A PRELIMENERY PROJECT REPORT
ON
"Web Application Fuzzer "

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE
IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF ENGINEERING (INFORMATION TECHNOLOGY)

SUBMITTED BY

Mr. Lokesh Pusdekar (B400730362)
Mr. Sidhhant Mane (B400730348)
Mr. Shubham Khaire (B400730342)
Mr. Prajwal Kedar (B400730341)

Under the guidance of

Prof. Pranita Ingale

DEPARTMENT OF INFORMATION TECHNOLOGY

JSPM'S, BHIVARABAI SAWANT INSTITUTE OF TECHNOLOGY & RESEARCH

GAT NO:720 /1&2 . NAGAR ROAD, WAGHOLI, PUNE-412207
SAVITRIBAI PHULE PUNE UNIVERSITY

2024 -2025

JSPM'S, BHIVARABAI SAWANT INSTITUTE OF TECHNOLOGY & RESEARCH
WAGHOLI, PUNE-412207

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project report entitles

"Web Application Fuzzer "

Submitted by

Mr. Lokesh Pusdekar (B400730362)
Mr. Sidhhant Mane (B400730348)
Mr. Shubham Khaire (B400730342)
Mr. Prajwal Kedar (B400730341)

are bonafide students of this institute and the work has been carried out by him/her under the supervision of **Prof. Pranita Ingale** and it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University, for the award of the degree of **Bachelor of Engineering** (Information Technology).

Prof. Pranita Ingale — Project Guide

Prof. Rekha Kotwal — Project Coordinator

Dr. Vinod Wadne — HOD

Dr. T. K. Nagaraj — Principal

External Examiner

Place : Wagholi, Pune
Date :

ACKNOWLEDGEMENT

It gives us great pleasure in presenting the preliminary project report on. **"Web Application Fuzzer "**.We would like to take this opportunity to thank our internal guide **Prof. Pranita Ingale** for giving me all the help and guidance we needed. We are really greatful to our project coordinator **Prof. Rekha Kotwal** for her kind support. Her valuable suggestions were very helpful.

We are also obliged to **Dr. Vinod Wadne** Head of Information Technology Department, Bhivarabai Sawant Institute Of Tecnology and Research, Pune for his valuable time, support, comment, suggestions and persuasion.

We would like to thanks our Principal **Dr. T. K. Nagaraj**, for allowing us to pursue our project in this institute.

We would also like to thank the institute for providing the required facilities, internet access and important books.

Mr. Lokesh Pusdekar (B400730362)
Mr. Sidhhant Mane (B400730348)
Mr. Shubham Khaire (B400730342)
Mr. Prajwal Kedar (B400730341)

ABSTRACT

This report presents the implementation of a web application fuzzer designed to identify security vulnerabilities in web applications. The fuzzer uses predefined payloads to test for potential security weaknesses, particularly focusing on cross-site scripting (XSS). Additionally, a machine learning (ML) model has been incorporated to enhance the accuracy and precision of vulnerability detection. The report details the system architecture, initialization process, machine learning approach, and comparative analysis of model accuracy

TABLE OF CONTENTS

LIST OF ABBREVATIONS

ABBREVIATION	ILLUSTRATION
VPN	Virtual Private Network
IP	Internet Protocol
IDS	Intrusion Detection System
TCP	Transmission Control Protocol

LIST OF FIGURES

LIST OF TABLES

1. Introduction

Web applications play a crucial role in modern digital ecosystems, handling sensitive data, financial transactions, and personal user information. However, they are frequently targeted by cyber attackers exploiting input validation vulnerabilities such as Cross-Site Scripting (XSS), SQL Injection, Command Injection, and other OWASP Top 10 threats.

The Web Application Fuzzer is an automated security testing tool designed to proactively detect these vulnerabilities by:

- Injecting various malicious payloads into web input fields
- Analyzing system responses to detect security weaknesses
- Logging findings in real-time for further security assessment

How This Fuzzer Enhances Traditional Approaches

- Limitations of Traditional Fuzzing: Static payload-based fuzzing often fails against evolving attack techniques.
- Our Improvement: Integrating Machine Learning models helps identify anomalies and classify potential security threats more effectively.
- Technologies Used: Selenium WebDriver for automated browsing, Flask for backend communication, and WebSockets for real-time updates.

To combat these threats, traditional methods like manual penetration testing and static code analysis are commonly used. However, they are time-consuming, require skilled security professionals, and may not scale effectively across large or frequently updated applications. In contrast, automated fuzz testing (fuzzing) offers a more efficient and scalable solution by systematically injecting malformed or unexpected inputs into the application to uncover vulnerabilities.

This system offers a scalable and efficient method to automate security testing, reducing human effort while improving detection accuracy.

Advantages:

- Automated Security Testing
- Eliminates the need for manual penetration testing, saving time and effort.
- Scans multiple web applications efficiently without human intervention.
- Machine Learning-Driven Analysis
- Improves accuracy in detecting vulnerabilities by reducing false positives.
- Re-tests known vulnerabilities after updates to ensure they are fixed.
- Handles authentication tokens and sessions for secured web pages.
- Can integrate with additional tools and ML models for advanced testing.
- Can be enhanced with dashboards and graphs for better threat visualization.
- Learns from new data over time, improving detection accuracy and adaptability.
- Fits into DevSecOps workflows, helping developers catch issues early during builds.
-

Disadvantages:

- Potential for False Positives/Negatives
- While machine learning improves detection, some anomalies may still be misclassified.
- Requires constant retraining to maintain accuracy.
- Limited to Input-Based Attacks
- Primarily detects injection and input-based vulnerabilities.
- Cannot detect logical security flaws, authentication bypasses, or business logic vulnerabilities.
- Requires knowledge of ML, Python, and web testing concepts for proper use.
- Cannot directly fuzz encrypted or encoded inputs like JWT tokens without extra decoding.
- May require custom scripts for Single Page Applications that use JavaScript routing.
- Resource consumption increases significantly with large-scale or enterprise apps.
- Heavy usage may slow down applications and consume significant system resources.

- Unauthorized use on live or third-party systems may breach cybersecurity laws or terms of service.

1.1 Motivation

As cyber threats evolve, web applications face increasing risks from automated bot attacks, zero-day exploits, and injection vulnerabilities. Security testing must move beyond manual penetration testing to an automated, intelligent approach.

Why Automated Security Testing?

- Manual security audits are time-consuming and expensive.
- Attackers increasingly use automated attack bots, so security defenses must match their speed.
- Many vulnerabilities remain undetected until an actual attack occurs.
- Compliance with cybersecurity standards (e.g., OWASP, GDPR, PCI DSS) requires rigorous security testing.

A machine learning-powered Web Fuzzer helps mitigate these challenges by:

- Automatically detecting vulnerabilities before they are exploited.
- Providing structured datasets for continuous improvement of security measures.
- Minimizing human intervention while increasing test coverage and accuracy.

With the rapid expansion of web-based technologies and the increasing reliance on online services, web applications have become prime targets for cyberattacks. Attackers exploit even the smallest weaknesses to gain unauthorized access, exfiltrate data, or disrupt services. Manual security testing, while effective, is time-consuming, labor-intensive, and not scalable for modern agile development environments. Traditional scanners also struggle to keep up with evolving attack techniques and dynamic application behaviors. This motivated us to develop an automated, intelligent solution capable of identifying vulnerabilities efficiently and accurately. By integrating machine learning into fuzz testing, our system can not only simulate real-world attacks using payload injections but also learn from patterns and responses to detect complex threats. The goal is to provide a proactive and adaptive tool that minimizes human effort.

1.2 Problem Definition:

The objective of this project is to develop a **Web Application Fuzzer** capable of detecting vulnerabilities through automated testing, payload injection, and response analysis.

2. Literature Survey

1. Fuzzing Frameworks for Server-Side Web Applications:

Traditional vs. Modern Fuzzing Approaches

Traditional fuzzing frameworks such as Sulley, Peach, and American Fuzzy Lop (AFL) primarily focus on client-side vulnerabilities like JavaScript injections and browser exploits. However, server-side fuzzing has gained significant attention due to the increasing complexity of web applications and backend systems.

How Server-Side Fuzzing Works:

Server-side fuzzing targets the underlying infrastructure of web applications, including:
SQL Injection (SQLi): Bypassing authentication and exfiltrating sensitive data.
Remote Code Execution (RCE): Injecting malicious commands to execute unauthorized operations.
Authentication Bypass: Exploiting weak session management mechanisms.
Deserialization Attacks: Targeting insecure object deserialization in languages like Java and Python.

Popular Server-Side Fuzzing Frameworks:

Sulley Framework: Generates structured test cases and supports stateful fuzzing.
Peach Fuzzer: Uses data modeling to generate diverse payloads for structured protocols.
AFL (American Fuzzy Lop): Uses instrumentation-based fuzzing to mutate inputs and discover crashes.
Recent advancements in server-side fuzzing leverage machine learning and symbolic execution to improve test coverage and detection efficiency.

2. Fuzzing Techniques in Web Applications & Beyond:

Fuzzing techniques have evolved from simple random input injection to intelligent, adaptive fuzzing approaches. These techniques are used not only for web applications but also for APIs, network protocols, and embedded systems.

- Classification of Fuzzing Techniques
- Generation-Based Fuzzing

- Uses predefined rules and grammars to construct test cases.
- Ideal for testing applications that follow structured input formats (e.g., JSON, XML, SQL queries).
- Example: Grammarinator, which generates inputs based on language grammars.

Mutation-Based Fuzzing:

- Takes existing valid inputs and mutates them to generate new test cases.
- Common mutation techniques: Bit flipping, string manipulation, boundary value testing.
- Example: AFL (American Fuzzy Lop) mutates inputs intelligently based on feedback from previous crashes.

Hybrid Fuzzing (Combination of Static & Dynamic Analysis):

- Uses static analysis to extract input structures and dynamic analysis to modify them.
- Enhances code coverage and finds deep vulnerabilities.
- Example: Driller, a hybrid fuzzer combining AFL and symbolic execution.

Beyond Web Applications: Expanding Fuzzing Scope

- APIs & Web Services: REST and GraphQL APIs require specialized fuzzing tools like RESTler to handle complex query structures.
- Authentication & authorization fuzzing checks for bypass vulnerabilities.
- Network Protocols
- Tools like Boofuzz and Scapy generate malformed packets to test TCP/IP stack vulnerabilities.
- Embedded Systems: Firmware fuzzing tests IoT and embedded devices for security flaws.
- Example: P2IM (Peripheral-Peripheral Interaction Modeling) for hardware fuzzing.

3. Blackbox Fuzzing Approaches to Secure Web Applications:

Blackbox fuzzing is one of the most widely used security testing techniques, as it requires no access to the application's source code. Instead, it simulates external attacks by injecting malformed or malicious inputs.

How Blackbox Fuzzing Works:

- Target Identification: The fuzzer identifies attack surfaces like form fields, URL parameters, and HTTP headers.
- Payload Injection: Malicious or unexpected inputs are inserted into application interfaces.
- Response Analysis: The fuzzer checks for anomalies like crashes, slowdowns, or unusual HTTP responses.
- Logging & Reporting: Security issues are recorded for further analysis.

Popular Blackbox Fuzzing Tools:

- OWASP ZAP: Actively scans web applications for vulnerabilities like XSS and SQLi.
- Wfuzz: A command-line fuzzer designed for brute-force testing of web parameters.
- Skipfish: Uses recursive crawling and payload injection to detect security flaws.

Challenges in Blackbox Fuzzing:

- High False Positives: Many detected issues may not be real vulnerabilities.
- Limited Code Coverage: May miss vulnerabilities deep in application logic.
- Blocked by Security Mechanisms: Web Application Firewalls (WAFs) and rate-limiting mechanisms can detect and block fuzzing attempts.

To improve blackbox fuzzing, AI-driven models are now being used to prioritize high-risk inputs.

4. Effective Fuzzing of Web Applications for Server-Side Vulnerabilities:

Why Server-Side Fuzzing is Challenging?

Session & Authentication Handling: Modern web applications require valid user sessions and authentication tokens.

- Complex Backend Logic: Unlike client-side fuzzing, server-side fuzzing must simulate real user interactions.

- Asynchronous Processing: Server-side logic often involves background tasks, delayed responses, and multi-threading, making real-time fuzzing difficult.

Advanced Techniques for Server-Side Fuzzing:

- Intelligent Payload Crafting: Uses context-aware payloads to bypass input sanitization.
- Guided Fuzzing: Leverages symbolic execution to target unexplored application paths.
- ML-Enhanced Vulnerability Detection: Uses machine learning models to predict high-risk inputs based on past attack data.

Recent Advancements:

- FuzzGen: Generates API fuzzers automatically for server-side testing.
- Redqueen: Uses feedback-based fuzzing to improve mutation strategies dynamically.

5. Machine Learning-Based Fuzz Testing Techniques:

Machine Learning (ML) has transformed fuzz testing by making it adaptive, intelligent, and more efficient.

How ML is Used in Fuzzing?

1.Supervised Learning for Vulnerability Prediction

- Trains ML models on past vulnerability data to predict the likelihood of an input causing a security flaw.
- Example: Random Forest classifiers trained on historical web security incidents.

2.Reinforcement Learning (RL) for Optimized Fuzzing

- Uses reward-based training to adjust fuzzing strategies dynamically.
- Example: Deep reinforcement learning optimizes payload selection by learning from previous fuzzing attempts.
- Generative Adversarial Networks (GANs) for Test Case Generation
-

GANs generate synthetic attack payloads that mimic real-world attack patterns.

Example: FuzzGAN, which produces new security test cases beyond human-crafted payloads.

Why ML-Driven Fuzzing is Superior?

- Better Input Selection = Reduces false positives & focuses on high-risk areas.
- Adaptive Learning = Adjusts test cases based on real-time results.
- Higher Efficiency = Finds vulnerabilities faster compared to brute-force fuzzing.

Real-World Implementations:

- Microsoft Security Risk Detection: Uses ML for intelligent fuzzing in Windows applications.
- Google ClusterFuzz: An AI-driven fuzzing system used in Chromium security testing.

3.Software Requirements Specification

3.1 Introduction

Web applications play a crucial role in modern digital ecosystems, handling sensitive data, financial transactions, and personal user information. However, they are frequently targeted by cyber attackers exploiting input validation vulnerabilities such as Cross-Site Scripting (XSS), SQL Injection, Command Injection, and other OWASP Top 10 threats.

3.1.1 Project Scope

To develop and identify the spammers and spam content using reviews and performs four categories of features including review-behavioral, user-behavioral, review-linguistic and user-linguistic.

3.1.2 User Classes and Characteristics

User's

- Registration
- Login (If Needed)
- Input
- Get result

3.2 Functional Requirements

Registration and Authentication

The user should be able to create an account and login on the system using it. While logging in, the system should authenticate that it's the right user details.

3.3 External Interface Requirements (If Any)

3.3.1 System Interfaces:

System Interfaces

- Authorize_user();
- preprocessing();
- clustering();
- featureExtraction();
- finalResult();

3.3.2 Hardware Requirements :

Processor - Intel i5 (6core)

Speed - 3.4GHz

RAM - 8GB (Recommended 16GB)

Hard Disk - 50 GB

Key Board - Standard Windows Keyboard

Mouse - Two or Three Button Mouse

Monitor - SVGA

3.3.3 Software Interfaces:

Operating system: - Windows 7/8/10, Linux OS

Coding Language: - Python, React js , Tailwind , CSS, Vite, Javascript , HTML

IDE: - VsCode

3.4 Non-Functional Requirements

3.4.1 Performance Requirements

Performance will depend on uploaded data (Payloads).

3.4.2 Safety Requirements

Validation should be properly provided.

3.4.3 Security Requirements

Account password should follow some standard rules.

3.4.4 Software Quality Attributes

Usability:

The system should be user friendly and self-explanatory. Proposed system is Flexible, Robust, and easily Testable.

Accuracy:

The system will give results related to query.

Openness:

The system should be extensible to guarantee that it is useful for community system.

Usability:

The proposed system will helpful in hospital where users can get trusted and accurate result.

Reliability:

The system should have accurate results and fast responses to users.

3.5 Analysis Models: SDLC Model to be applied:

The Waterfall Model is sequential design process, often used in Software development processes; where progress is seen as flowing steadily download through the phase of conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation and Maintenance. There are 5 Phase of water fall model:

Requirement Gathering and analysis:

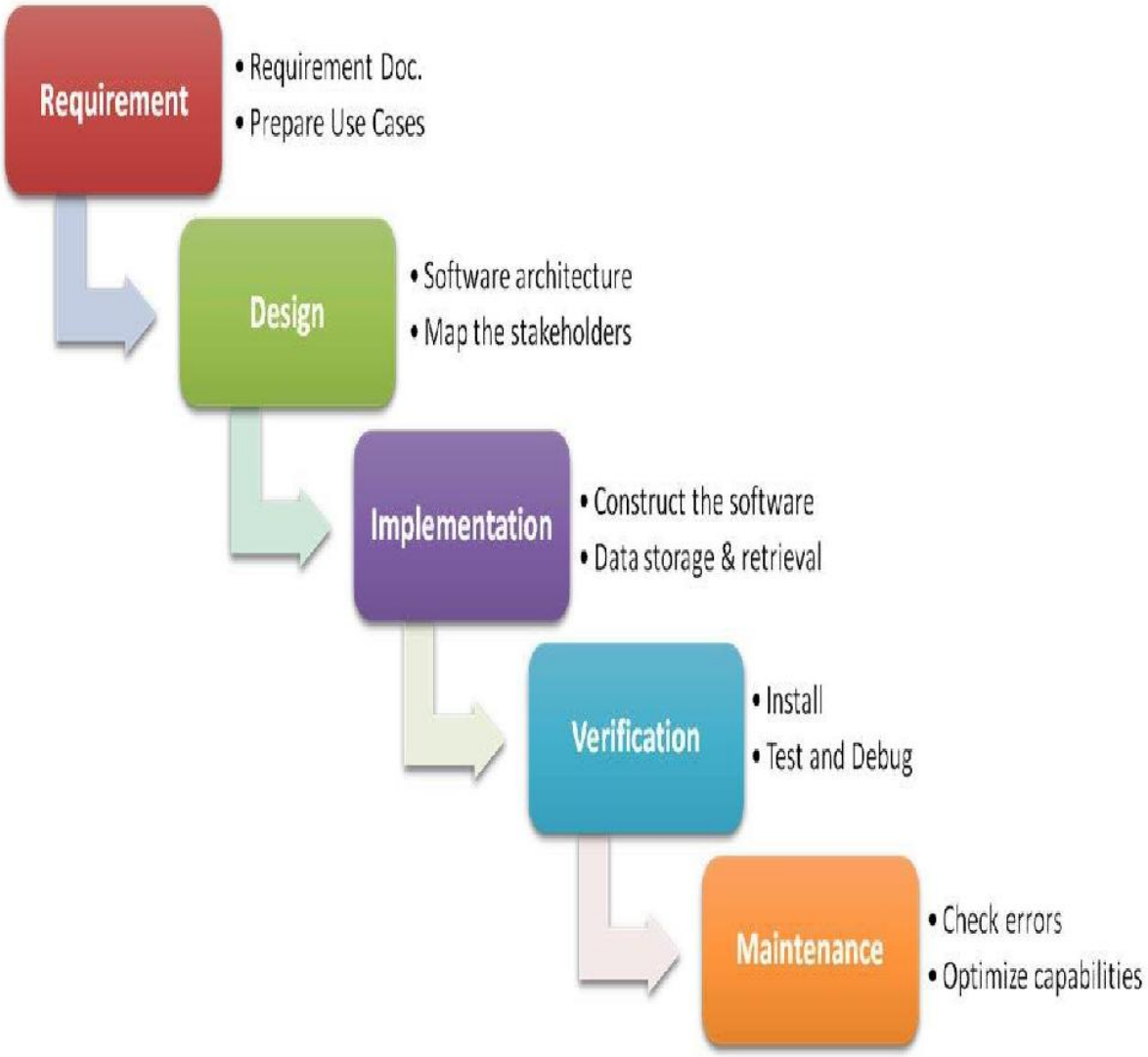
Here requirements are gathered means which kind of dataset we have to use. Then what are functional requirement of system. Document is prepared that use cases are designed. System Design: In this stage we decide which hardware and software we require to design the system.

Implementation: In this stage system is developed according to module wise: Admin module and user module.

Verification: This stage all developed software are installed and they are tested with different way against system requirements.

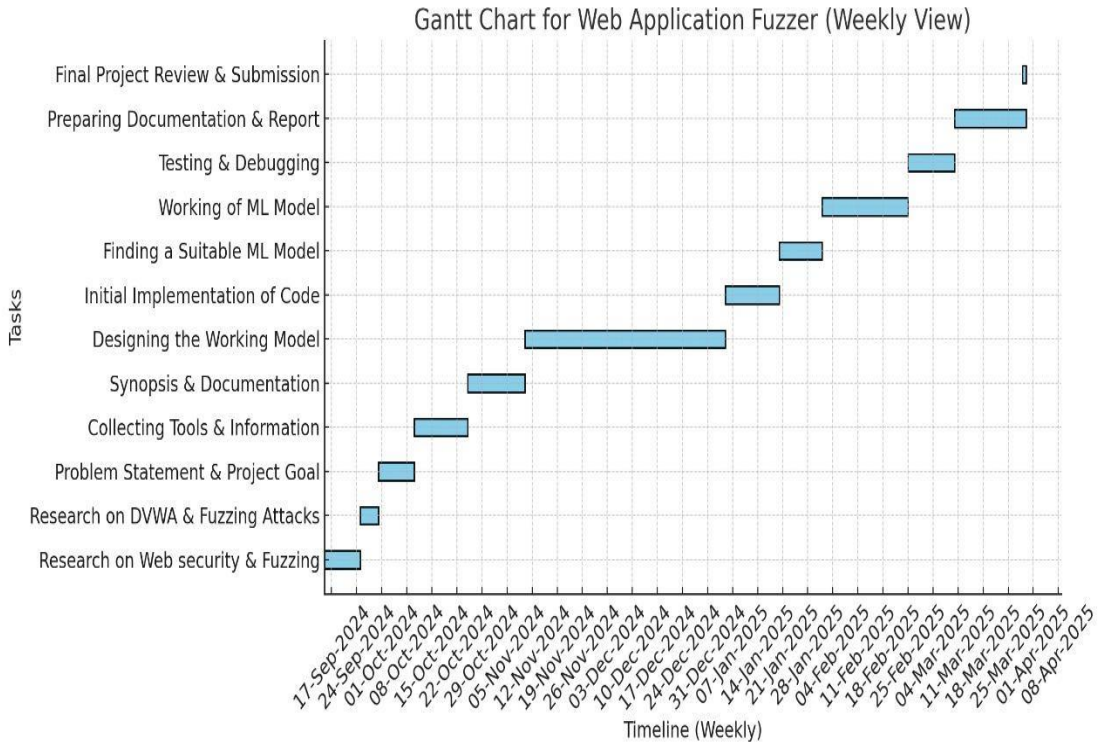
Maintenance: According to software's new version and there use them need to update. In our system some browser are not supportive to our web pages for that we need to use only specific browser.

3.6 Waterfall Model



3.7 System Implementation Plan

Project task set



Schedule of Project

Start Date: - 30-Jul-2024

End Date: - April 2025

Duration: - Approx. 09 Months

Task network

Activity	I week	II week	III week	IV week	V Week	VI week	VII week	VIII week	IX week
	Aug 4	Aug 11	Aug 18	Aug 25	Sept 1	Sept 8	Sept 15	Sept 22	Sept 29
Initiate the project									
Communication									
Literature survey									
Define scope									
Develop SRS									
Plan the project									
Design mathematical model									
Feasibility Analysis									
Develop work breakdown structure									
Planning project schedule									
Design UML and other diagrams									
Design test plan									
Design risk management plan									

Activity	XI week	XII week	XIII week	XIV week	XV week	XVI week	XVII week	XVIII week	XIX week	XX week	XXI week	XXII week
	Jan 5	Jan 15	Jan 19	Jan 26	Feb 2	Feb 9	Feb 16	Feb 23	Mar 2	Mar 9	Mar 16	Mar 23
Execute the project												
Build and test basic functional unit												
Build and test database with login and session maintenance facility												
Build and test Bluetooth mode												
Build and test security features												

4. System Design

4.1 System Architecture

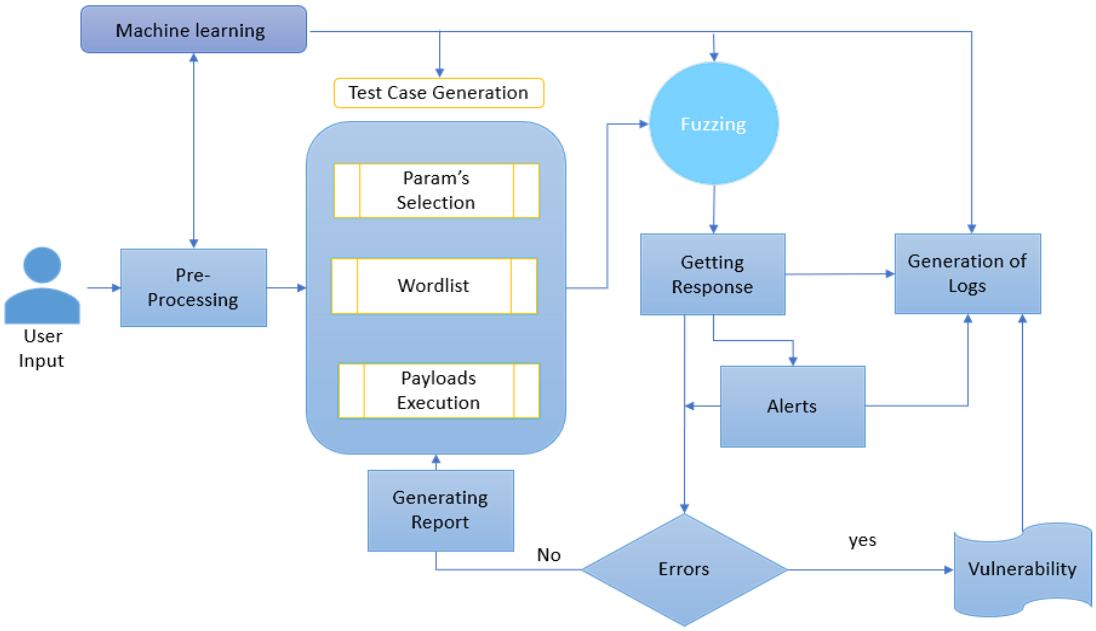


Fig 1: Proposed System Architecture

4.2 Data Flow Diagrams

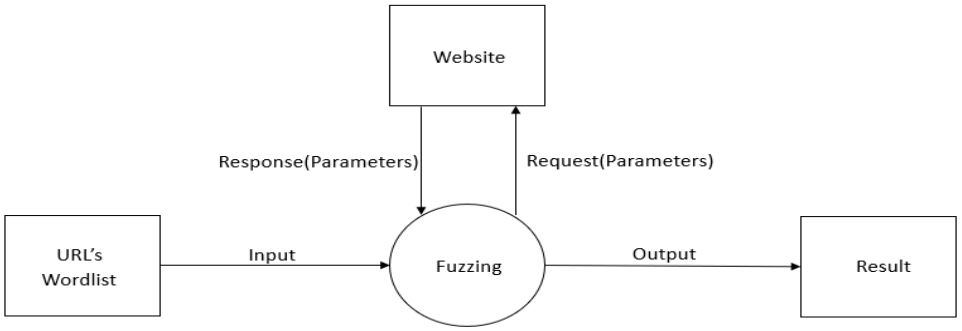


Fig.2. DFD Level 0

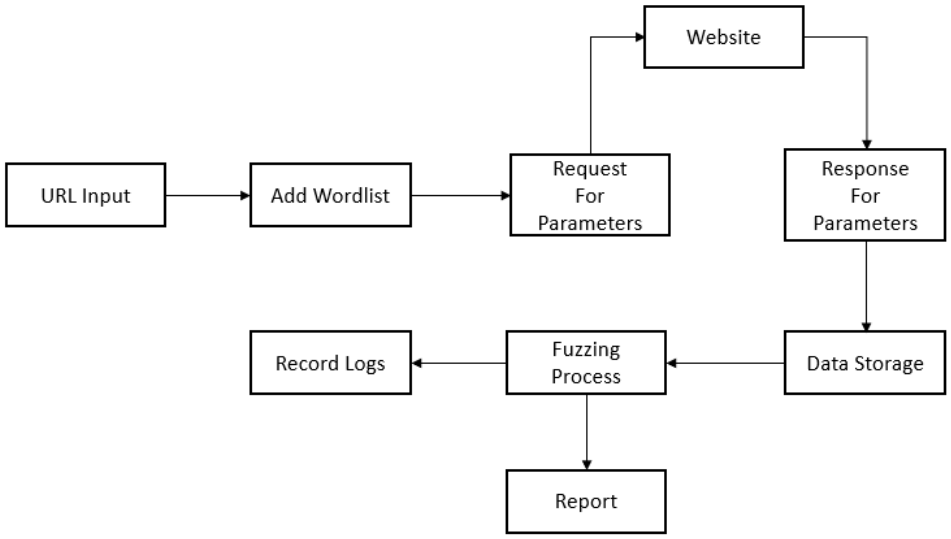


Fig.3. DFD Level 1

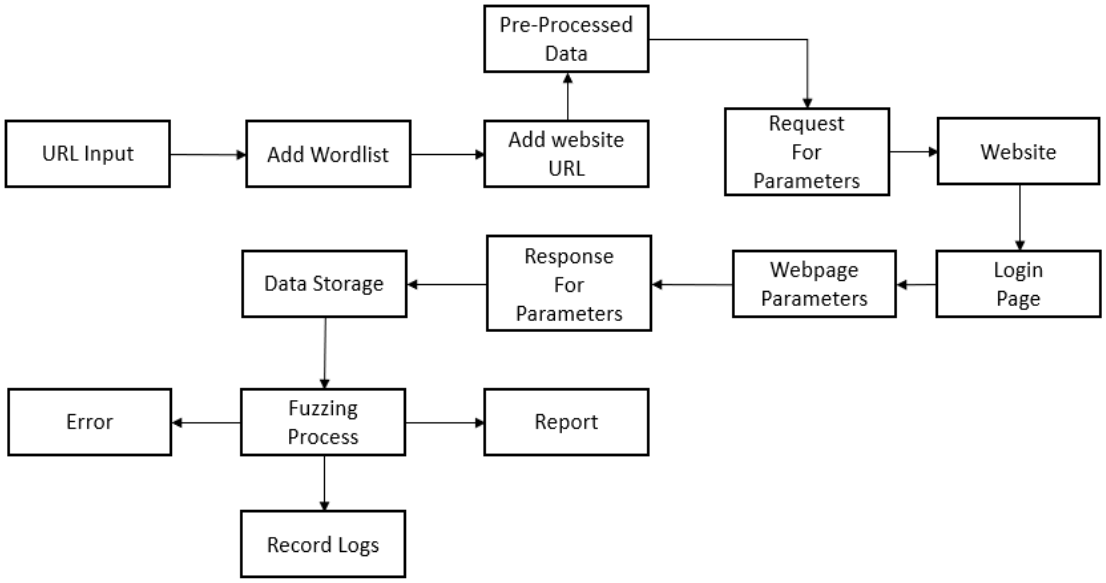


Fig.4. DFD Level 2

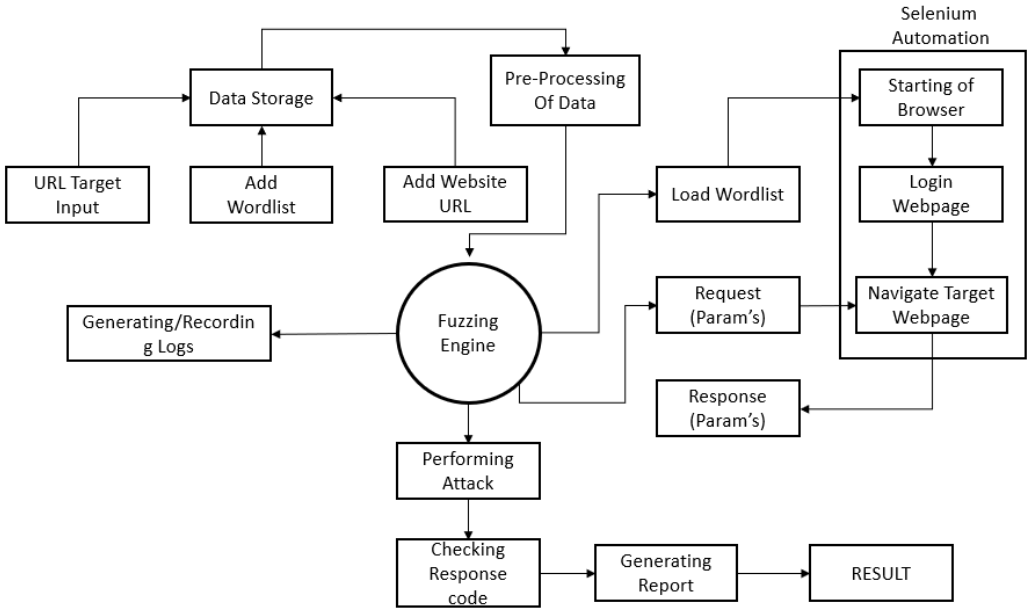


Fig.5. DFD Level 3

4.3 Sequence Diagram

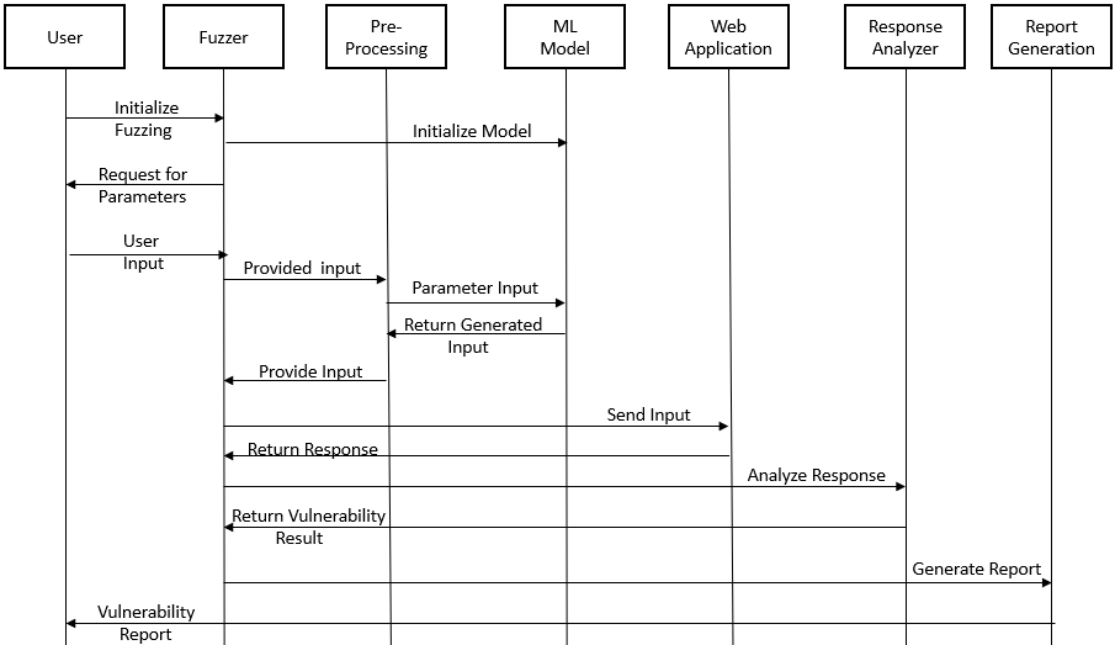


Fig.5. Sequence Diagram

4.4 UML Diagrams

1) Use case Diagram:

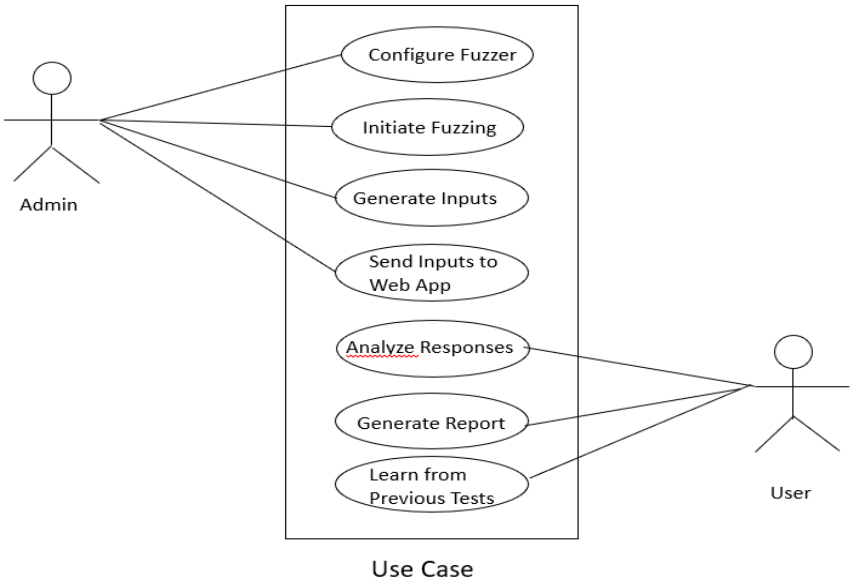


Fig.6. Use Case Diagram

2) Class Diagram:

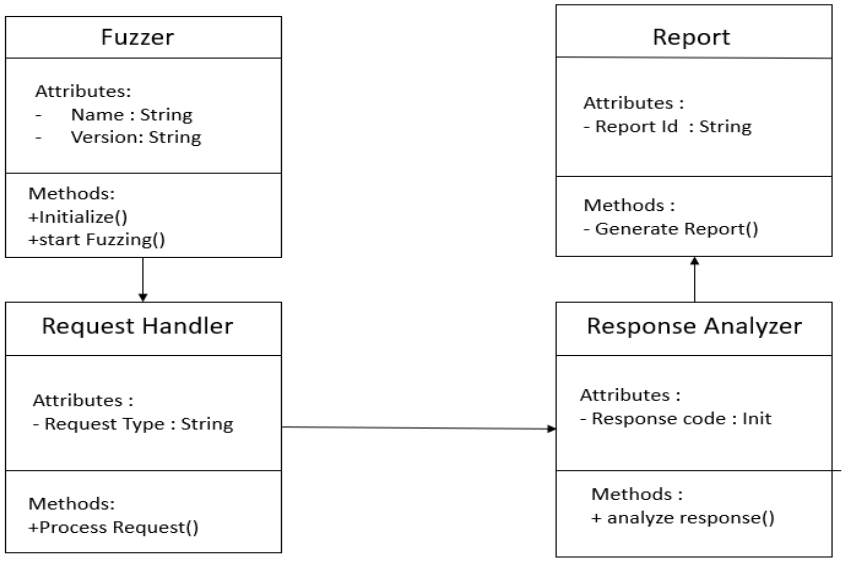


Fig.7. Class Diagram

3) Uml Activity Diagram

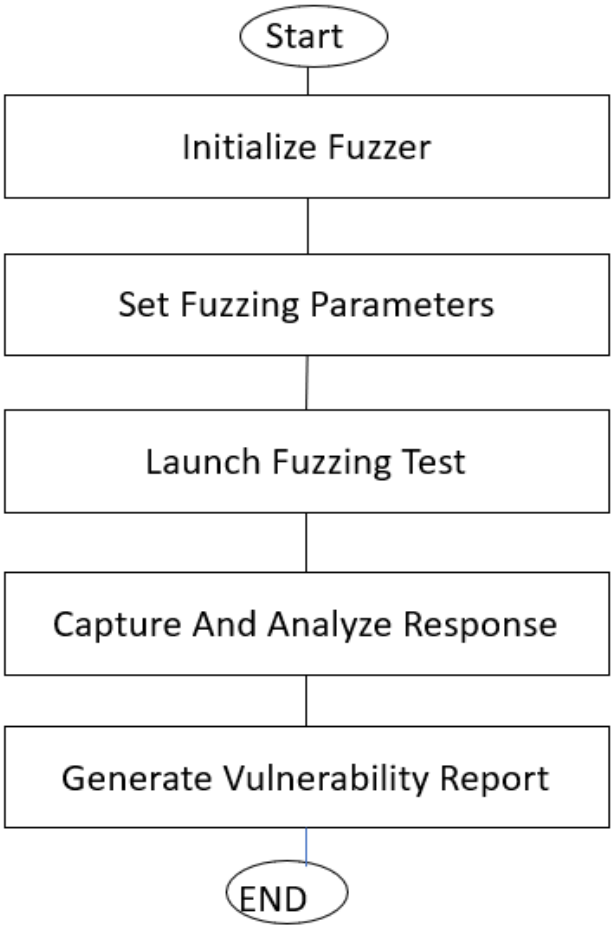


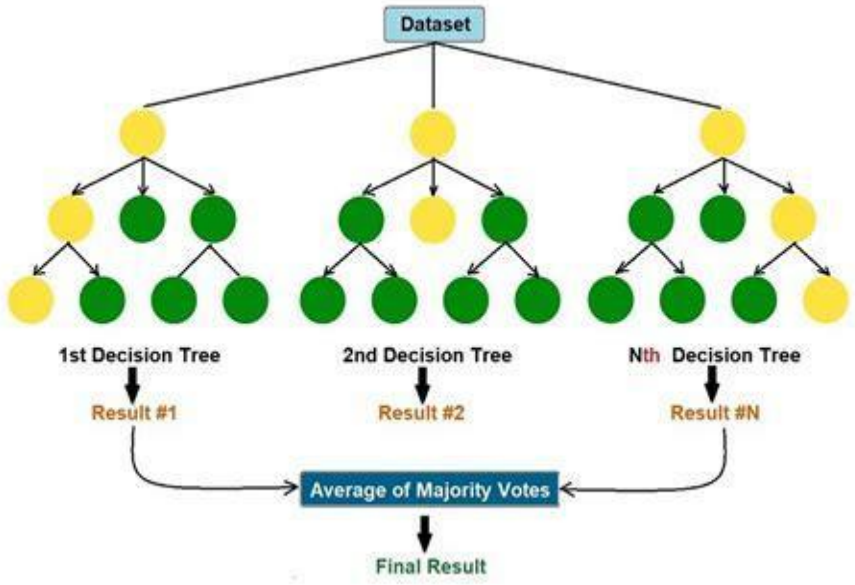
Fig.8. UML Activity Diagram

5. Machine Learning Models

The fuzzer leverages two machine learning models to enhance vulnerability detection: a Random Forest model and an Anomaly Forest. The Random Forest model is a supervised learning technique used to classify web responses as either vulnerable or non-vulnerable. It operates by building multiple decision trees and aggregating their predictions to improve classification accuracy. Trained on labeled datasets, it uses features such as HTTP status codes, response length, reflected payloads, and execution time. This model helps reduce false positives and improves the reliability of detecting known security issues. On the other hand, the Anomaly Forest is an unsupervised learning approach designed to detect zero-day vulnerabilities and irregular response patterns. It learns the normal behavior of web applications and assigns anomaly scores to new responses, flagging those that deviate significantly from the norm. Techniques like Isolation Forests are employed to efficiently isolate suspicious activity. Features considered include status codes, response length, payload reflection, error messages, execution time, and content-type. This model ensures the fuzzer can adapt in real time to new and evolving security threats, making it more robust against unknown attacks.

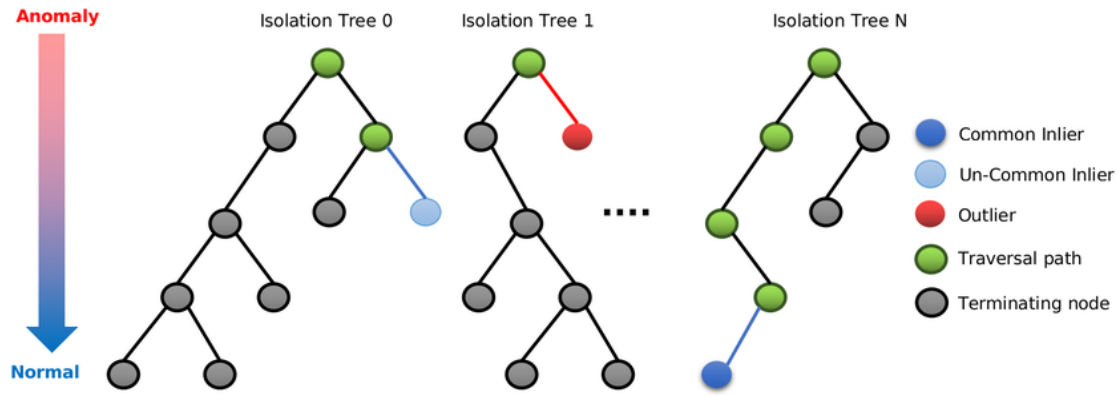
5.1. Random Forest Model

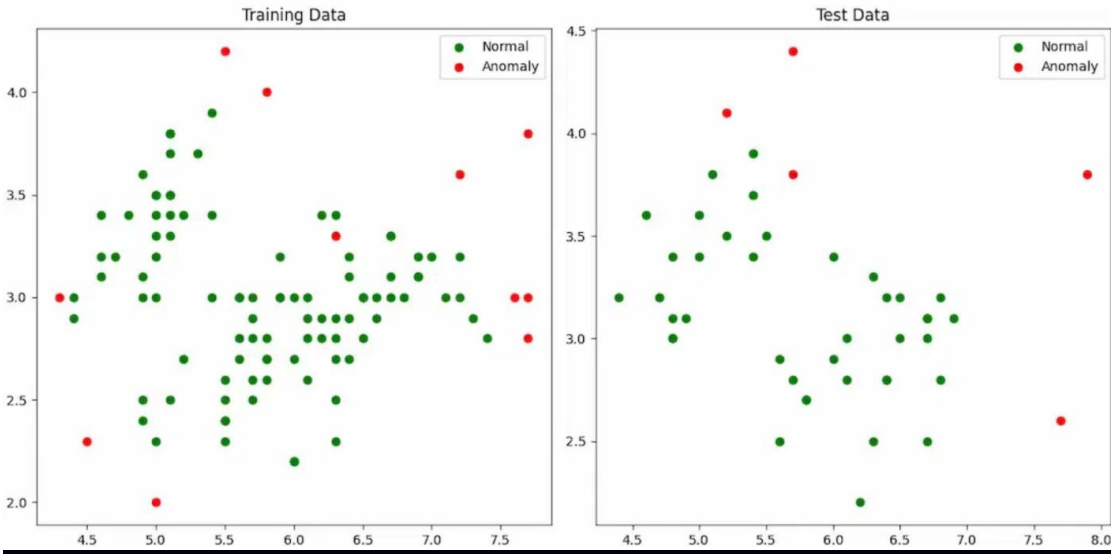
The Random Forest model is used in the fuzzer to classify web responses as vulnerable or non-vulnerable. It is an ensemble learning technique that builds multiple decision trees and aggregates their predictions to improve accuracy. The model is trained on a dataset containing labeled web responses, considering features such as status codes, response length, payload reflection, and execution time**.** It enhances fuzzing efficiency by reducing false positives and improving the detection of known vulnerabilities**.**



5.2. Anomaly Detection (Anomaly Forest)

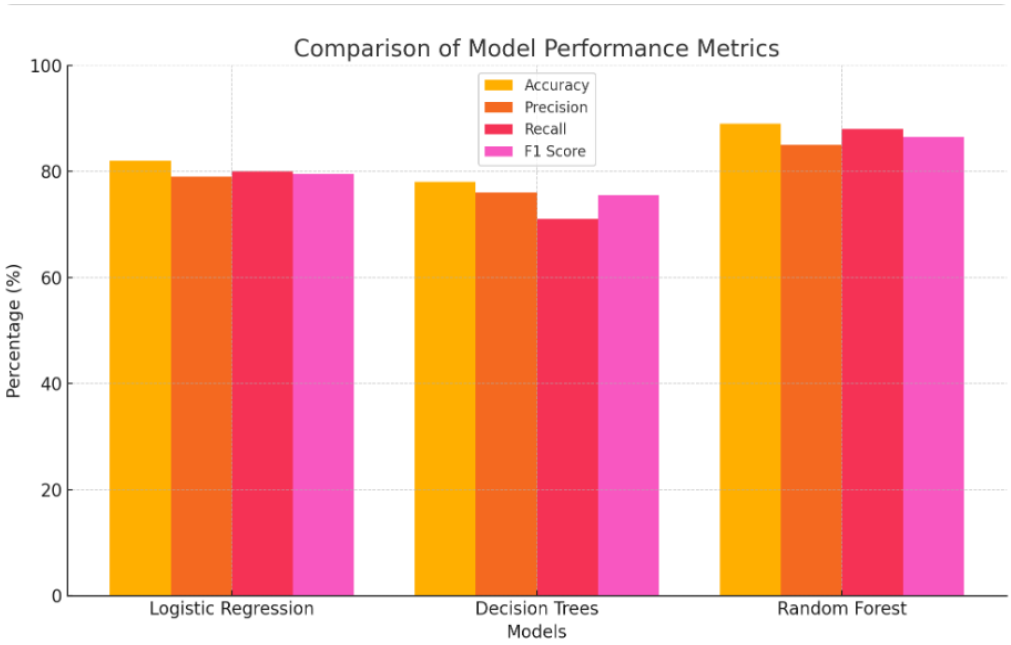
The Anomaly Forest approach, based on unsupervised learning, detects zero-day vulnerabilities and unexpected response patterns. It learns normal web behavior and flags deviations as anomalies. The model assigns an anomaly score to new responses, marking them as potential security risks if they exceed a predefined threshold. Key techniques include Isolation Forests**,** which efficiently isolate suspicious responses. This method ensures real-time adaptation to evolving security threats, making the fuzzer more effective against unknown attacks. Features that are used in Anomaly Detection are HTTP Status Code, Response Length, Reflected Payloads, Error Message Presence, Execution Time, Content-Type





5.3 Model Accuracy Comparison

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	82%	79%	80%	79.5%
Decision Trees	78%	76%	71%	75.5%
Random Forest	89%	85%	88%	86.5%



6. Other Specification

6.1 Advantages:

- Automated Security Testing
- Eliminates the need for manual penetration testing, saving time and effort.
- Scans multiple web applications efficiently without human intervention.
- Machine Learning-Driven Analysis
- Improves accuracy in detecting vulnerabilities by reducing false positives.
- Adapts over time using anomaly detection (Isolation Forest) and classification models (Random Forest).
- Detects Multiple Types of Vulnerabilities
- Identifies XSS, SQL Injection, and other input-based attacks.
- Monitors form fields, URL parameters, and response anomalies dynamically.
- Real-Time Logging & Reporting
- Provides instant feedback on detected threats.
- Logs all fuzzing attempts, helping in security audits and compliance.
- Scalability & Adaptability
- Works on a wide range of web applications with minimal configuration.
- Can be extended with new attack patterns and trained on larger datasets.
- Integration with Existing Security Tools
- Can be used alongside penetration testing frameworks (e.g., Burp Suite, OWASP ZAP).
- Supports automated scanning pipelines for DevSecOps environments.

6.2 Limitations:

- Potential for False Positives/Negatives
- While machine learning improves detection, some anomalies may still be misclassified.
- Requires constant retraining to maintain accuracy.
- Limited to Input-Based Attacks
- Primarily detects injection and input-based vulnerabilities.

- Cannot detect logical security flaws, authentication bypasses, or business logic vulnerabilities.
- High Resource Consumption
- Running Selenium WebDriver, logging mechanisms, and machine learning models may be CPU- and memory-intensive.
- Large-scale fuzzing can slow down web applications being tested.
- Requires Tuning for Optimal Performance
- ML models need feature selection and hyperparameter tuning for best results.
- Threshold settings for anomaly detection need careful adjustments to minimize false alerts.
- Legal & Ethical Concerns
- Unauthorized fuzzing can be seen as an attack if not performed with consent.
- Some web applications block automated requests, limiting the tool's effectiveness.
- Dependency on Web Environment
- Modern JavaScript-heavy applications (e.g., SPAs) may require custom fuzzing techniques.
- CAPTCHAs and WAFs (Web Application Firewalls) can block fuzzing attempts.

6.3 Real World Applications:

1. Web Application Security Testing

Automated Vulnerability Detection:

Identifies SQL Injection (SQLi), Cross-Site Scripting (XSS), Remote Code Execution (RCE) vulnerabilities in web applications.

Used by security teams to conduct regular security assessments without manual effort.

Continuous Security Monitoring:

Integrates with DevSecOps pipelines to ensure secure code deployment.

Can be scheduled to run automated fuzzing scans on production and test environments.

Example: E-commerce platforms (e.g., Amazon, Shopify, Flipkart) use fuzzers to protect payment gateways from injection attacks.

2. Cybersecurity Research & Ethical Hacking

Bug Bounty & Ethical Hacking:

Security researchers and bug bounty hunters use fuzzers to discover security vulnerabilities in web applications.

Helps ethical hackers simulate real-world attack scenarios.

Zero-Day Vulnerability Discovery:

Finds new security flaws in web applications and APIs that traditional scanners may miss.

Helps organizations patch security holes before attackers exploit them.

Example: Google's Project Zero uses fuzzing techniques to identify zero-day vulnerabilities in web browsers and applications.

3. Enterprise Security & Compliance

Regulatory Compliance Testing:

Assists in compliance audits for GDPR, PCI-DSS, HIPAA, ISO 27001, ensuring web applications meet security standards.

Detects data exposure risks in sensitive applications handling user and financial data.

Web Application Firewall (WAF) Testing:

Fuzzing can be used to evaluate WAF effectiveness by testing attack signatures.

Helps fine-tune security rules to prevent false positives and negatives.

Example: Banks & financial institutions (e.g., PayPal, JPMorgan Chase) use fuzzing to secure online banking portals.

4. API Security Testing

Fuzzing REST & GraphQL APIs:

Finds vulnerabilities in API endpoints that handle user authentication, data processing, and financial transactions.

Identifies broken access control, mass assignment, and input validation issues.

IoT & Cloud API Security:

Detects security flaws in IoT communication protocols (MQTT, CoAP) and cloud-based applications (AWS, Azure, Google Cloud).

Example:

Companies like Tesla, AWS, and Google use fuzzing to secure APIs in connected devices and cloud services.

5. Red Team & Blue Team Cyber Defense

Simulating Advanced Persistent Threats (APTs):

Red teams use fuzzing to simulate real-world attacks, testing an organization's security defenses.

Helps blue teams strengthen intrusion detection systems (IDS) and security logs.

Threat Intelligence Integration:

Integrates with SIEM (Security Information and Event Management) tools to detect suspicious traffic patterns.

Enhances incident response & forensic analysis.

Example:

Cybersecurity firms like CrowdStrike, Palo Alto Networks, and FireEye use fuzzing in penetration testing & APT simulations.

6. Government & National Security

Securing Government Websites & Services:

Used by government agencies to protect critical infrastructure (e.g., defense systems, tax portals, national ID services).

Helps detect state-sponsored cyberattacks.

Law Enforcement & Digital Forensics:

Assists in investigating cybercrimes by testing web applications for data breaches and unauthorized access points.

Example: The US Department of Homeland Security (DHS) and NSA use fuzzing for national cybersecurity.

7. Conclusions & Future Work

The Web Application Fuzzer is a powerful tool designed to automate security testing and detect vulnerabilities in web applications. By leveraging machine learning models alongside traditional fuzzing techniques, this project enhances accuracy, efficiency, and adaptability in identifying security threats like SQL Injection (SQLi), Cross-Site Scripting (XSS), and Remote Code Execution (RCE).

Key Achievements

- Automated Security Testing: Reduces manual effort by automating payload injection and response analysis.
- ML-Enhanced Detection: Uses Isolation Forest and Random Forest models to classify threats more effectively.
- Real-Time Analysis & Logging: Provides instant feedback on potential vulnerabilities.
- Scalability & Integration: Can be integrated into DevSecOps pipelines for continuous security assessment.

Future Scope & Improvements

- AI-Driven Fuzzing: Implementing reinforcement learning to optimize attack strategies dynamically.
- Cloud & API Security: Extending fuzzing capabilities to cloud environments and microservices.
- Zero-Day Exploit Detection: Enhancing anomaly detection to uncover unknown security flaws.
- Better False Positive Reduction: Fine-tuning ML models to improve accuracy and precision.

References

1. Assiri, F. Y., & Aljahdali, A. O. (2024). Software Vulnerability Fuzz Testing: A Mutation-Selection Optimization Systematic Review. *Engineering, Technology & Applied Science Research*, 14(4), 14961-14969. https://doi.org/10.48084/etasr.6971

2. Godefroid, P., Peleg, H., & Singh, R. (2017). Learn Fuzz: Machine Learning for Input Fuzzing. *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 50-59. https://doi.org/10.1109/ASE.2017.8115618

3. Pham, V. T., et al. (2019). Smart Greybox Fuzzing. *IEEE Transactions on Software Engineering*, 47(9), 1980-1997. https://doi.org/10.1109/TSE.2019.2941681

4. Rawat, S., Jain, V., Kumar, A., Cojocar, L., Giuffrida, C., & Bos, H. (2017). VUzzer: Application-aware Evolutionary Fuzzing. *NDSS Symposium*, 1-14. https://doi.org/10.14722/ndss.2017.23404

5. She, D., Pei, K., Epstein, D., Yang, J., Ray, B., & Jana, S. (2019). NEUZZ: Efficient Fuzzing with Neural Program Smoothing. *IEEE Symposium on Security and Privacy (SP)*, 803-817. https://doi.org/10.1109/SP.2019.00052

6.Chen, P., & Chen, H. (2018). Angora: Efficient fuzzing by principled search. IEEE Symposium on Security and Privacy (SP), 711-725. https://doi.org/10.1109/SP.2018.00046.

7.Zhao, D. (2020). Fuzzing Technique in Web Applications and Beyond. Journal of Physics: Conference Series, 1678(1), 012109. https://doi.org/10.1088/1742-6596/1678/1/012109:contentReference[oaicite:0]{index=0}

9. Research Paper & certificates

1. Paper 1 (Research paper):

Web Application Fuzzer Using Machine Learning.

Siddhant Mane, Lokesh Pusdekar, Shubham Khaire, Pramod Kedar

Prof. Ashwini Taksal Information Technology.

Abstract: Fuzz testing is a technique for finding software vulnerabilities by feeding programs with unexpected or unusual data. As software becomes more complex and widespread, traditional fuzz testing struggles with issues like incomplete coverage of program behaviors, limited automation, and insufficient test cases. Machine learning (ML) offers promising ways to improve fuzz testing due to its ability to analyze data and make predictions. This paper reviews recent advancements in fuzz testing and explores how ML can enhance the process. It explains how ML can optimize various stages, such as preparing and filtering data before testing, generating diverse and relevant test cases, selecting the best inputs to increase coverage, and analyzing results to identify important patterns. The discussion also covers how ML can boost fuzz testing by refining the processes for modifying, generating, and filtering test cases, while comparing different technical approaches. It highlights the benefits, including better coverage, improved vulnerability detection, and more efficient testing. Finally, it addresses the challenges in integrating ML with fuzz testing and suggests future research directions in this area.

Index Terms **- Fuzzing Techniques in web applications, Machine Learning, Selenium Automation in Web fuzzer, Mutation in Fuzzer..**

I.INTRODUCTION

In recent years, network attacks and the number of vulnerabilities has rapidly increased, posing risks such as data leaks or loss. Techniques for finding and fixing vulnerabilities are crucial for reducing security threats and keeping networks secure. Fuzz testing is a widely-used method for discovering vulnerabilities. It works by automatically or semi-automatically generating test cases to identify program weaknesses, monitoring how programs respond, and using feedback to improve the test cases.[1] This approach is easy to deploy and can be used in various situations .The idea of fuzz testing was first introduced by Miller in 1990 with a tool called Fuzz, which tested software by feeding it unexpected data to see how it reacted.[2][5]

Since then, different types of fuzzers—like black-box, white-box, and gray-box fuzzers—have been developed to make the process more effective. Many researchers have worked to improve fuzz testing, enhancing its ability to find issues and cover more program behaviors. Still, traditional fuzz testing has its limitations, including a shortage of test cases, low success in finding vulnerabilities, and a lack of effective prioritization in choosing test cases. Machine learning (ML), with its strengths in data analysis, language processing, and pattern recognition, has shown potential for boosting cybersecurity efforts like detecting malware and identifying intrusions. Researchers are now exploring how ML can be used to improve fuzz testing by learning patterns and rules from large sets of data, making the testing process smarter and more effective. [2]

II. Literature Survey

Web application security has become a critical concern in the modern digital era due to the increasing complexity of web technologies and the rise of cyber threats. Web application fuzzing is a dynamic security testing approach aimed at identifying vulnerabilities by injecting automatically generated or mutated inputs into web applications. The goal is to expose security flaws such as SQL injection, cross-site scripting (XSS), and remote code execution (RCE).

2.1 Overview of Fuzzing :

A Basic flow of fuzzing : Fuzzing is a process used to find software vulnerabilities by creating many invalid or unexpected inputs, running the program with these inputs, monitoring its behavior, and recording any unusual activity, such as crashes. The goal is to identify and analyze the causes of these issues to detect vulnerabilities. The fuzzing process can be broken down into six main steps: preprocessing, generating test cases, selecting inputs, executing tests, evaluating the results, and analyzing the findings. [6] Fuzzing can be divided into different types based on how much information is available about the program. Black-box fuzzing does not use any internal details and relies only on observing the program’s outputs to find problems. It is quick and easy to use but may miss many issues.[2] On the other hand, white-box fuzzing uses detailed information about the program's inner workings to generate high-quality test cases, resulting in better detection of problems but at the cost of more time and resources.[4] Gray-box fuzzing sits between these two approaches, using some internal information to guide testing while aiming for a balance between effectiveness and resource use. There are also different approaches to creating test cases. Generation-based fuzzing creates new inputs based on known data formats or protocols, while mutation-based fuzzing modifies existing test cases using techniques like flipping bits, changing bytes, or combining data. Mutation-based approaches can quickly produce many test cases, though not all may be useful. The introduction of machine learning helps guide these mutations, making them more effective. A combined approach uses both generation and mutation techniques to produce a wider range of test cases, maximizing the benefits of each method for different testing scenarios. [4].

III. Fuzzing Techniques in Web Applications:

In this section, we will discuss some fuzzing techniques used specifically for web applications. Unlike traditional fuzzers, which often generate random inputs or modify existing data, web fuzzers typically use dictionaries of known values. This dictionary may be partially created through mutation, but it mainly relies on accumulated knowledge and experience, making the fuzzing process more effective. The techniques we'll cover include multi-test fuzzing and fuzzing for specific vulnerabilities. [3]

Multi-test Fuzzing General-purpose fuzzers can detect various types of vulnerabilities and scan for vulnerable paths or logic errors. This broad approach allows them to identify many different problems .One example of a multi-test fuzzer is Wfuzz, which helps find sensitive directories and vulnerabilities like SQL injection, XSS (Cross-Site Scripting), and XXE (XML External Entity). Users can specify a target URL and include the keyword "FUZZ" where they want the fuzzer to insert payloads or random data .Another tool, Burp Suite, is a comprehensive web application attack tool that includes a fuzzing component called Intruder. Users can select where to apply the fuzzing by using special placeholders like $target$, and then choose from a variety of provided payloads.[6]

Specific Vulnerability Fuzzing Fuzzers focused on specific vulnerabilities may cover a narrower range but allow for deeper detection. These tools typically use different dictionaries and strategies tailored to the specific vulnerability being tested. By applying targeted fuzzing strategies, the chances of finding vulnerabilities are increased, and there's a better understanding of the specific issues.[6]

❑ Block Diagram

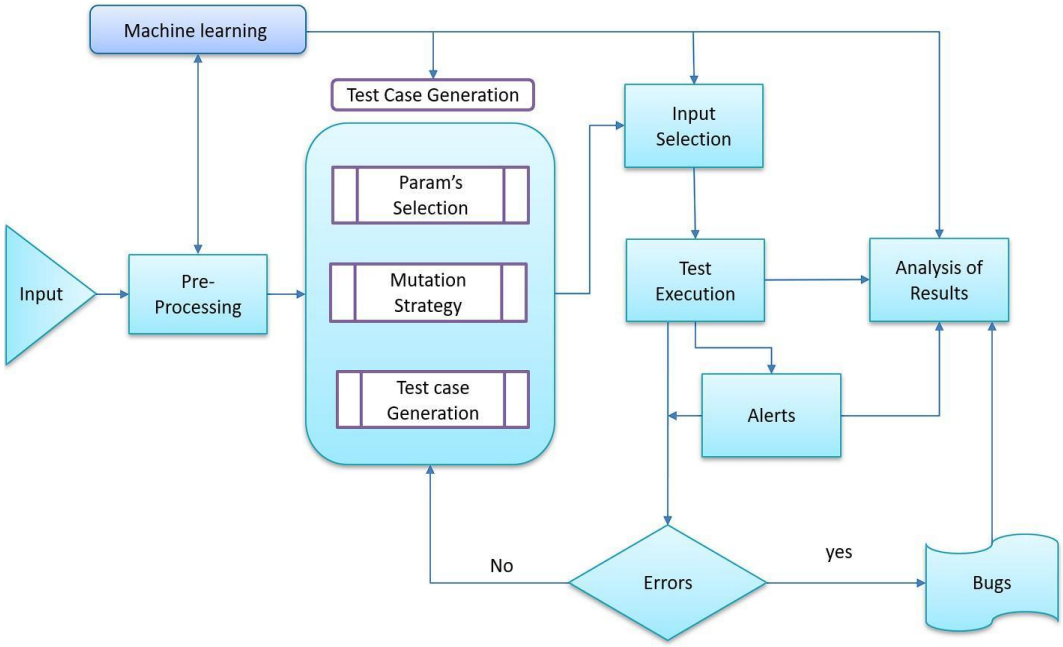


Fig. Block Diagram

3.1.Cross-Site Scripting (XSS)

XSS is one of the most common attacks on web applications and can have widespread effects. In an XSS attack, an attacker injects JavaScript code into web forms or links to manipulate web pages and steal information, such as user cookies. There are three main types of XSS: reflected XSS, stored XSS, and DOM-based XSS. When testing for XSS vulnerabilities, security experts use various JavaScript payloads and continually refine their attack methods for better detection efficiency.[3] Techniques like genetic algorithms may be employed to create better payloads. A widely used tool for detecting XSS vulnerabilities is XSSer, developed by OWASP.

3.2. SQL Injection

SQL Injection is a significant threat to web applications. Here, attackers insert SQL commands into database queries to access sensitive information. SQL injection can be categorized based on whether errors occur or how the database responds over time. Testing for SQL injection often involves using different payloads tailored to various SQL injection types. Security experts optimize these payloads using mutation methods, such as changing values or statements. SQL map is an open-source penetration testing tool that automates the detection and exploitation of SQL injection vulnerabilities and can take control of database servers. Users provide a target and select the type and level of testing .Overall, these fuzzing techniques are essential for identifying and addressing vulnerabilities in web applications, helping to enhance their security.[7]

IV . Machine Learning:

Integrating machine learning into a web application fuzzer involves adding models that can help detect unusual behavior and identify effective payloads more efficiently. Two useful techniques in this context are Isolation Forest for anomaly detection and Random Forest for classifying successful payloads. These models can enhance the fuzzer's ability to detect vulnerabilities by learning from the patterns in the responses it receives.

The first model, Isolation Forest, is used for anomaly detection. After each payload is injected into the application, the fuzzer receives a response, which can sometimes indicate a problem if the application behaves unexpectedly. For example, application crashes or responds with an unusual status code, this could be a sign of a vulnerability being triggered. The Isolation Forest model is trained on "benign" responses—responses that occur under normal, non-malicious conditions. Once trained, the model can then flag any deviations from these normal responses as potential anomalies. This allows the fuzzer to prioritize tests based on pages that show unexpected behaviors after payloads are injected.[5]

The second model, Random Forest, is used to classify payloads based on their effectiveness. This model helps the fuzzer decide which payloads are more likely to exploit a vulnerability. After each payload is injected and a response is received, the fuzzer collects features such as the response code, word count, or any other relevant metric. The Random Forest model uses these features to predict whether a given payload is likely to succeed in causing a vulnerability or not. This allows the fuzzer to focus on testing more promising payloads rather than wasting time on those that are unlikely to lead to meaningful results.

To implement these models in a fuzzer, you first need to collect data for training. For anomaly detection, you would gather a variety of benign responses from the web application to teach the Isolation Forest how normal behavior looks.[1] For the Random Forest model, you would gather data that labels payloads as either successful or unsuccessful based on whether they triggered a vulnerability. Using a machine learning library like scikit-learn, you can then train these models on the collected data.

Once the models are trained, they can be incorporated into the fuzzer’s workflow. For each payload injection, the fuzzer can use the Isolation Forest to check if the response is anomalous, and then use the Random Forest to determine the likelihood that the payload will exploit a vulnerability. By doing this, the fuzzer becomes more intelligent and efficient, focusing its efforts on the most promising payloads and potential vulnerabilities.

In summary, machine learning models like Isolation Forest and Random Forest can significantly improve the performance of a web application fuzzer by enabling it to identify anomalies in responses and classify payloads based on their likelihood of success. By training these models on relevant data and integrating them into the fuzzer, you can make the fuzzing process smarter and more effective in discovering vulnerabilities.

v. Selenium Automation in Web fuzzer.

Incorporating Selenium with mutation-based fuzzing in a web application fuzzer enhances the ability to test application vulnerabilities through dynamic interaction. Selenium automates the testing of user-driven actions, such as form submissions and link navigation, which makes it well-suited for simulating real-world usage patterns within the application.[3] By automating these interactions, testers can repeatedly inject mutated inputs across various parts of the web application, allowing for the efficient

detection of flaws that may arise from improperly handled user inputs. The flexibility of Selenium's WebDriver API makes it possible to test multiple types of browsers and platforms, supporting the identification of cross-platform vulnerabilities.

Mutation-based fuzzing introduces variability to input data, enabling the fuzzer to explore a wide range of potential error states. With Selenium handling the automation of form submissions, button clicks, and other interactions, mutated inputs like SQL injection patterns, cross-site scripting (XSS) payloads, and buffer overflow strings can be tested seamlessly. This approach helps uncover weaknesses in both front-end input validation and backend data handling. Overall, using Selenium for mutation-based fuzzing allows researchers to evaluate how well applications handle unexpected inputs, making it an invaluable tool for improving application resilience.[5]
Using Selenium in a fuzzing framework allows for automated exploration of user-accessible components in web applications, simulating real user actions like typing, clicking, and form submission across different input fields. This level of automation is particularly useful when testing for security vulnerabilities, as it can inject various types of malicious payloads to assess how the application responds to potentially unsafe inputs. With mutation-based fuzzing, Selenium can iterate through a wide range of input variations, such as SQL injections, XSS attacks, and directory traversal patterns, enabling the discovery of vulnerabilities in input handling that might not be immediately visible in manual testing. This helps identify areas where input sanitization may be inadequate, leading to potential security breaches.

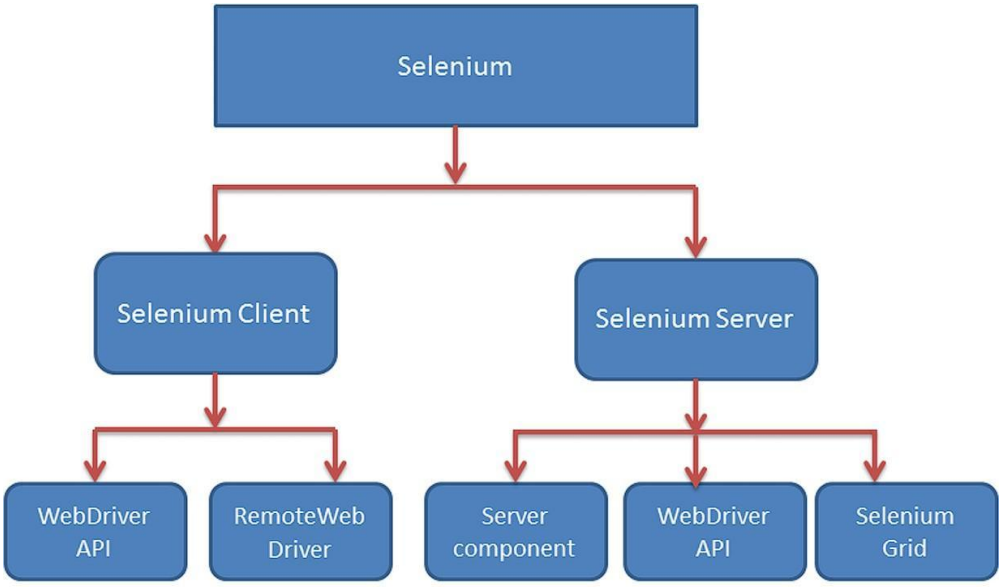


Fig : Selenium Architecture

In addition, Selenium's compatibility with various browsers and its capability to handle asynchronous loading and JavaScript make it ideal for dynamic applications where elements might appear or change after page load. [1]This adaptability enables thorough testing of modern applications that rely heavily on client-side rendering and interactivity. Mutation-based fuzzing with Selenium not only exposes vulnerabilities but also simulates the complex sequences of user actions that a real attacker might take

advantage of. Integrating Selenium and mutation-based fuzzing in web application security testing offers a proactive approach to vulnerability detection, enhancing both coverage and depth of testing.[6] By automatically interacting with the application, Selenium executes input variations that uncover subtle issues, such as unhandled exceptions, unexpected redirects, and error disclosures. These insights are critical for pinpointing weak spots in the application's defense mechanisms,
especially against complex attacks like injection flaws and cross-site scripting.

Moreover, the combination of Selenium's automation with diverse mutation payloads allows the fuzzer to simulate a variety of threat scenarios under realistic conditions, increasing the likelihood of identifying vulnerabilities before they can be exploited in a production environment. As a result, this approach strengthens the security of web applications by providing continuous, scalable, and comprehensive testing across dynamic, user-driven workflows

VI. Mutation in Fuzzer

Mutation-based fuzzing is a technique that generates test cases by modifying existing inputs to uncover software vulnerabilities. This approach begins with a set of "seed" inputs, which are then altered in various ways—such as bit flipping, character insertion, or adding special characters—to create new test cases. By using mutations, fuzzers can explore a wide range of input possibilities without needing prior knowledge of the software's structure or specifications, making mutation-based fuzzing highly adaptable across applications and protocols.[3][4] Mutation-based fuzzing has been effective in testing different software layers, from application logic to network protocols, by exposing unexpected behaviors and errors when inputs are slightly altered(ETASR_6971). In recent years, research on optimizing mutation-based fuzzing has highlighted the importance of selecting efficient mutation operators to improve testing outcomes. Traditional mutation-based fuzzers like American Fuzzy Lop (AFL) randomly choose mutation operators, but optimization-focused studies have proposed methods to prioritize mutations that lead to higher code coverage and more unique vulnerabilities. Optimization techniques, including machine learning models and reinforcement learning, guide the fuzzer to apply specific mutations that are likely to yield new or interesting
execution paths, reducing redundancy and enhancing the efficacy of the fuzzing process. These advancements have shown significant improvements in identifying security flaws, offering a more targeted approach that maximizes resources during testing (ETASR_6971)[3].

Mutation-based fuzzing also benefits from adaptive feedback mechanisms that continuously refine the mutation strategy based on real-time feedback from the system under test. For instance, by analyzing coverage metrics and runtime behavior, the fuzzer can learn which mutations have successfully triggered new paths and focus future efforts on similar operations. This feedback loop enables the fuzzer to navigate complex input spaces more efficiently, adapting its mutation strategy to achieve higher coverage and better vulnerability detection rates. By implementing these optimizations, mutation-based fuzzing not only speeds up the testing process but also increases the overall reliability and security of software applications(ETASR)

❑Sequence Diagram

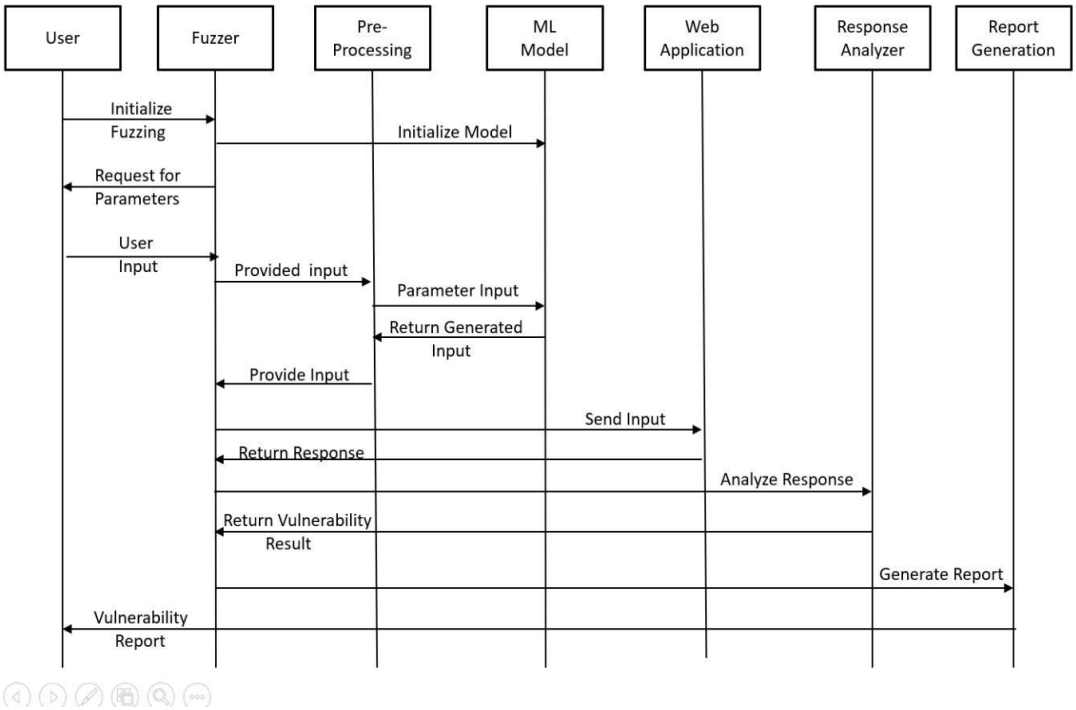


Fig : Sequence Diagram

IV. RESULTS AND DISCUSSION

This research paper looks at fuzzing techniques used in web applications and how effective they are in finding vulnerabilities. Multi- test fuzzing tools like Wfuzz and Burp Suite have proven capable of detecting a range of vulnerabilities, including SQL injection and XSS. By using dictionaries filled with known data, these tools create focused test cases that can uncover various issues, such as logic errors.[3] However, while they offer versatility, they might not provide deep insights into specific vulnerabilities.

On the other hand, fuzzers aimed at specific vulnerabilities allow for more thorough testing. Tools like XSSer and SQL map optimize their methods to improve detection rates for threats like XSS and SQL injection. They adapt

their payloads based on various scenarios, which increases their chances of finding hidden issues .Despite these advancements, fuzzing techniques still face challenges. Traditional methods often struggle with incomplete logic coverage, which can lead to missed vulnerabilities. Additionally, specific vulnerability fuzzers may overlook problems that fall outside their targeted focus.

REFERENCES

1. Assiri, F. Y., & Aljahdali, A. O. (2024). Software Vulnerability Fuzz Testing: A Mutation-Selection Optimization Systematic Review. *Engineering, Technology & Applied Science Research*, 14(4), 14961-14969. https://doi.org/10.48084/etasr.6971

2. Godefroid, P., Peleg, H., & Singh, R. (2017). Learn Fuzz: Machine Learning for Input Fuzzing. *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 50-59. https://doi.org/10.1109/ASE.2017.8115618

3. Pham, V. T., et al. (2019). Smart Greybox Fuzzing. *IEEE Transactions on Software Engineering*, 47(9), 1980-1997. https://doi.org/10.1109/TSE.2019.2941681

4. Rawat, S., Jain, V., Kumar, A., Cojocar, L., Giuffrida, C., & Bos, H. (2017). VUzzer: Application-aware Evolutionary Fuzzing. *NDSS Symposium*, 1-14. https://doi.org/10.14722/ndss.2017.23404

5. She, D., Pei, K., Epstein, D., Yang, J., Ray, B., & Jana, S. (2019). NEUZZ: Efficient Fuzzing with Neural Program Smoothing. *IEEE Symposium on Security and Privacy (SP)*, 803-817. https://doi.org/10.1109/SP.2019.00052

6. Chen, P., & Chen, H. (2018). Angora: Efficient fuzzing by principled search. *IEEE Symposium on Security and Privacy (SP)*, 711- 725. https://doi.org/10.1109/SP.2018.00046.

7. Zhao, D. (2020). Fuzzing Technique in Web Applications and Beyond. *Journal of Physics: Conference Series*, 1678(1), 012109. https://doi.org/10.1088/1742-6596/1678/1/012109:contentReference[oaicite:0]{index=0}.

1. Certificate of Research Paper:


Certificate of Publication
INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR) | IJRAR.ORG
An International Open Access, Peer-reviewed, Refereed Journal
E-ISSN: 2348-1269, P-ISSN: 2349-5138
The Board of
International Journal of Research and Analytical Reviews (IJRAR)
Is hereby awarding this certificate to
Pramod Kedar
In recognition of the publication of the paper entitled
Web Application Fuzzer using Machine Learning
Published In IJRAR (www.ijrar.org) UGC Approved (Journal No : 43602) & 7.17 Impact Factor
Volume 12 Issue 1 February 2025, Date of Publication: 06-February-2025
PAPER ID : IJRAR25A1865
Registration ID : 305141
R.B.Joshi
EDITOR IN CHIEF
UGC and ISSN Approved - Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.17 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal
INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS | IJRAR
An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal
Website: www.ijrar.org | Email: editor@ijrar.org | ESTD: 2014
Manage By: IJPUBLICATION Website: www.ijrar.org | Email ID: editor@ijrar.org
IJRAR | E-ISSN: 2348-1269, P-ISSN: 2349-5138


Certificate of Publication
INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR) | IJRAR.ORG
An International Open Access, Peer-reviewed, Refereed Journal
E-ISSN: 2348-1269, P-ISSN: 2349-5138
The Board of
International Journal of Research and Analytical Reviews (IJRAR)
Is hereby awarding this certificate to
Lokesh Pusdekar
In recognition of the publication of the paper entitled
Web Application Fuzzer using Machine Learning
Published In IJRAR (www.ijrar.org) UGC Approved (Journal No : 43602) & 7.17 Impact Factor
Volume 12 Issue 1 February 2025, Date of Publication: 06-February-2025
PAPER ID : IJRAR25A1865
Registration ID : 305141
R.B.Joshi
EDITOR IN CHIEF
UGC and ISSN Approved - Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.17 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal
INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS | IJRAR
An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal
Website: www.ijrar.org | Email: editor@ijrar.org | ESTD: 2014
Manage By: IJPUBLICATION Website: www.ijrar.org | Email ID: editor@ijrar.org
IJRAR | E-ISSN: 2348-1269, P-ISSN: 2349-5138

Certificate of Publication

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR) | IJRAR.ORG

An International Open Access, Peer-reviewed, Refereed Journal

E-ISSN: 2348-1269, P-ISSN: 2349-5138

The Board of

International Journal of Research and Analytical Reviews (IJRAR)

Is hereby awarding this certificate to

Siddhant Mane

In recognition of the publication of the paper entitled

Web Application Fuzzer using Machine Learning

Published in IJRAR (www.ijrar.org) UGC Approved - Journal No : 43602 & 7.17 Impact Factor

Volume 12 Issue 1 February 2025, Date of Publication: 06-February-2025

PAPER ID : IJRAR25A1865

Registration ID : 305141

R.B.Joshi

EDITOR IN CHIEF

UGC and ISSN Approved - Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.17 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS | IJRAR

An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal

Website: www.ijrar.org | Email: editor@ijrar.org | ESTD: 2014

Manage By: IJPUBLICATION Website: www.ijrar.org | Email ID: editor@ijrar.org

IJRAR | E-ISSN: 2348-1269, P-ISSN: 2349-5138

2. Paper (Implementation paper):

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Web Application Fuzzer

Shubham Khaire, Siddhant Mane, Lokesh Pusdekar, Pramod Kedar

Prof. Pranita Ingale

Dept. of Information Technology, Bhivarabai Sawant Institute Of Technology And Research, Wagholi, Pune, India

ABSTRACT: Fuzz testing is a technique for finding software vulnerabilities by feeding programs with unexpected or unusual data. As software becomes more complex and widespread, traditional fuzz testing struggles with issues like incomplete coverage of program behaviors, limited automation, and insufficient test cases. Machine learning (ML) offers promising ways to improve fuzz testing due to its ability to analyze data and make predictions. This paper reviews recent advancements in fuzz testing and explores how ML can enhance the process. It explains how ML can optimize various stages, such as preparing and filtering data before testing, generating diverse and relevant test cases, selecting the best inputs to increase coverage, and analyzing results to identify important patterns. The discussion also covers how ML can boost fuzz testing by refining the processes for modifying, generating, and filtering test cases, while comparing different technical approaches. It highlights the benefits, including better coverage, improved vulnerability detection, and more efficient testing. Finally, it addresses the challenges in integrating ML with fuzz testing and suggests future research directions in this area.

Key Words - Fuzzing Techniques in web applications, Machine Learning, Selenium Automation in Web fuzzer, Mutation in Fuzzer..

I.INTRODUCTION

In recent years, the rise in network attacks and system vulnerabilities has increased significantly, posing threats such as data leaks and losses. Techniques to identify and fix vulnerabilities are essential for reducing security risks and keeping systems protected. Fuzz testing is a widely-used approach for uncovering vulnerabilities. It works by automatically or semi- automatically generating test cases, observing how software responds, and using feedback to enhance future inputs. This method is easy to implement and applicable in many contexts.The concept of fuzz testing was introduced by Miller in 1990 through a tool called Fuzz, which tested programs by supplying unexpected inputs to observe their behavior. Since then, several types of fuzzers have emerged—such as black-box, white-box, and gray-box fuzzers—each designed to

improve the discovery process and broaden behavior coverage.Researchers have continuously worked on refining fuzzing techniques to increase their ability to detect flaws more efficiently.However, traditional fuzzing has its challenges, such as a limited number of test cases, reduced effectiveness in finding bugs, and lack of intelligent prioritization when selecting inputs. To address these issues, researchers are exploring the integration of machine learning (ML). Known for its capabilities in analyzing data, recognizing patterns, and processing language, ML has shown promise in enhancing cybersecurity tasks like malware detection and intrusion identification. Work is ongoing to apply ML to improve fuzzing methods further.

II. LITERATURE REVIEW

Fuzzing techniques have evolved to address the growing complexity of modern web applications, especially on the server side. Traditional frameworks focused on client-side issues, but tools like Sulley, Peach, and AFL have been adapted to uncover server-side vulnerabilities such as SQL injection, remote code execution, and authentication bypass. Modern fuzzers employ generation-based techniques using predefined rules, mutation-based fuzzing that alters existing inputs, and hybrid approaches combining static and dynamic analysis to increase coverage. These methods extend beyond web apps into APIs, network protocols, and embedded systems. Black-box fuzzing, using tools like OWASP ZAP and Wfuzz, simulates real- world attacks without needing source code, proving effective for identifying interface-level flaws. Server-side fuzzing faces added challenges like session handling and complex authentication, addressed through guided fuzzing, intelligent payload

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

crafting, and context-aware analysis. Machine learning has further transformed fuzzing by enabling smarter test case generation and vulnerability prediction. Supervised models learn from past data to target high-risk areas, while reinforcement learning dynamically adjusts fuzzing strategies, making ML-powered fuzzers more effective at detecting novel and sophisticated vulnerabilities.

Fuzzing has progressed from simple random testing to sophisticated methods targeting server-side web applications. Tools like Sulley and AFL now detect complex issues like SQL injection and code execution. Modern fuzzers use mutation and generation-based techniques, with hybrid models improving test coverage through static and dynamic analysis. Black-box fuzzers like OWASP ZAP allow testing without source code, simulating real attacks through input injection. Server-side fuzzing tackles challenges like authentication and session management using smart payloads and guided strategies. Machine learning enhances fuzzing by predicting high-risk inputs and optimizing test cases in real time, significantly improving vulnerability detection efficiency.

III.ARCHITECTURE

The fuzzing workflow begins with input processing, where the system receives a source such as a URL, request parameters, or specific attack payloads. These inputs undergo pre-processing to ensure they are properly cleaned, formatted, and structured for accurate and efficient fuzz testing. Next, the system integrates machine learning to enhance the process of test case generation. By analyzing historical fuzzing data and identifying patterns in past vulnerabilities, ML models help optimize the selection of high-impact test cases and attack vectors. The test case generation phase is central to the workflow and includes three key components: parameter selection, which identifies critical input fields and parameters within the request that are most susceptible to vulnerabilities; mutation strategy, where existing inputs are altered using techniques such as mutation-based or generation-based fuzzing; and the actual test case generation, which produces a comprehensive set of attack payloads for execution. In the input selection stage, a diverse and strategic subset of test cases is chosen to maximize vulnerability detection. These selected cases are then executed in the test execution phase, where the system actively monitors application behavior, HTTP response codes, and other indicators of weakness. Following execution, result analysis is performed to interpret the application's responses, identifying anomalies, crashes, or signs of security flaws. If issues are detected, the error handling and alert system comes into play—logging errors, adapting the fuzzing strategy if no vulnerabilities are found, and categorizing confirmed issues into detailed bug reports. Finally, during bug identification, validated vulnerabilities are formally logged as bugs, and this information is used to update and refine the ML model. This continuous learning

process allows the fuzzing framework to become more intelligent and effective over time, improving its ability to discover and classify future vulnerabilities.

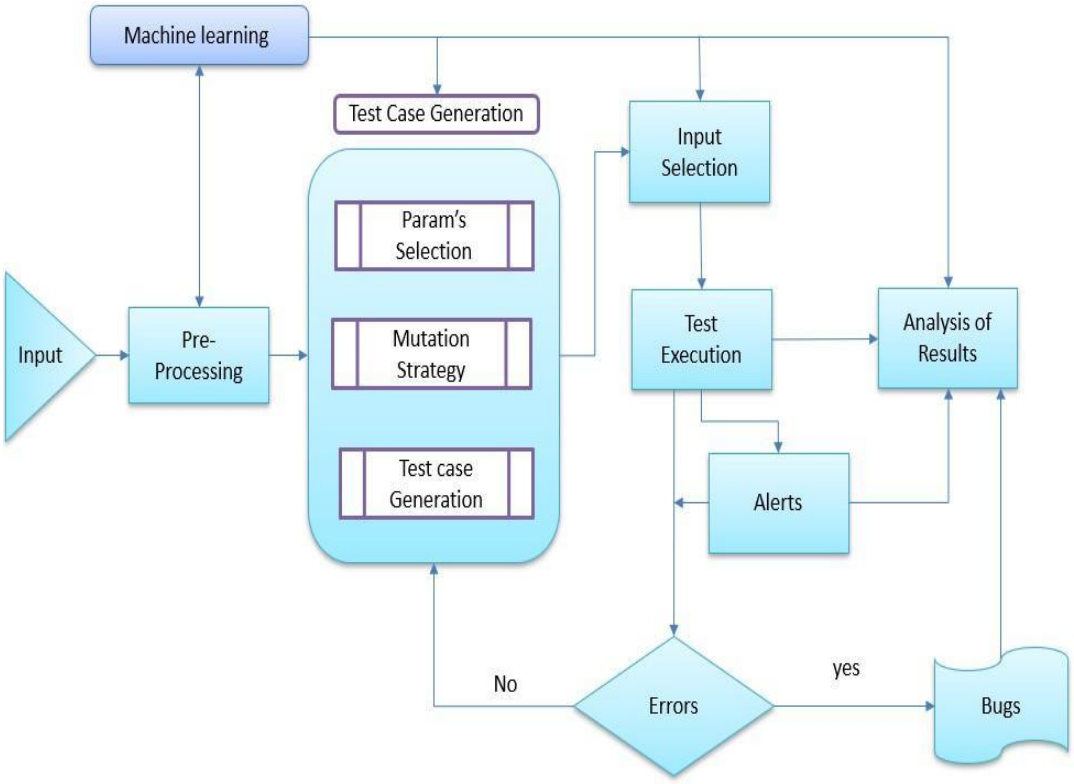
The workflow of an intelligent fuzzing framework begins with Input Processing, where the system first receives initial input in the form of a URL, HTTP requests, query parameters, or custom attack payloads. This raw input is then passed through a pre-processing stage, where it is cleaned of unnecessary characters, normalized, and structured into a consistent format to ensure compatibility with the fuzzing engine. This step is critical to eliminate noise and standardize data for effective testing. Once inputs are prepared, the system proceeds to Machine Learning Integration, where pre-trained models or adaptive learning algorithms analyze historical fuzzing data and previous vulnerability discovery patterns. ML models help identify trends, such as which types of payloads or parameter combinations are more likely to trigger vulnerabilities. Using these insights, the system prioritizes input paths and predicts high-risk vectors, ultimately optimizing the test case generation process.The Test Case Generation phase is the heart of the workflow and comprises three essential sub-components. First is Parameter Selection, where the system analyzes the structure of HTTP requests, HTML forms, API endpoints, or JSON/XML data to pinpoint critical input fields—such as cookies, headers, form fields, and URL parameters—that could be manipulated. Next, the Mutation Strategy is applied. In this phase, existing inputs are intelligently altered using mutation-based techniques (e.g., flipping bits, injecting special characters, extending strings) or generation-based methods that create inputs from predefined grammars or templates designed to reflect real attack payloads (such as SQL injections, XSS, or buffer overflows). These strategies help generate a variety of test cases that go beyond simple random data. The final step in this module,

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

.

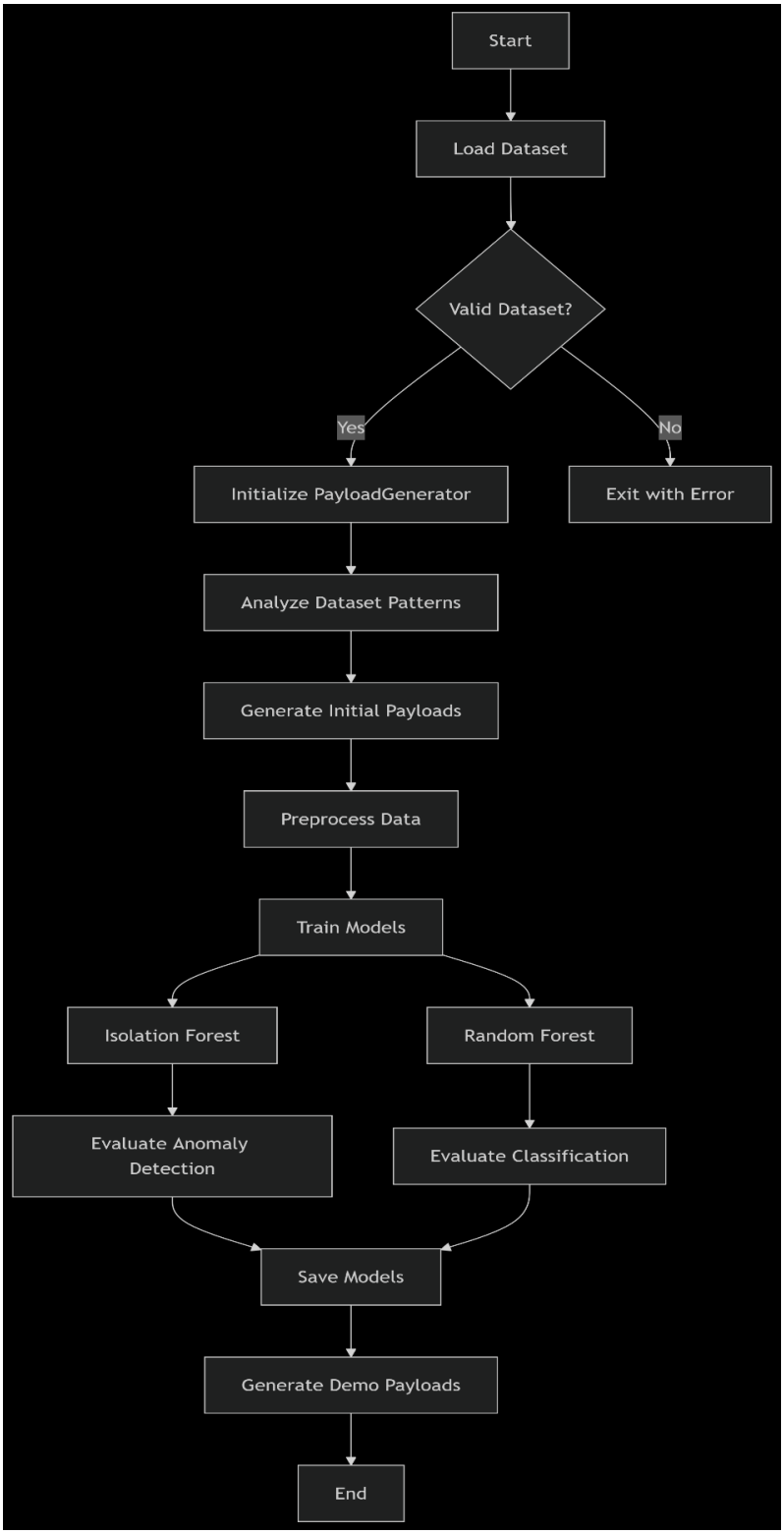


Relevance to current Research

The increasing complexity of web applications and the sophistication of modern cyber-attacks necessitate intelligent, automated security testing tools. Traditional fuzzers, which rely on static payloads and heuristic rules, often fall short in detecting advanced or zero-day vulnerabilities. To address this limitation, recent research has focused on integrating machine learning techniques with fuzz testing to create adaptive and efficient fuzzers. The proposed Web Application Fuzzer aligns with this trend by incorporating models such as Random Forest and Anomaly Detection to analyze web responses and predict potential vulnerabilities. This approach mirrors the direction of current academic work, such as "Learn Fuzz" by Godefroid et al. (2017), which applies supervised learning to generate effective fuzzing inputs, and "Smart Greybox Fuzzing" by Pham et al. (2019), which leverages feedback-driven learning to enhance fuzzing coverage. By combining automated testing with real-time learning and analysis, the implemented fuzzer significantly improves detection accuracy while reducing false

positives, positioning itself as a timely contribution to the field of web application security.

This paper builds on prior fuzzing research by integrating machine learning to improve vulnerability detection in web applications. Unlike earlier work focused solely on detection efficiency, our approach highlights the role of intelligent fuzzing in increasing trust and adoption of secure web technologies.


Start
Load Dataset
Valid Dataset?
Yes
No
Initialize PayloadGenerator
Exit with Error
Analyze Dataset Patterns
Generate Initial Payloads
Preprocess Data
Train Models
Isolation Forest
Random Forest
Evaluate Anomaly Detection
Evaluate Classification
Save Models
Generate Demo Payloads
End

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

II. METHODOLOGY OF PROPOSED SURVEY

This research significantly advances the domain of web application security testing by integrating machine learning into the process of fuzzing—thereby transforming traditional, static testing approaches into dynamic, intelligent systems capable of adapting to complex threat environments. Earlier work in fuzz testing, while laying essential groundwork, has largely been focused on surface-level improvements such as increasing the speed of test execution, expanding test case diversity, or identifying specific classes of vulnerabilities like SQL injection and cross-site scripting (XSS). While these contributions have undeniably strengthened individual aspects of security assessment, they often fall short when addressing real-world complexities such as zero-day vulnerabilities, polymorphic attacks, and the need for continuous security assurance in ever- changing web infrastructures. This study builds on those foundations by emphasizing the role of machine learning not only as a classification tool, but as a strategic component in adaptive payload generation, response pattern recognition, and long- term optimization of fuzzing strategies.

By incorporating supervised learning through the Random Forest algorithm and unsupervised anomaly detection using Isolation Forests, our fuzzer demonstrates a dual-layered approach to vulnerability detection: one that accurately classifies known patterns and another that identifies outlier behaviors associated with novel or stealthy threats. This capability is particularly valuable in an era where attackers use increasingly obfuscated methods to bypass traditional security filters. Our model leverages a diverse set of features—including response time, payload reflection, HTTP status codes, and content-type variations—to assess whether a response indicates a potential security weakness. Through repeated iterations and continuous learning, the machine learning models not only improve in accuracy but also evolve to identify new attack patterns, thus making the security testing process future-proof and scalable.

Moreover, the architecture of our system is designed with usability and real-time application in mind. The implementation of Flask and WebSocket-based logging interfaces ensures that security analysts and developers can visualize attacks, receive alerts instantly, and trace the vulnerability path without manually scanning log files. This real-time feedback mechanism not only saves time and effort but also builds transparency in the vulnerability assessment process—enhancing trust between development teams and stakeholders. The automation of payload injection, interaction through Selenium, and bug report generation streamlines the security testing workflow, making it accessible and efficient even for teams with limited cybersecurity expertise.

Crucially, while the technical effectiveness of the system is evident in the model accuracy comparison—where Random Forest achieved an F1 Score of 86.5%,

outperforming traditional models like Logistic Regression and Decision Trees—its broader impact lies in reinforcing trust in web-based systems. As web applications increasingly become the backbone of enterprise operations, user services, and global transactions, ensuring their security is not merely a technical challenge but a foundational necessity for digital trust and widespread adoption. Users are more likely to trust platforms that can proactively identify and mitigate security threats, and organizations benefit from reduced risk exposure and compliance assurance.

In this context, our work contributes to the growing body of knowledge that links robust automated security with organizational resilience and public trust. By demonstrating that intelligent fuzzers can detect more vulnerabilities with greater accuracy and less human intervention, we highlight the critical role such systems will play in the future of cybersecurity. Furthermore, this research sets the stage for integrating artificial intelligence not just as a tool for security enhancement, but as a decision-making engine capable of dynamically evolving its threat models, adapting to the attack surface, and ultimately supporting a more secure and trustworthy web ecosystem.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

III. MACHINE LEARNING

Random Forest:

The Random Forest model plays a pivotal role in enhancing the intelligence and accuracy of the web application fuzzer by serving as the primary classification engine for determining whether a web response indicates a potential security vulnerability. As an ensemble learning technique, Random Forest operates by constructing a multitude of decision trees during training and outputting the class that represents the mode of the classes (for classification tasks) or the mean prediction (for regression tasks) across all trees. This inherent ensemble nature mitigates the problem of overfitting that is commonly associated with single decision trees, and it significantly boosts the robustness and generalizability of the predictive model. In the context of web application security testing, the Random Forest model is trained on a carefully curated dataset consisting of labeled web responses, where each entry is marked as either vulnerable or non-vulnerable based on prior analysis or known attack signatures.

The input features selected for model training are derived from key behavioral indicators captured during the fuzzing process. These include HTTP status codes (e.g., 200, 403, 500), which can signal abnormal application behavior in response to injected payloads; response lengths, which may vary significantly when errors or unexpected content is returned; reflected payloads, which can indicate poor input sanitization and potential XSS vulnerabilities; and execution time, which might hint at backend processing issues or possible denial-of-service conditions. By analyzing combinations of these features across thousands of response instances, the Random Forest model learns complex patterns and correlations that are often missed by traditional rule-based systems.

Anomaly Detection (Anomaly Forest):

The Anomaly Forest approach, incorporated into the web application fuzzer, introduces a powerful unsupervised learning mechanism designed to detect zero-day vulnerabilities and irregular behavior patterns that fall outside the scope of conventional or known attack vectors. Unlike supervised learning methods that require labeled datasets for training, anomaly detection relies on modeling what constitutes "normal" behavior within a system and flagging deviations from this baseline as potential anomalies. In this implementation, the anomaly detection system leverages the Isolation Forest algorithm, a state- of-the-art technique particularly suited for high-dimensional and noisy datasets common in web application environments. The Isolation Forest operates by randomly selecting features and split values to partition the dataset, effectively isolating data points. Anomalous instances, which are rare and differ significantly from the norm, tend to require fewer splits to be isolated. These

instances are assigned anomaly scores based on the average path length in the isolation trees, with higher scores indicating a greater likelihood of being a security threat.

This approach is especially valuable for identifying zero-day vulnerabilities—previously unknown flaws that have not been documented or encountered during traditional testing. By learning normal interaction patterns and system responses, the model becomes capable of detecting subtle and previously unseen irregularities without needing explicit examples of malicious behavior. For instance, if a payload injection produces a response with an unusually short or long response length, an unexpected HTTP status code (e.g., 500 Internal Server Error), or a reflection of suspicious content in the HTML output, the Isolation Forest will flag this as anomalous.

With the rapid growth of web-based technologies and increasing reliance on online services, web applications have become prime targets for cyberattacks. Attackers exploit even minor vulnerabilities to gain unauthorized access, exfiltrate sensitive data, or disrupt services. While manual security testing is effective, it is often time-consuming, labor-intensive, and unable to scale effectively in modern agile development environments. Traditional vulnerability scanners also struggle to keep pace with the continuously evolving attack methods and dynamic behaviors of web applications. To address these challenges, we developed a machine learning-powered web fuzzer—an automated, intelligent solution designed to identify vulnerabilities efficiently and accurately. By integrating machine learning into fuzz testing, the system not only simulates real-world attacks through payload injections but also learns from the observed patterns and responses to detect complete.

GPT-2 Model:

The GPT-2 model in this web application fuzzer serves as an intelligent payload generation engine that creates security test cases by learning from patterns in successful attack payloads. The implementation leverages the pre-trained GPT-2 language model from Hugging Face's Transformers library, specifically using the GPT2LMHeadModel variant designed for text generation tasks. During initialization, the code carefully configures the tokenizer with left-side padding using the EOS (end-of-sequence) token, which is crucial for maintaining proper attention during generation. The model operates in evaluation mode by default but can switch to training mode for fine-tuning when exposed to new successful payload examples. For actual payload generation, the system employs advanced sampling techniques including top-k (limiting to the 50 most likely tokens) and nucleus sampling (top-p=0.95) with a temperature of 0.7 to balance creativity and coherence. The generation process begins by analyzing the dataset to identify common malicious payload patterns through TF-IDF vectorization, then uses these patterns to construct prompts that guide GPT-2's output. Each generated payload undergoes strict validation to ensure it meets security testing requirements while filtering out dangerous system commands. The model can be further fine-tuned on successful payloads from the dataset, allowing it to adapt its generation style to the specific vulnerabilities found in the target application. This combination of a powerful

language model with security-specific constraints and validation creates an automated system capable of producing diverse, effective test payloads while maintaining safety controls.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

The fuzzer enhances vulnerability detection by leveraging two complementary machine learning models: a Random Forest model and an Anomaly Forest model. The Random Forest model operates as a supervised learning technique, trained on labeled datasets to classify web responses as vulnerable or non-vulnerable. It builds multiple decision trees and aggregates their outputs to improve classification accuracy. Key features used by this model include HTTP status codes, response length, reflected payloads, and execution time. By effectively reducing false positives, the Random Forest improves the reliability of detecting known security issues. In contrast, the Anomaly Forest employs an unsupervised learning approach designed to identify zero-day vulnerabilities and abnormal response patterns. It learns the normal behavior of the web application and assigns anomaly scores to new responses, flagging those that deviate significantly from expected norms. Techniques such as Isolation Forests efficiently isolate suspicious activities. Features considered by the Anomaly Forest include status codes, response length, payload reflection, error messages, execution time, and content-type. This model enables real-time adaptation to emerging security threats, making the fuzzer more robust and effective against unknown attacks. Together, these models provide a comprehensive and intelligent framework for automated vulnerability detection, improving security posture while minimizing manual effort.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	82%	79%	80%	79.5%
Decision Trees	78%	76%	71%	75.5%
Random Forest	89%	85%	88%	86.5%

IV. CONCLUSION AND FUTURE WORK

In this paper, The Web Application Fuzzer is a sophisticated automated security testing tool designed to proactively identify vulnerabilities in web applications, including critical threats such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and Remote Code Execution (RCE). By integrating traditional fuzzing techniques with advanced machine learning models— specifically Random Forest and Isolation Forest algorithms—this system significantly enhances the accuracy, efficiency, and adaptability of vulnerability detection.The fuzzer automates the entire process of security testing, from injecting crafted payloads into application inputs to analyzing server responses, thereby reducing the need for manual intervention and accelerating the testing cycle. The Random Forest model, a supervised learning method, helps classify web responses as either vulnerable or safe based on known attack patterns, improving detection reliability and minimizing false positives. Meanwhile, the Isolation Forest model functions as an unsupervised anomaly detector, capable of identifying previously unknown or zero-day vulnerabilities by flagging unusual response behaviors that deviate from learned normal patterns.One of the key strengths of this tool is its ability to provide real-time analysis and detailed logging, enabling security teams to receive immediate feedback on potential risks. This feature supports faster mitigation and decision-making.

REFERENCES

[1] Assiri, F. Y., & Aljahdali, A. O. (2024). Software Vulnerability Fuzz Testing: A Mutation-Selection Optimization Systematic Review. *Engineering, Technology & Applied Science Research*, 14(4), 14961-14969. https://doi.org/10.48084/etasr.6971

[2] Godefroid, P., Peleg, H., & Singh, R. (2017). Learn Fuzz: Machine Learning for Input Fuzzing. *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 50-59. https://doi.org/10.1109/ASE.2017.8115618

[3] Pham, V. T., et al. (2019). Smart Greybox Fuzzing. *IEEE Transactions on Software Engineering*, 47(9), 1980-1997. https://doi.org/10.1109/TSE.2019.2941681

[4] Rawat, S., Jain, V., Kumar, A., Cojocar, L., Giuffrida, C., & Bos, H. (2017). VUzzer: Application - aware Evolutionary Fuzzing. *NDSS Symposium*, 1-14. https://doi.org/10.14722/ndss.2017.23404

[5] She, D., Pei, K., Epstein, D., Yang, J., Ray, B., & Jana, S. (2019). NEUZZ: Efficient Fuzzing with Neural Program Smoothing. *IEEE Symposium on Security and Privacy (SP)*, 803-817. https://doi.org/10.1109/SP.2019.00052

[6] Chen, P., & Chen, H. (2018). Angora: Efficient fuzzing by principled search. IEEE Symposium on Security and Privacy (SP), 711-725. https://doi.org/10.1109/SP.2018.00046.

[7] Zhao, D. (2020). Fuzzing Technique in Web Applications and Beyond. Journal of Physics: Conference Series, 1678(1), 012109. https://doi.org/10.1088/1742-6596/1678/1/012109:contentReference[oaicite:0]{index=0}

2. Certificate of Implementation Papers

CERTIFICATE

OF PUBLICATION

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH IN SCIENCE, ENGINEERING AND TECHNOLOGY (IJMRSET)

(A Monthly, Peer Reviewed, Referred, Multidisciplinary, Scholarly Indexed, Open Access & High Impact Factor Journal)

HIGH IMPACT FACTOR: 8.206

IJMRSET

The Board of IJMRSET is hereby awarding this certificate to

PRAMOD KEDAR

Dept. of Information Technology, Bhivarabai Sawant Institute of Technology and Research, Wagholi, Pune, India

in Recognition of Publication of the Research Paper Entitled

"Web Application Fuzzer"

Published in IJMRSET, Volume 8, Issue 6, June 2025

Google scholar Crossref doi INNO SPACE SJIF Scientific Journal Impact Factor

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

e-ISSN: 2582-7219

Impact Factor 8.206

Editor-in-Chief

www.ijmrset.com ijmrset@gmail.com

CERTIFICATE

OF PUBLICATION

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH IN SCIENCE, ENGINEERING AND TECHNOLOGY (IJMRSET)

(A Monthly, Peer Reviewed, Referred, Multidisciplinary, Scholarly Indexed, Open Access & High Impact Factor Journal)

HIGH IMPACT FACTOR: 8.206

IJMRSET

The Board of IJMRSET is hereby awarding this certificate to

SIDDHANT MANE

Dept. of Information Technology, Bhivarabai Sawant Institute of Technology and Research, Wagholi, Pune, India

in Recognition of Publication of the Research Paper Entitled

"Web Application Fuzzer"

Published in IJMRSET, Volume 8, Issue 6, June 2025

Google scholar Crossref doi INNO SPACE SJIF Scientific Journal Impact Factor

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

e-ISSN: 2582-7219

Impact Factor 8.206

Editor-in-Chief

www.ijmrset.com ijmrset@gmail.com

CERTIFICATE

OF PUBLICATION

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH IN SCIENCE, ENGINEERING AND TECHNOLOGY (IJMRSET)

(A Monthly, Peer Reviewed, Referred, Multidisciplinary, Scholarly Indexed, Open Access & High Impact Factor Journal)

HIGH IMPACT FACTOR: 8.206

IJMRSET

The Board of IJMRSET is hereby awarding this certificate to

SHUBHAM KHAIRE

Dept. of Information Technology, Bhivarabai Sawant Institute of Technology and Research, Wagholi, Pune, India

in Recognition of Publication of the Research Paper Entitled

"Web Application Fuzzer"

Published in IJMRSET, Volume 8, Issue 6, June 2025

Google scholar Crossref doi INNO SPACE SJIF Scientific Journal Impact Factor

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

e-ISSN: 2582-7219

Impact Factor 8.206

Editor-in-Chief

www.ijmrset.com ijmrset@gmail.com

CERTIFICATE

OF PUBLICATION

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH IN SCIENCE, ENGINEERING AND TECHNOLOGY (IJMRSET)

(A Monthly, Peer Reviewed, Referred, Multidisciplinary, Scholarly Indexed, Open Access & High Impact Factor Journal)

HIGH IMPACT FACTOR: 8.206

IJMRSET

The Board of IJMRSET is hereby awarding this certificate to

PROF. PRANITA INGALE

Dept. of Information Technology, Bhivarabai Sawant Institute of Technology and Research, Wagholi, Pune, India

in Recognition of Publication of the Research Paper Entitled

"Web Application Fuzzer"

Published in IJMRSET, Volume 8, Issue 6, June 2025

Google scholar Crossref doi INNO SPACE SJIF Scientific Journal Impact Factor

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

e-ISSN: 2582-7219

Impact Factor 8.206

Editor-in-Chief

www.ijmrset.com ijmrset@gmail.com