monitoring.

The payload analysis in Figure 7 shows a simple 27-character plain-text input with six special characters and no scripts or SQL keywords. Only a single Base64 pattern appeared in the response. These characteristics help the system assess input behaviour and determine likely vulnerabilities.



Fig. 6: Web Vulnerability Scan Results Summary



Fig. 7: Feature Values of First Test Payload

Figure 8 shows multiple detected issues with their payloads, confidence scores, severities, and HTTP codes. The scan flagged two critical SQL Injection cases, major XXE and XSS risks, and minor issues like Insecure Deserialization and Broken Access Control. Insufficient Logging scored 0.625. All responses were HTTP 200, indicating gaps in input validation and server configuration.

[illegible]

Fig. 8: Detailed Vulnerability Report (Excel Export)

## VIII. CONCLUSION AND FUTURE WORK

As web platforms became central to daily operations, they also became easy targets for attacks that expose sensitive data and reduce user trust. The Automated Web Application Vulnerability Detection System addresses this by unifying crawling, fuzzing, and feature-based classification. The system scans the given URL, identifies input points, and injects OWASP-based

test payloads for issues like SQLi, XSS, and CSRF. It observes the application's responses in real time, extracts indicators such as errors, latency, and payload behaviour, and classifies them using an Adaptive Random Forest to determine vulnerability type and severity. A final report lists affected fields, risk levels, and mitigation steps. This automated approach reduces the need for manual penetration testing and offers a cost-effective, scalable solution, especially for small and medium businesses. It also supports broader digital-security goals by strengthening infrastructure, enhancing online safety, and helping organizations reduce losses from cyberattacks.

## REFERENCES

- [1] OWASP Foundation, “OWASP Top Ten Web Application Security Risks,” 2023.
- [2] A. Zhang, Y. Zhang, Y. Xu, C. Wang, and S. Li, “Machine learning-based fuzz testing techniques: A survey,” *IEEE Access*, vol. 12, pp. 151380–151399, 2024.
- [3] B. Pejcs and N. Kapui, “SQLi detection with machine learning: A data-source perspective,” in *Proc. 20th Int. Conf. Security and Cryptography (SECRYPT)*, 2023, pp. 642–648.
- [4] C. Wang, W. Meng, C. Luo, and P. Li, “Predator: Directed web application fuzzing for efficient vulnerability validation,” in *Proc. IEEE Symposium on Security and Privacy (SP)*, San Francisco, USA, 2025.
- [5] F. Ali and K. Rahman, “Detecting advanced cross-site scripting attacks using classical machine-learning models,” *Applied Computing Review*, vol. 10, no. 1, pp. 12–20, 2025.
- [6] S. K. Shukla and P. Patel, “Machine-learning-based detection of injection attacks in web applications,” *Computers & Security*, vol. 126, pp. 102–118, Elsevier, 2023.
- [7] I. P. A. Dharmmaadi, E. Athanasopoulos, and F. Turkmen, “Fuzzing frameworks for server-side web applications: A survey,” *International Journal of Information Security*, vol. 24, pp. 197–220, Springer, 2025.
- [8] J. Lee and K. Park, “Automated web vulnerability scanning: A systematic review,” *Future Generation Computer Systems*, Elsevier, vol. 152, pp. 450–466, 2024.
- [9] L. Wang and Z. Cheng, “Machine learning approaches for web vulnerability detection,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 1450–1464, 2022.
- [10] R. Sharma and P. Singh, “Machine learning approaches for automated vulnerability detection in web applications,” *Journal of Network and Applications*, vol. 16, no. 1, pp. 1–12, 2025.
- [11] T. Lin, Y. Hu, and J. Wang, “Machine learning fuzzers with machine learning for vulnerability detection: A survey,” *Computers & Security*, vol. 139, p. 103667, Elsevier, 2024.
- [12] K. Amrin and L. Bao, “ML-based fuzzing for modern web applications,” *Computers & Security*, vol. 139, p. 103774, Elsevier, 2024.
- [13] Y. Chen, J. Li, and R. Zhao, “BERT-based parameter tampering vulnerability detection in web applications,” *Symmetry*, vol. 17, no. 7, p. 985, MDPI, 2025.
- [14] H. Liu, X. Zhang, and T. Feng, “Web security analysis using Random Forest classifiers,” *ACM Transactions on the Web*, vol. 17, no. 2, pp. 1–20, 2023.
- [15] G. Bifet, J. Read, I. Žilobaite, and A. Bouchachia, “Adaptive Random Forests for evolving data streams,” *Machine Learning Journal*, Springer, vol. 110, pp. 1–25, 2020.
- [16] Source of data: <https://www.kaggle.com/datasets/cyberprince/web-application-payloads-dataset>