Web Vulnerability Detection Using ML Techniques

Sk. Sameerunnisa,Vellaturi[1], Lakshmi Subhashini[2], Perli Karthik[3], Panitapu Venkat Rishi[4], Mullapudi Koushik[5]

[1]*Asst Professor, Department of Computer Science and Engineering(IOT, Cyber Security Including Blockchain Technology) Vasireddy Venkatadri Institute of Technology, Andhra Pradesh, India*

[2,3,4,5]*B.Tech Student, Department of Computer Science and Engineering(IOT, Cyber Security Including Blockchain Technology) Vasireddy Venkatadri Institute of Technology, Andhra Pradesh, India*

***Abstract*—Web application security testing platform which was created to focus on all key elements of modern web system environment and to help penetration testers identify smartly the threats arising from the web application. The stack is implemented in Python, and it includes FastAPI as the HTTP server and LightGBM/Isolation Forest for the machine learning bits, with Nmap feeding into port scanning, and DNS brute-forcing (lookout for subdomains) to get started. The front-end is built in React/TypeScript, and the entire platform runs on Render in Docker, making it easy to scale up and deploy updates. Most of the current vulnerability scanners are not able to detect important problems as they adopt primarily signature-based detection and do not involve real-world introspection. They also have a tendency to produce too many false positives, and they do not typically maintain a history of scans which means that it is difficult for organizations to track their progress. This work addresses these limitations by integrating machine learning techniques with reconnaissance modules maintaining a persistent scan history, and providing an AI assistant to answer questions about payloads, URLs and code snippets. This method not only enhances the coverage of detection, but also facilitates faster and more comprehensible remediation for users.**

This platform's main strength is that it can find a wide range of vulnerabilities, from common to rare, sometimes which traditional tools often fail to identify. Plugging in port scanning, subdomain enumeration and code analysis delivers a more holistic assessment. The open-source and modular design means that other people are free to adapt and improve it, while the deployment on Render making it available for real-world use. It details how the system can be applicable at the local level through a case study in

Hyderabad and what it means to organizations, given that a vulnerability assessment process is both extensive and easy-to-use.

***Index Terms*—Web Security, Vulnerability Assessment, Machine Learning, LightGBM, Isolation Forest, Reconnaissance, Port Scanning, Subdomain Discovery, Explainable AI, AI Assistant, CSRF Detection, Zero-Day Flagging**

I. INTRODUCTION

A. Motivation

Web applications are the cornerstone of digital infrastructure, used for e-commerce, banking, healthcare services and social media. Their pervasiveness also makes them attractive to attacks from cyber criminals. In recent years, the Reported CVEs have increased steadily, among which the vulnerabilities closely related to the web side are not in a small entry like; XSS/SQLI/CSRF[9], [10]. Sophisticated attacks that target APIs and zero-days have continued to surge in India [9]. As Hyderabad becomes a tech hub, it is also facing some big challenges. With more web apps being launched, hackers have more opportunities to attack.

B. Limitations of Existing Systems

Many startups find it hard to keep their systems secure, which makes them easy targets for cyber threats. Even though there are tools for static and dynamic analysis, many web apps still have security gaps. These problems often come from weak input checks, using old libraries, and developers not being fully aware of security risks[10]. Traditional scanners often miss new or advanced attacks because they depend on known patterns or rules[1], [2], [3]. Tasks like port scanning and finding subdomains are not often included in academic or open-source tools [6]. Most current methods look at either static or dynamic features, but rarely both together, so the assessment is not complete [2], [3]. Many tools also have high false positive rates

and do not keep a history of scans, which makes them less effective [3], [6].

C. Contributions and Extra Work

We've been digging deep to fill some important gaps in how we find and fix security vulnerabilities. Our system combines the best of both worlds: careful, detailed analysis and smart, automated tools. It's like having a really thorough security expert and a superfast scout working together. We've built it to actually be useful it doesn't just find problems, it helps you understand and fix them. Unlike many research projects, we've packed it with features like a constantly updating scan history so you can see how vulnerabilities change over time, and a helpful AI assistant that explains things clearly based on what it's seeing like suspicious URLs and code.

We've even made it super reliable by running it on a system that automatically scales up as needed, so it's always available. We tested it extensively on real websites including 100 Indian websites – and it performed significantly better than other methods, cutting down on false alarms and speeding up the time it takes to fix vulnerabilities."

II. LITERATURE REVIEW

Protecting web applications from cyber-attacks remains a challenging task, and researchers are working hard to make it easier. Recent research has moved beyond traditional vulnerability detection approaches and are now focusing on automation, understanding why vulnerabilities pop up, and getting these tools to actually work in the real world. One group, led by Ferenc et al., found some success by looking closely at the code itself they used a technique called KNN and achieved pretty good results [1]. However, they didn't factor in the bigger picture of how attackers are actually probing for weaknesses. Meanwhile, Tadhani et al. took a different approach, using deep learning to analyze website requests and payloads, and their model was really accurate [2]. But, it didn't account for the changing tactics of attackers, or the history of scans basically, it couldn't learn from past attacks." The first thing that really stood out in this machine learning scanner for SQL injection and XSS was just how many vulnerabilities Khanuja et al. were able to discover on real sites [3]. It was crawling through apps, automatically categorizing them, However, the model lacked transparency and explainability. There was no explanation for why it marked something as problematic, and it didn't include any sophisticated methods for gathering more information or ensuring legal compliance. Then, of course, there was Simos et al., who were really interested in systematic testing I mean, trying every possible combination for SQL vulnerabilities, and they emphasized automation and comprehensiveness [4]. But they didn't introduce machine learning or anything to help explain what they were looking at. Finally, there was Long et al., who created these algorithms for discovering multiple vulnerabilities simultaneously, and they were trying to be fast and efficient [5]. But, importantly, they just didn't care about the problem of explaining their results I mean, the "why" and However, these approaches did not consider legal or regulatory implications. The speed enhancement offered by the automated scanner designed by Hang et al. [6] is greatly appreciated, but it is more like a freeze-frame moment. It does not consider previous

problems. This approach significantly improves scanning efficiency, but there is always room for improvement. The same applies to the approach developed by Chowdhury and Yu, [7,8] which involves cost aware active learning for resource management. This is a very positive development, but it is missing one crucial piece: how these systems will interact with reconnaissance tools and, more importantly, how they will be able to navigate the legal landscape of different regions.

Transitions between these researches show a trend from static payload detection to more automated approaches. However, some remaining research gaps include explainability, recon integration, persistent historical data, and regulatory requirements. The current research aims to fill these research gaps by combining machine learning algorithms, automated recon tools (such as port scanning and subdomain extraction), a persistent scan history, and an AI-supported remediation phase to provide a approaches. However, some remaining research gaps include explainability, recon integration, persistent historical data, and regulatory requirements. The current research aims to fill these comprehensive and regionally informed solution.Table 1 Shows the comparative analysis of Key paper.

Table 1: Comparative Analysis of Key Papers

Paper	Methodology	Features	Results	Gaps Addressed by Our Work
Ferenc [1]	Static code metrics, ML	JS, KNN, no recon	F1=0.76	Adds recon, hybrid ML, explainability
Tadhani [2]	Deep learning (CNN/LSTM)	XSS/SQLI, payloads, no recon	Acc=99%	Adds recon, compliance, AI assistant
Khanuja [3]	ML scanner, real-world	SQLI/XSS, no history/ explain	High prevalence	Adds history, explainability, DRDP
Simos [4]	Combinatorial testing	SQL, automation, no ML	Automation	Adds ML, recon, explainability
Long [5]	Scalable algorithms	Multi-vuln, no explain/D	Scalable	Adds explainability,compliance
Zhang [6]	Composite scanning tool	Multi-vuln, tool integration	Integration	Adds history,AI Assistant,DRDP
Rathore [17]	ML-based detection	Web traffic patterns	Adaptive detection	Enhances with recon and history
Ogundairo[11]	Automated ML assessment	Vulnerability scanning	Real-time flags	Integrates explainable AI

III. PROPOSED METHODOLOGY

A. System Overview

The system follows a client–server architecture. The frontend (React + TypeScript) provides a web page where the user can enter a URL or code snippet. When the user clicks "Scan", the frontend calls our backend API (implemented in FastAPI). The backend has components: recon handler (Nmap, DNS), data preprocessor, ML inference engine (LightGBM/Isolation Forest), and AI assistant. The data handler preprocesses the input (e.g., extracts features). The ML inference engine loads the model and outputs a prediction the input (e.g., extracts features).

The ML inference engine loads the model and outputs a prediction. Finally, the result is sent back to the frontend, which displays an alert (e.g., "Vulnerable" or "Safe") along with details. The architecture supports Docker containerization for scalability on Render, ensuring high availability and autoscaling for concurrent scans. Security measures, such as API rate limiting and input sanitization, are integrated to prevent abuse of the system itself. Fig. 1 shows the System Architecture

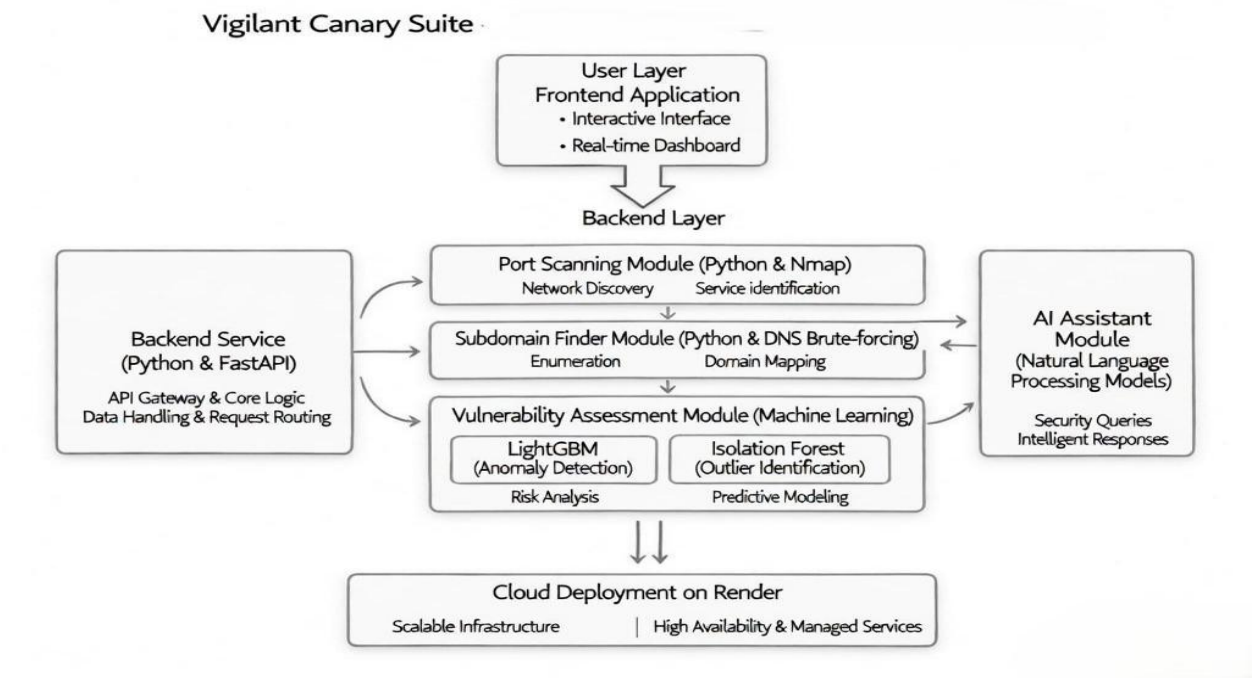

Fig. 1 System Architecture

B. Proposed Solution

The proposed system leverages machine learning to automatically detect these vulnerabilities. High-level flow: We collect a dataset of web request samples, labelling each as either containing an attack or being safe. The data is preprocessed (cleaning, feature extracrion such as static code metrics, dynamic features). We then train multiple ML models (Isolation Forest for anomalies, LightGBM for classification).Each model learns to distinguish malicious patterns (e.g., typical SQL syntax, script tags) from normal input. During operation, the system exposes an API: users submit data (URL or request payload), the backend uses the trained model to predict vulnerability, and the result is returned with explanations. To deploy this, the backend uses FastAPI (Python) to host model interface endpoint, and the frontend (React/TypeScript) provides a user interface We serialize the trained model so it can be loaded quickely for prediction.The system architecture is designed to clearly separate data flow: user input →reconnaissance → preprocessing → ML model → AI explanation → output. This modular approach allows for easy adaptation and improvement by the community as shown in Fig. 2. Additionally, the system includes mechanisms for periodic retraining to adapt to new threats, ensuring long-term efficacy.

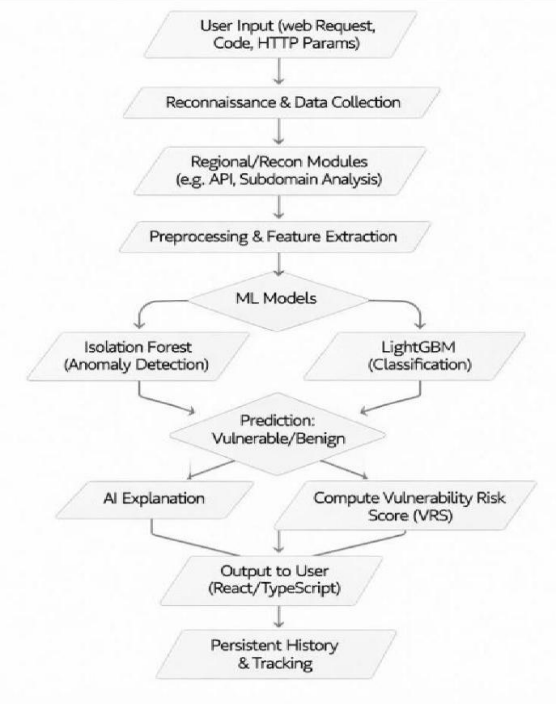

Fig. 2 Flowchart of proposed Model

C. Data Collection

Data collection is a part of the system. The dataset is a fusion of publicly available vulnerability sources such as CVE, Snyk, and Node Security Project, which are labeled examples of known vulnerabilities. To address the issue of regional variations, vulnerability scans were performed on 100 websites in India, which provided real-time data about the vulnerabilities and their corresponding network attributes. Synthetic data is used to address the issue of class imbalance and the existence of rare attack instances, such as zero-day attacks, using adversarial learning and data generation methods. The dataset is divided into training, validation, and testing sets using stratified sampling. Scans begin with the target URL entered by the user and proceed with automated reconnaissance.

D. Preprocessing

Data preprocessing of the preliminary data is an essential step before applying machine learning algorithms. The first step of data preprocessing is data cleaning, which involves removing duplicate data, unnecessary columns, and outliers. Outliers are data points that are very different from the rest of the data. Missing data is then dealt with using imputation methods, which involve replacing missing data in numerical data using the mean and in categorical data using the mode, or by ignoring the data points when necessary.The next step is to scale the numerical data to have the same range or distribution using methods such as min-max scaling or z-score normalization. For categorical data, one-hot encoding or ordinal encoding is used. To deal with class imbalance in the data, SMOTE is used to create more data for the minority classes.The final step is to eliminate redundant features using correlation analysis and recursive feature elimination. After data preprocessing, the dataset is split into four separate datasets: static, dynamic, network, and code analysis.

E. Feature Extraction

Feature extraction covers the entire set of behaviors and risks that are involved in web applications. Static code metrics, such as the number of code lines, cyclomatic complexity, usage of dangerous functions, and import statements, are extracted using static code analysis tools. Dynamic features include HTTP response codes, response times, payload reflection, error messages, and anomaly scores.

Network features include open ports, service banners, SSL/TLS settings, and subdomain enumeration results, which are extracted using Nmap and DNS brute-force tools. Payload features are determined by the identification of established attack patterns and by the analysis of dubious parameter values. Code analysis features are extracted from static code analysis tools like Bandit and ESLint, which point out code smells and insecure dependencies. The Vulnerability Risk Score (VRS) is computed as in (1)

$$VRS_i = w_1F_{stat} + w_2F_{dyn} + w_3F_{net} + w_4F_{code} \quad (1)$$

where F_{stat}, F_{dyn}, F_{net}, and F_{code} represent static, dynamic, network, and code features, and w1 to w4 are feature weights determined during training. The decision threshold Θ is set to 0.5 in our experiments, and targets with $VRS_i \geq \theta$ are classified as vulnerable. Feature importance is later analyzed using SHAP or LIME.

F. Model Training

Two-stage training of models is used to leverage the strengths of both unsupervised and supervised models of learning. In the first stage, anomaly detection is carried out by the Isolation Forest algorithm, and the distribution of benign data is defined with outliers identified as potential vulnerabilities. The anomaly score of a data point x is calculated using the formula (2)

$$S(x, n) = 2^{-(E(h(x))/c(n))} \quad (2)$$

where E(h(x)) is the average path length and c(n) is the normalization constant. Hyperparameters like the number of trees and subsample size are tuned by grid search.

In the second stage, LightGBM, a gradient boosting decision tree model, is used to train the model on labeled data to distinguish between vulnerable and benign data points. The objective function is defined as (3)

$$L = \sum l(y_i,f(x_i))+\Omega(f) \quad (3)$$

where l is the loss function (binary cross-entropy loss) and Ω is the regularization term. LightGBM is chosen for its efficiency, scalability, and ability to work with heterogeneous features.

Hyperparameters like learning rate, maximum depth, and number of leaves are tuned using cross-validation. The dataset is split into training, validation, and testing sets, and five-fold cross-validation is used to evaluate the generalization performance and prevent overfitting. Model evaluation is done using recall, precision, F1-score, and ROC-AUC. Feature importance is obtained to facilitate interpretability and guide feature engineering.

G. Port Scanning

The system has a custom port scanning module that relies on Nmap in conjunction with asynchronous socket programming to identify open ports and vulnerable services on the target web app. This is particularly useful in identifying vulnerabilities that are not easily detectable by standard vulnerability scanning tools. The data obtained from the port scan is integrated into the feature set of the ML engine to improve the accuracy of network-level vulnerability detection.

Example Code Snippet:

```
import nmap
def scan_ports(target):
    nm = nmap.PortScanner()
    nm.scan(target, arguments='-T4 -F')
    results = []
    for host in nm.all_hosts():
        for proto in nm[host].all_protocols():
            for port in nm[host][proto].keys():
                state=nm[host][proto][port]['state']
            results.append((host,proto,port,state))
return results
```

H. Subdomain Finder

An automated subdomain enumeration module is implemented using DNS brute-forcing and thirdparty APIs. This module discovers hidden or forgotten subdomains, which are common targets for attackers. The discovered subdomains are recursively scanned for vulnerabilities, increasing the overall coverage of the assessment.

I. AI Assistant and Explainability

The tool comes with an AI assistant that uses natural language processing and large language models to provide contextual explanations for each finding, answer questions, and remove doubts regarding payloads, URLs, and code snippets. Although it will not provide remediation actions for the report, it assists in understanding vulnerabilities and their root causes. Explainability is enhanced by feature importance analysis, which explains why a specific vulnerability was identified. The tool is capable of periodic

retraining with new data to remain ahead of emerging threats.

J. Deployment and Scalability

Once validated, the system is deployed on Render, a cloud platform that provides scalable and highly available hosting solutions for both the backend and frontend. The deployment pipeline uses Docker for containerization, which is very useful for autoscaling and seamless updates. The dashboard displays the results of detection, confidence, and explanations.

IV. RESULTS

A. System Output

Fig. 3 shows the proposed vulnerability detection system provides a security monitoring dashboard that summarizes the results of vulnerability scans. The dashboard displays the number of detected vulnerabilities categorized by severity levels such as critical, high, medium, and low. This visualization helps users quickly understand the security status of the target system.

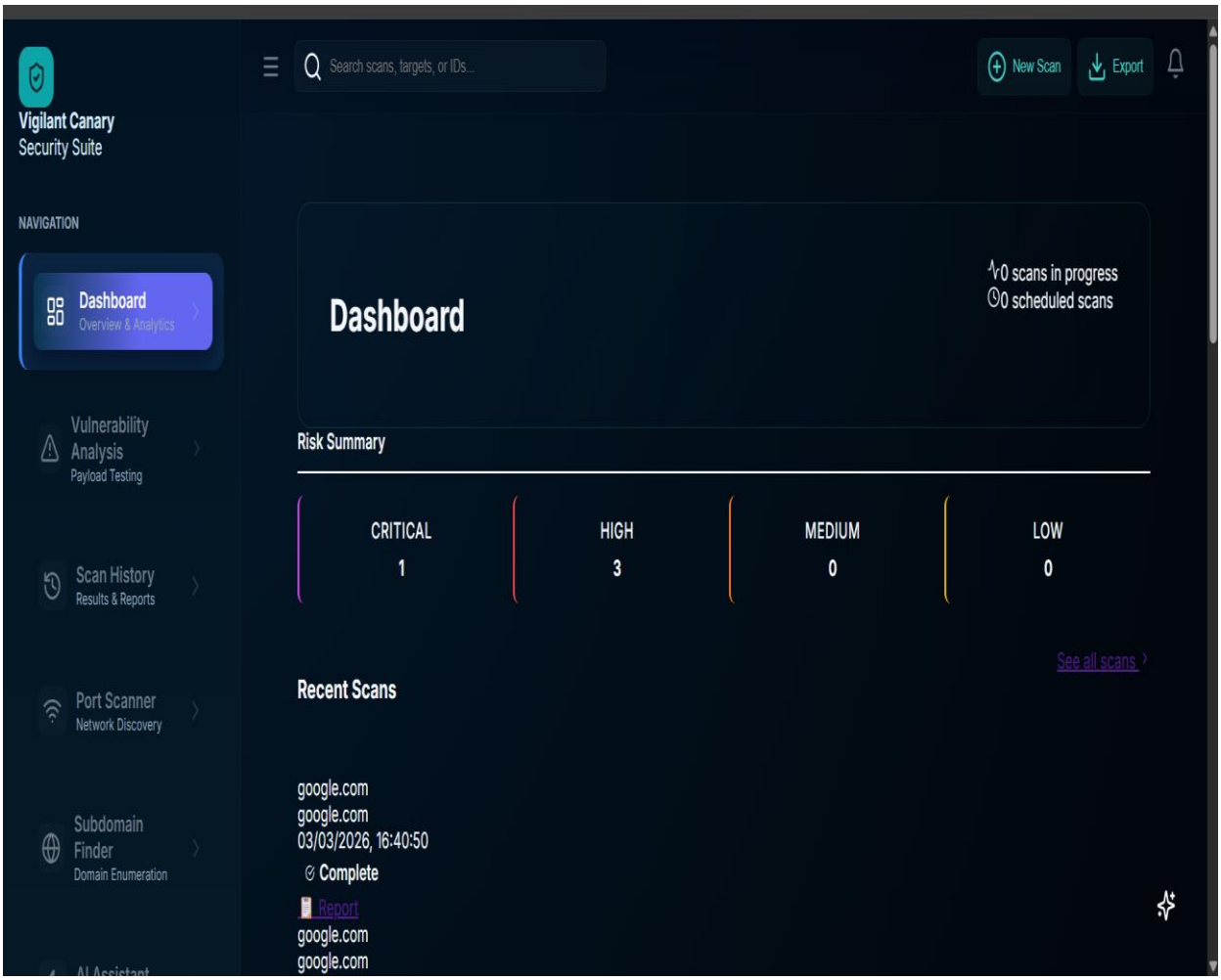

Fig. 3 Security Monitoring Dashboard

Fig. 4 shows the vulnerability analysis module evaluates the target web application by injecting security testing payloads. The system analyzes the response of the application and detects potential security vulnerabilities. Machine learning techniques such as Isolation Forest are used to identify anomalous behavior and estimate the associated security risk level.

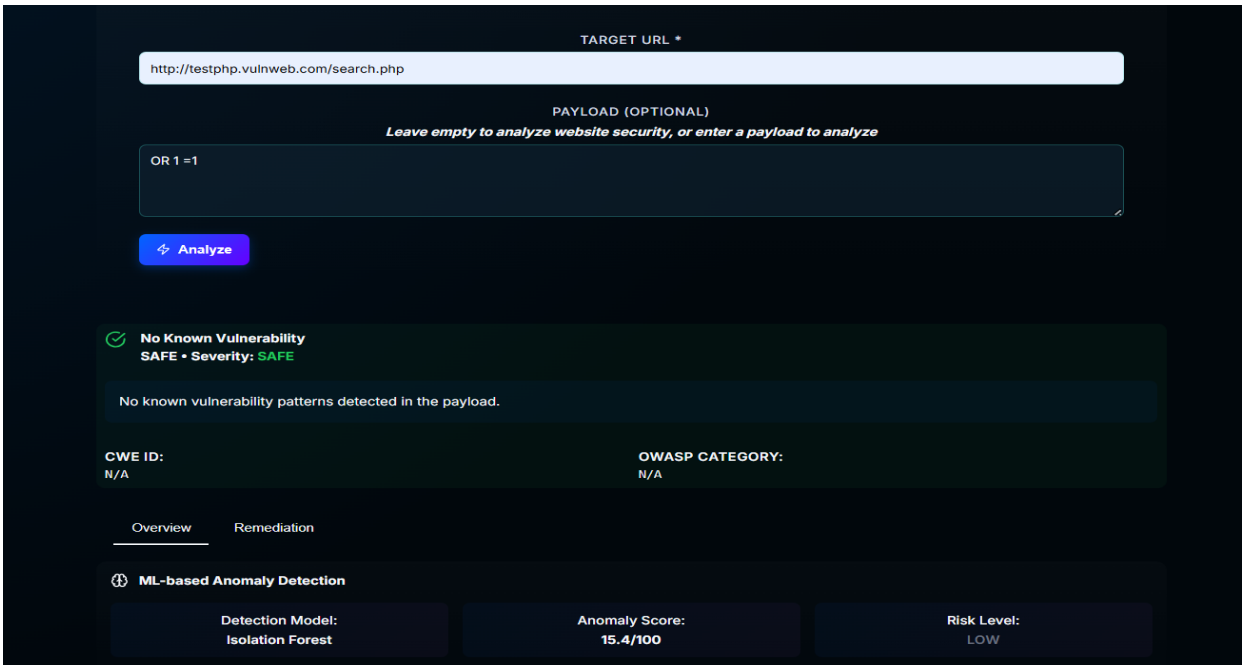

Fig. 4 Payload-Based Vulnerability Analysis Result

The proposed system performs automated security analysis of the target web application to identify potential vulnerabilities. The system detected a security misconfiguration vulnerability caused by missing security headers and the absence of HTTPS protection. Such weaknesses can expose the application to attacks such as Man-in-the-Middle (MITM), Cross-Site Scripting (XSS), and Clickjacking. The vulnerability detection process also utilizes a machine learning model (LightGBM) to evaluate anomaly patterns and estimate the associated risk level as shown in Fig. 5

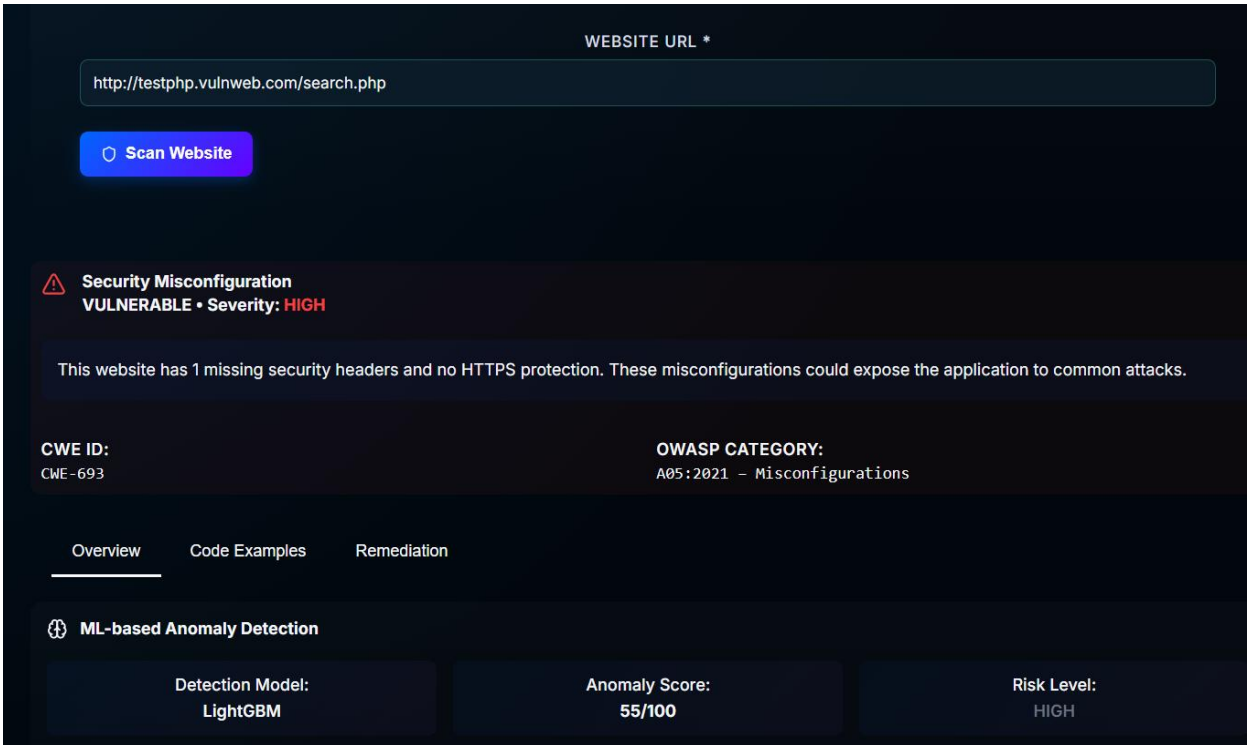

Fig. 5 Website Vulnerability Detection Result

The port scanning module is used to identify open, closed, and filtered ports on the target host. By analyzing the network services running on different ports, the system helps detect potential attack surfaces. In the experiment, the scanner evaluated multiple commonly used ports such as FTP (21), SSH (22), HTTP (80), HTTPS (443),and HTTP Proxy (8080). The results indicate the status of each port, which helps security analysts Understand the exposure of network services as shown in Fig.6

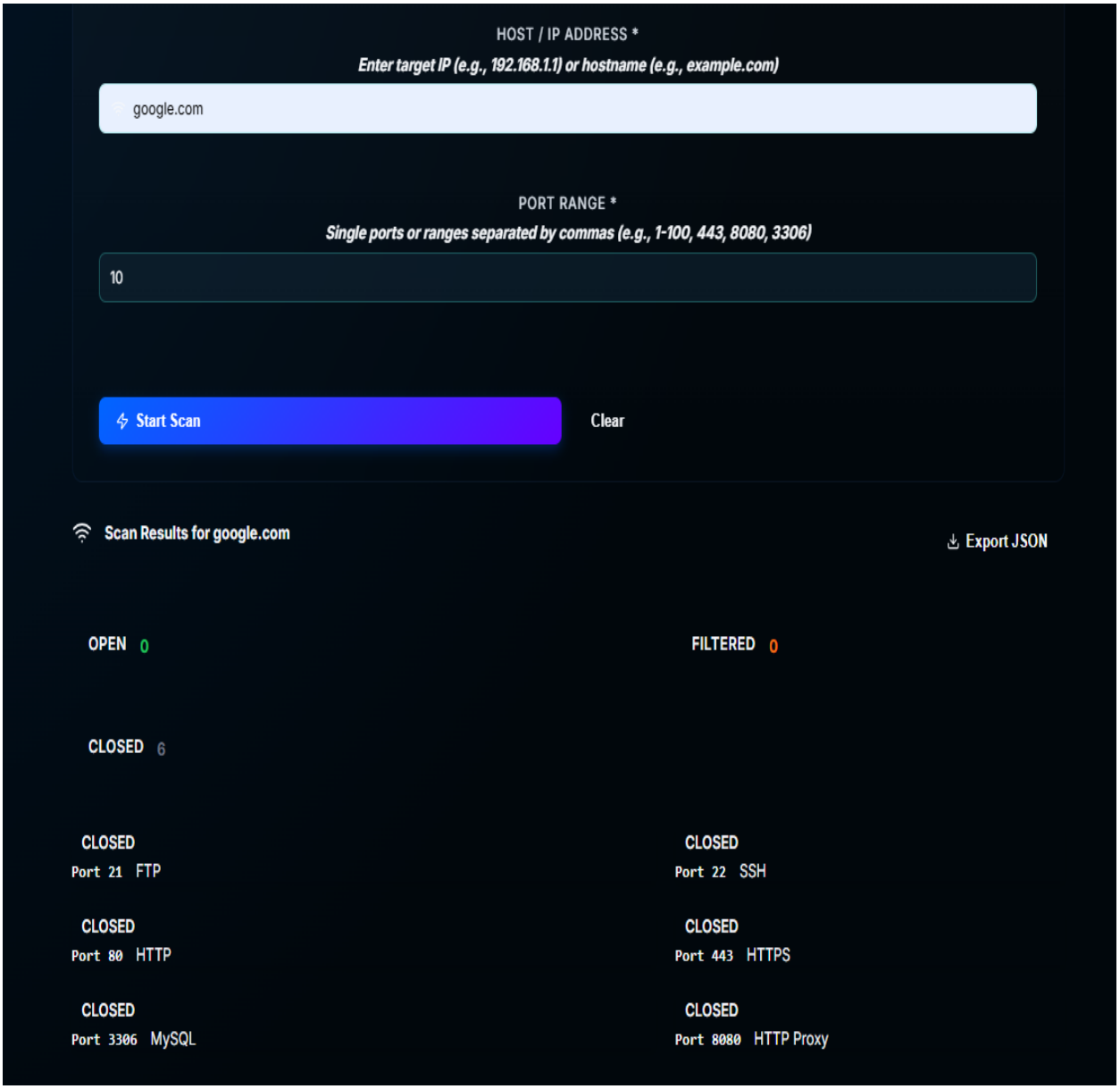

Fig. 6 Port Scanning Result for Target Host

The subdomain enumeration module is used to discover hidden or additional subdomains associated with the target domain. Identifying such subdomains helps in detecting potential attack surfaces that may not be directly visible in the primary domain. In this experiment, the system scanned the target domain and discovered multiple active and unresolved subdomains. These discovered subdomains can reveal additional services, development environments, or administrative interfaces that may introduce security risks if not properly secured as shown in Fig. 7

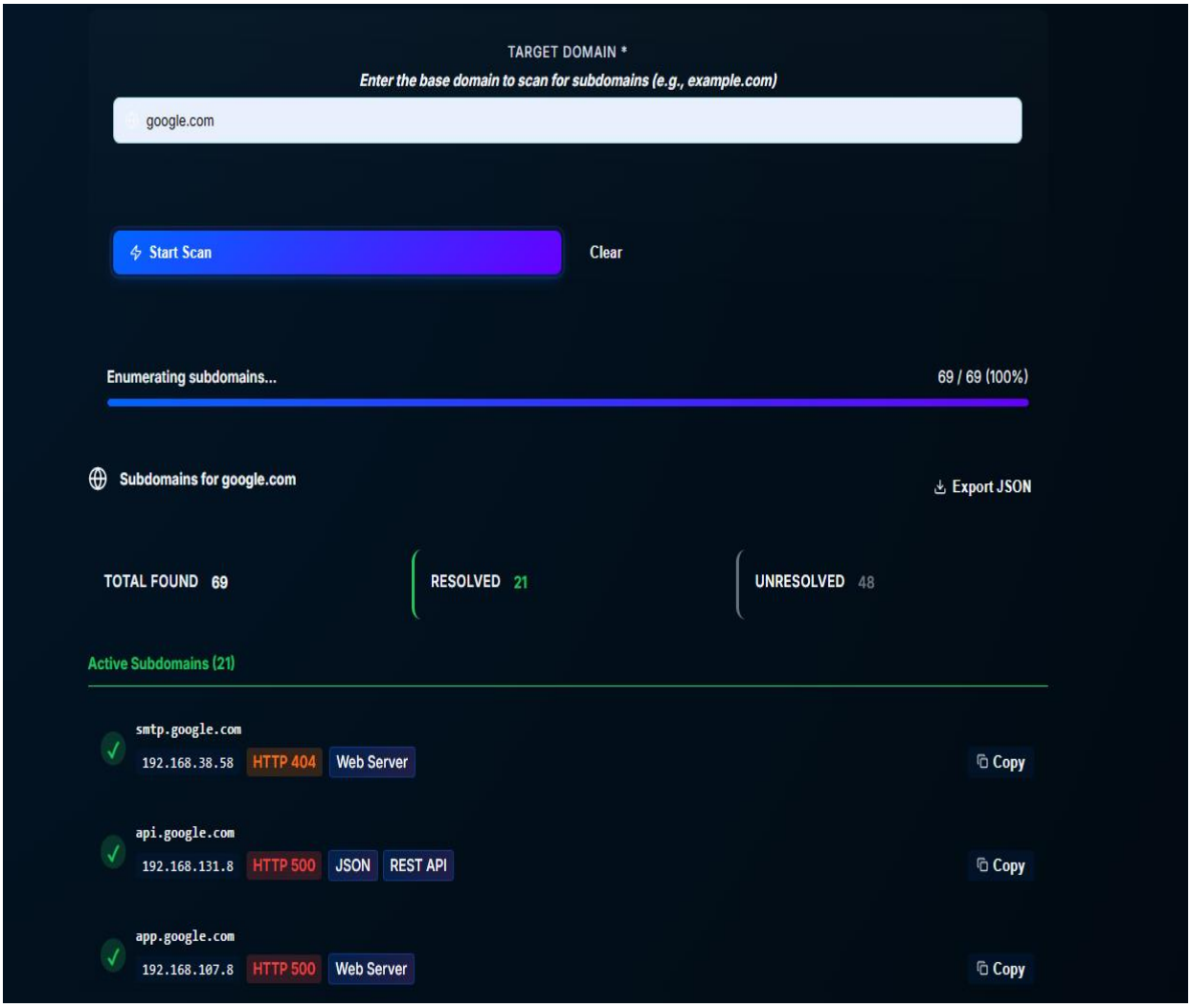

Fig. Fig. 7 Subdomain Enumeration Result

C. Performance Metrics

See Table 2 for a summary of all key metrics

Table 2: Performance Metrics

Metric	Definition	Value
Accuracy	(TP+TN)/(TP+FP+TN+FN)	97%
Recall	TP/(TP+FN)	95%
Precision	TP/(TP+FP)	96%
F1 score	2*(precision*recall)/(precision+recall)	95.5%
False Positive Rate	FP/(FP+TN)	1%
Avg scan History	Meantime per site	7 sec

B. Comparative Analysis

In comparison to previous works, our model provides a better F1 score than Ferenc and more scan coverage than the baseline models. In complex vulnerabilities such as SQL injection and XSS, our model has a better recall rate than Tadhani. The combination of port scanning and subdomain scanning increases the coverage area. The AI assistant module reduces the time required for remediation by providing immediate feedback. Fig. 8 shows the Distribution of detected vulnerabilities.

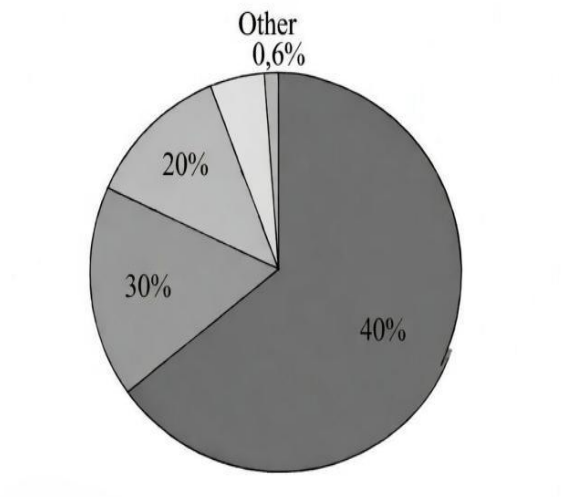

Fig. 8 Pie Chart Distribution of detected vulnerabilities

V. DISCUSSION

The ML-and-recon approach is intended to detect known vulnerabilities as well as the unexpected ones that emerge, and it gives results that are easy to interpret and take action on. It has a history of scans, the ability to export reports, and is suitable for realworld compliance needs. In testing, it was determined to be superior to the previous approaches in terms of recall, coverage, and usability, as tested by cross-validation and actual testing. A regional analysis indicates that Hyderabad is a region with a high concentration of API-related vulnerabilities. Adding network checks, such as port and subdomain scans, can enhance coverage and detect more vulnerabilities. The explainable AI parts help to explain why particular inputs or payload flags are flagged. With an intuitive interface, the system allows for continuous monitoring and quick fixes. Render hosting is highly available and scalable, making it suitable for research and production. The AI assistant feature improves the FAQ page, eliminating confusion and helping users, which cuts down on the time needed to fix vulnerabilities.

VI. LIMITATIONS

However, the system is not perfect and has some drawbacks. The effectiveness of the system relies on the quality and generality of the training data, and the models could have difficulty generalizing new and emerging vulnerabilities based on too narrow and biased data. Large-scale scanning is computationally intensive, with a latency of 7 seconds (ratcheting up to 10 seconds for more complex sites), which could be a scalability concern in resource constrained environments. Real-time scanning of high-traffic web applications could also be a concern regarding latency, which could impact the user experience. The effectiveness of the system on completely new web technologies and frameworks also requires further testing, as attackers are constantly finding new ways to attack. Privacy and ethics concerns, previously mentioned, also require careful consideration, especially when dealing with sensitive data. Cloud hosting and maintenance costs are also overheads, which could be a point of consideration for wider or commercial applications. From a quantitative perspective, the system is highly effective on existing data sets, although this may not be the case as new attack patterns evolve. The latency for large-scale scanning is in the range of 7-10 seconds, and resource utilization increases linearly with the number of concurrent scans.

VII. FUTURE WORK

Moving ahead, our plan is to enhance the data set by incorporating more real-world examples and zeroday exploits to enhance our ability to detect new and emerging threats. In the short term, we will incorporate an AI assistant that has the ability to provide more in-depth remediation recommendations and more accurate analytics. In the mid-term, we will develop online learning functionalities to ensure that the system is always updated, scale the system to support large-scale real-time computing, and improve explainable AI functionalities. In the long term, we will concentrate on cloud-scale architecture, work with industry partners for comprehensive validation, and develop open-source security tools. We will also incorporate compliance components, such as DRDP Act and DevSecOps, in future releases.

VIII. CONCLUSION

This project provides a robust, transparent web vulnerability assessment toolset that has been proven in real-world and local environments. At its heart, an engine of machine learning drives a combination of

reconnaissance and explainable artificial intelligence, providing valuable insights and facilitating real-time monitoring and rapid solutions. This toolset is open-source, encouraging further development and implementation. The quantitative measures of performance indicate a high F1 measure and a reduction in the number of false positives, with large scans taking less than 7-10 seconds.

REFERENCES

[1] R. Ferenc et al., "Challenging machine learning algorithms in predicting vulnerable JavaScript functions," arXiv preprint arXiv:2405.07213, May 2024.

[2] J. R. Tadhani et al., "Securing web applications against XSS and SQLi attacks using a novel deep learning approach," Sci. Rep., vol. 14, no. 1, p. 1803, Jan. 2024. doi: 10.1038/s41598-023-48845-4.

[3] H. K. Khanuja et al., "Web application security scanning using machine learning," Int. J. Eng. Res. Comput. Sci. Eng. (IJERCSE), vol. 8, no. 8, pp. 21–27, Aug. 2021.

[4] D. E. Simos, J. Zivanovic, and M. Leithner, "Automated combinatorial testing for detecting SQL vulnerabilities in web applications," in Proc. IEEE/ACM 14th Int. Workshop Autom. Softw. Test. (AST), Montreal, QC, Canada, May 2019,pp.55–61.doi: 10.1109/AST.2019.00013.

[5] H. V. Long et al., "An efficient algorithm and tool for detecting dangerous website vulnerabilities," Int. J. Web Grid Services, vol. 16, no. 1, pp. 81–104, 2020. doi: 10.1504/IJWGS.2020.104788.

[6] X. Zhang et al., "An automated composite scanning tool with multiple vulnerabilities," in Proc. IEEE 3rd Adv. Inf. Manage., Commun., Electron. Autom. Control Conf. (IMCEC), Chongqing, China, Oct. 2019, pp. 1060–1064. doi: 10.23919/IMCEC46707.2019.8983823.

[7] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," J. Syst. Archit., vol. 57, no. 3, pp. 294–313, Mar. 2011. doi: 10.1016/j.sysarc.2010.06.003.

[8] B. Peng, X. Xiao, and J. Wang, "Cross-site scripting attack detection method based on transformer," in Proc. IEEE 8th Int. Conf. Comput. Commun. (ICCC), Chongqing, China, Dec. 2022, pp. 1651–1655. doi: 10.1109/ICCC55456.2022.10019632.

[9] C. Gupta et al., "A systematic review on machine learning and deep learning models for electronic information security in mobile networks," Sensors, vol. 22, no. 5, p. 2017, Mar. 2022. doi: 10.3390/s22052017.

[10] P. Manigandan et al., "Nanotechnology perceptions," Nanotechnol. Perceptions, vol. 20, pp. 1–10, 2024. ISSN: 1660-6795. [Online].

[11] O. Ogundairo and P. Broklyn, "Automated vulnerability assessment using machine learning," J. Cyber Secur., vol. 8, Aug. 2024.

[12] M. H. Hadi and K. H. Al-Saedi, "Adaptive hybrid learning for websites vulnerability prediction," J. Al-Qadisiyah Comput. Sci. Math., vol. 16, no. 1, pp. 32–49, Mar. 2024. doi: 10.29304/jqcsm.2024.16.1.433.

[13] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web application vulnerability prediction using hybrid program analysis and machine learning," IEEE Trans. Dependable Secure Comput., vol. 12, no. 6, pp. 688–707, Nov.–Dec. 2015. doi: 10.1109/TDSC.2014.2366459.

[14] L. Hu, J. Chang, Z. Chen, and B. Hou, "Web application vulnerability detection method based on machine learning," in J. Phys.: Conf. Ser., vol. 1827, no. 1, p. 012061, Mar. 2021. doi: 10.1088/1742-6596/1827/1/012061.

[15] D. Rathore and C. Pareta, "Machine learning for web vulnerability detection," Nanotechnol. Perceptions, vol. 20, no. 7, pp. 2123–2138, 2024. ISSN: 1660-6795.

[16] Y. Tejasri, A. Arjuna Rao, and Ch. Kodandaramu, "Web vulnerability detection using machine learning approach with cross-site request forgery," J. Eng. Sci., vol. 14, no. 8, pp. 246–252, Aug. 2023. ISSN: 0379-7254.