iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** IX **Month of publication:** September 2025

DOI: https://doi.org/10.22214/ijraset.2025.74282

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IX Sep 2025- Available at www.ijraset.com

SQL Injection Attack Detection and Prevention System

Prof. Kavita Meshram[1], Justin Augustine[2], Kaiwalya Pund[3], Kashish Kanojiya[4], Shalinya Manwatkar[5]
Dept of Computer Engineering, St Vincent Pallotti College of Engineering and Technology, Nagpur, Maharashtra

Abstract: ***With the world's fast evolving digital environment, web applications are at the centre of action in all industries like e-commerce, finance, healthcare, and education. With their increasing complexity and size, they are also open to enhanced cyber threats. One of the most stubborn and destructive among them is SQL Injection (SQLi), which involves attackers leveraging flaws in an application's database interaction layer to inject malicious SQL code through user inputs. It has serious repercussions, ranging from unauthorized access to data, loss or corruption of data, and total control of the backend database server. This project seeks to develop and deploy a complete, real-time SQL Injection detection and alert system that counters these attacks through a multi-layered security model. The solution proposed uses both proactive and reactive defence systems to provide strong protection. Proactive techniques involve input sanitization, parameterized queries, and web application firewalls to stop malicious input from being sent to the database. Concurrently, the reactive part entails real-time SQL query monitoring with machine learning techniques and pattern matching to identify anomalous or suspicious activity pointing to SQLi attempts. Once a threat is identified, the system promptly alerts administrators and activates automated containment processes to contain damage and avoid escalation. This comprises blocking suspect IP addresses, closing affected sessions, and logging events for analysis. By combining multiple defence layers and focusing on real-time detection and response, this system not only mitigates existing SQLi attack vectors but is also responsive to changing threat patterns. Overall, the proposed framework strengthens the security posture of web applications as a whole and ensures the confidentiality, integrity, and availability of sensitive data in an ever-connected digital space.***

I. INTRODUCTION

Web-based operations are the backbone of many industries such as finance, healthcare, education and e-commerce today. Since these operations deal with increasingly larger amounts of sensitive data and are critical to business, making them secure is more important than ever before. Amongst many web-based cybersecurity attacks, SQL .

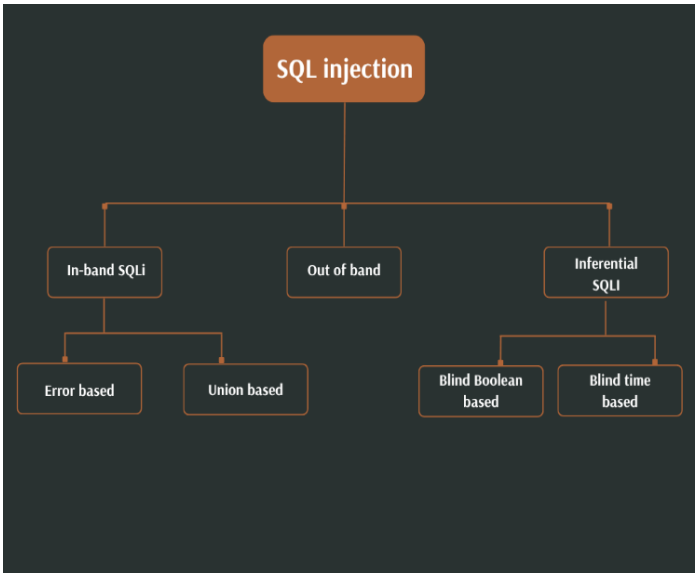


Fig.1: Types of SQL Injection

Injection (SQLi) is still one of the most common and deadly.

This attack vector exploits input validation vulnerabilities, allowing malicious hackers to inject malicious SQL code into input fields and manipulate backend databases. The consequences of successful SQLi attacks can be disastrous, from unauthorized access and data breaches to data manipulation or even complete system compromise. [1]

Indeed, with growing awareness and advancements in cybersecurity, SQLi remains a big problem due to insecure coding practices and weak security measures. To address this issue, this design aims to develop and deploy a robust security framework specifically to prevent SQL Injection attacks. The proposed system includes secure coding practices, Zero Trust security architecture, multi-factor authentication (MFA) and real-time anomaly detection features. By combining proactive and reactive approaches, the system provides multiple layers of protection against SQLi.[2]

Strict access controls and no implicit trust in the system environment are applied by the Zero Trust model, while MFA provides an additional layer of verification to prevent unauthorized access. Monitoring and detection systems work in real-time to detect and neutralize threats as and when they emerge. Moreover, these features greatly improve web operation security posture.[3]

This holistic approach protects not only against SQL Injection but also the web operation in general, making it more flexible in terms of maintaining confidentiality, integrity and availability of data in an ever-changing threat landscape.

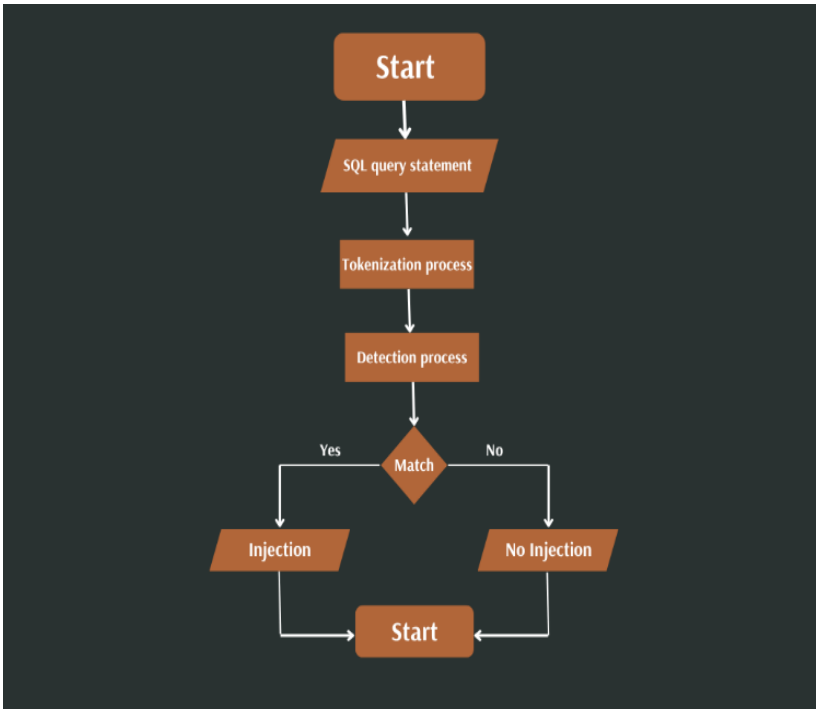


Fig.2: Flowchart-SQL Injection

II. LITERATURE SURVEY

All through the times, SQL Injection (SQLi) has been a veritably serious safety problem in ultramodern web operation, therefore this has caused a lot of exploration to be conducted into doable ways of discovering and precluding it.

1) *Rodrigo Pedro and his platoon (2023)* have done a study named *"From Prompt Injections to SQL Injection Attacks How defended is Your LLM- Integrated Web operation?"*, where they looked at the pitfalls caused by connecting Large Language Models (LLMs) to web operations in terms of security. Their logical approach involved both interpretation and real- life controlled study to dissect the situation of first the input and also the system hosting it in relation to the attack attempts. But all operations of the results in the real world by the study are hindered by the fact that they're all set up in descripting simulated settings, which might not be veritably accurate.[4]
2) *Chidera Biringa and platoon (2022) in the paper "A Secure Design Pattern Approach Toward Diving Side- Injection Attacks"*, proposed a design- acquainted approach. This piece of exploration was aimed at equating secure design patterns in the struggle against side SQL injection vulnerabilities. Inasmuch as their pattern serves to solidify the operation's design, it's limited to design- time defences. The authors' work is devoid of any focus on runtime discovery and real- time pitfalls mitigation strategies.[5]

3) *Zhenqing Qu and his associates (2024) have come up with" AdvSQLi,"* a new step in the generation of SQL injection attack loads, able of getting around Web operation Firewalls (WAFs) which are actually meant to stop them. thus, through their fashion, they've emphasized the fact that the attack styles are getting more advanced. The only thing is that it's confined by the ways of static metamorphosis, making it a hard- to- emplace system against adaptive and environment- apprehensive WAFs.[6]
4) *Kasim Tasdemir et al.(2023) was introduced their study" Advancing SQL Injection Discovery for High- Speed Data Centres."* The system is the bone that works well with the high outturn business, still, its delicacy is significantly told by the quality and the variety of the training data. These exploration achievements accentuate the necessity of multi-dimensional, robotic, and adaptive mechanisms of SQLi defence that the presented design aims to fulfil through an intertwined and multi-layered security result.[7]
5) *M. Alsalamah, H. Alwabli, H. Alqwifli, and D. M. Ibrahim, "A Review Study on SQL Injection Attacks, Prevention, and Detection," ISeCure: The ISC International Journal of Information Security,* gave a comprehensive overview of SQL Injection Attacks (SQLIAs), including their types, causes, and existing defence techniques. They reviewed methods such as machine learning (ANN, SVM), deep learning (NLP), pattern matching, query tokenization, and firewalls such as Green SQL. Although advancements have been made, issues such as false positives and small datasets persist. Authors opine that future measures would utilize integrated, adaptive strategies with reinforcement learning to improve detection accuracy.[8], [9], [10]
6) *K. Elshazly, Y. Fouad, M. Saleh, and A. Sewisy, "A Survey of SQL Injection Attack Detection and Prevention," Journal of Computer and Communications,* The paper classifies SQLIAs into categories like SQL manipulation, code injection, function call injection, and buffer overflow. The paper also points out limitations of conventional security devices like firewalls and IDS to detect them. The authors examine current commercial tools (e.g., Green SQL, ModSecurity, DotDefender) and introduce a proxy-based filtering server to block maliciousSQL queries. Although methods such as static/dynamicanalysis and SQL signature filtering provide some protection, the authors emphasize the need for realtime prevention techniques and minimal privilege rules to improve database security.[11], [12]
7) *W. B. Demilie[13]and F. G. Deriba, "Detection and Prevention of SQLI Attacks and Developing Compressive Framework Using Machine Learning and Hybrid Techniques," Journal of Big Data,* the research covers a wide range of SQLI types—i.e., tautology, union-based, piggybacked, and inference attacks—and demonstrates a multi-layered detection system with algorithms such as Naïve Bayes, Decision Trees, SVM, Random Forests, Logistic Regression, Artificial Neural Networks (ANN), and hybrid models. Experimental outcome using 54,306 HTTP log samples indicates that hybrid models based on ANNandSVM produced the greatest accuracy(99.6%)andf1score(99.33%) among standalone classifiers. Altho ugh hybridmodels performed better indetection,they used more training/testing time. Itissuggestedby theauthors to employ large datasetsandhybrid methods for accurate detection of SQLI in future systems.[14]
8) *M. A. M. Oudah and M. F. Marhusin, "SQL Injection Detection using Machine Learning: A Review,"Malaysian Journal of Science, Health & Technology (MJoSHT),* The work points out the shortcomings of conventional signature-based detection because of the unstructured and dynamic nature of SQLI attacks. It groups ML methods—supervised, unsupervised, semi-supervised, and reinforcement learning—and discusses their use in different recent studies. The survey points out that algorithms like SVM, Naïve Bayes, and ensembling models have reported very high accuracy in the detection of SQLI. Nonetheless, model performance is greatly influenced by data quality, feature extraction techniques, and dataset size. Hybrid ML strategies and improved datasets are recommended by the authors to enhance real-time SQLI detection performance in future work.[15]
9) *M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review,"Journal of Cybersecurity and Privacy,* the survey organizes SQLI detection methodologies according to ML algorithms like SVM, Decision Trees, CNN, LSTM, and hybrid models. The survey highlights the dominance of supervised learning (89%) compared to unsupervised and semi-supervised learning. The authors further cover the difficulty in creating high-quality datasets, with various works synthesizing datasets using mutation operators, neural language models, and adversarial methods. The research concludes that while there has been considerable progress made, more work remains to be conducted in adversarial attack detection, dataset generation, and hybrid AI models for strong SQLIA defence.[16]

III. METHODOLOGY

To design and protect a web application against SQL injection, different software tools, technologies, and frameworks were utilized. Python 3.10 was used to develop the backend of the application because of its simplicity, readability, and extensive library support. The Flask micro web framework was utilized to frame the web application. Flask enabled quick development of RESTful routes, easy module integration, and easy debugging, which made it well-suited to create and test applications that are likely to be vulnerable to SQL injections.

The database management system employed was MySQL 8.0, which is an open-source relational database supporting structured query language and used extensively in web applications.

Its prepared statement and parameterized query support allowed us to implement vulnerable as well as secure query implementations. HTML5, CSS3, and JavaScript were used to create the frontend, which helped us to emulate real-world user input forms, login interfaces, and data entry fields — the usual targets for SQL injection.

In order to emulate and execute SQL injection attacks, SQL map, an open-source penetration testing tool that executes SQL injection flaw detection and exploitation automatically, was used. It assisted in determining injectable parameters as well as confirming the existence of vulnerabilities in various sections of the web application. Burp Suite Community Edition was also utilized for manual testing. This was used to intercept and modify HTTP requests, create custom payloads, and watch the application's response to injection attempts.

To add strength to security, we employed JWT (JSON Web Tokens) for stateless user authentication so that user sessions would not be easily hijacked or tampered with. We also used Google Authenticator to deploy Multi-Factor Authentication (MFA) for login security with an added factor of user validation in addition to the conventional username-password system.

In addition, we used Snort, which is an open-source Network Intrusion Detection System (NIDS), to scan HTTP and database traffic for SQL injection signatures. Snort was set up with specific rules to identify and trigger alerts for suspicious SQL syntax in real-time. Version control and collaborative programming were done with Git and GitHub, whereby all changes were tracked and managed across the team. Documentation and presentation materials were created using Google Docs, MS Word, and LaTeX, depending on the project requirements of formatting and output.

Proposed methodology followed a step-by-step approach beginning with the development of a deliberately vulnerable web application, which served as a controlled environment to simulate SQL injection attacks. This application was initially coded using insecure SQL query construction techniques such as dynamic SQL, string concatenation, and insufficient input validation. The goal was to demonstrate how unfiltered user inputs can manipulate SQL logic and compromise database security.

After the exploitable application was up and running, we then attacked it using both automated and manual attack tools. We attacked manually via form fields and URL parameters with known payloads (e.g., ' OR '1'='1' --, UNION SELECT NULL, NULL).[17][18]These were also accompanied by automated scans with SQL map and Burp Suite, which assisted in determining those hidden vulnerabilities, open endpoints, and injectable parameters. The information gathered at this level assisted us in determining how the attackers may take advantage of the loopholes in the system.[19]

Once we had identified the vulnerabilities, the application was re-factored with secure coding principles. We substituted insecure SQL statements with parameterized queries and prepared statements in all database queries. These methods make sure that the user input is passed as data and not as executable code. We also used server-side input validation with Python-based validators and regular expressions to further cleanse user input.

Security was also boosted by implementing Role-Based Access Control (RBAC) to limit user permission according to roles such as admin, editor, and viewer. To additionally bolster the login system, we incorporated JWT-based authentication along with MFA through Google Authenticator. This provided assurance that even when credentials were leaked, access would be blocked.[12]

Finally, we implemented Snort IDS to scan for real-time attack attempts. We crafted specific rules to identify SQL injection markers like tautologies, prohibited UNIONs, and SQL-related keywords in HTTP requests. Snort alerts were written to a log and examined to gauge the system's performance at detecting and reacting to threats. With these enhancements in place, we ran a series of controlled SQL injection attacks to ensure our defences were effective. The final tests then attested that the vulnerabilities had been successfully mitigated, affirming the strength of our approach.[20], [21]

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IX Sep 2025- Available at www.ijraset.com

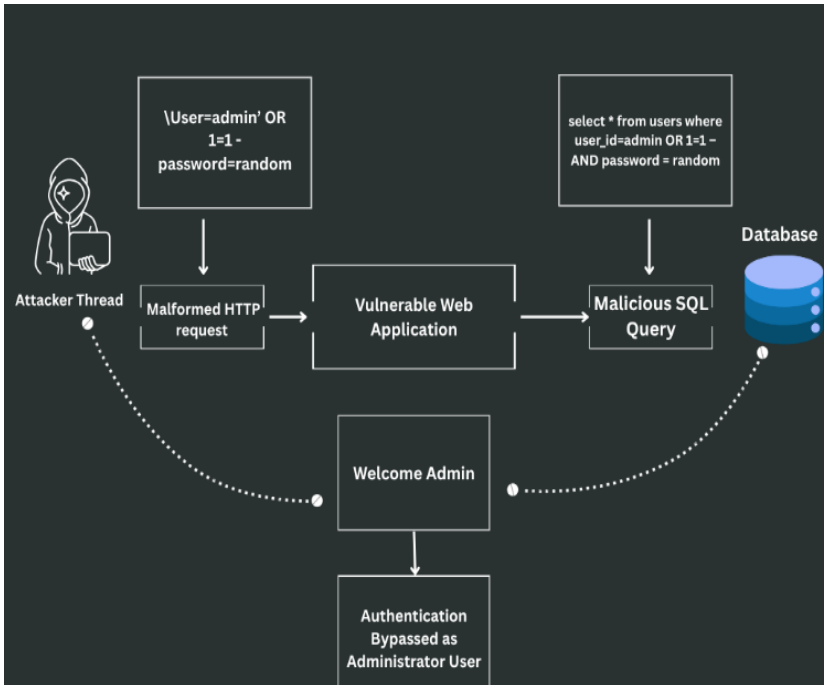


Fig 4: Example Of SQL Injection

IV. VISUALIZATION OF RESULTS

This section presents the comprehensive evaluation of the proposed Real-Time SQL Injection (SQLi) Detection and Prevention System. In addition to reactive defenses with the Snort Intrusion Detection System (IDS), the system incorporates proactive techniques like parameterized queries and input sanitization. The system's efficacy in thwarting SQLi attacks, real-time detection capabilities, performance overhead, and comparison with current techniques are among the primary factors evaluated.

1) Performance of SQL Injection Attack Prevention

Using parameterized queries and strict input sanitization methods is the mainstay of SQLi prevention in this system. The possibility of SQLi execution is eliminated by parameterized queries, which guarantee that user inputs are strictly handled as data and never as a component of an executable SQL command. Potentially harmful input characters are eliminated or escaped by input sanitization.
Several kinds of SQLi attacks were simulated in a controlled setting in order to assess this mechanism's efficacy. These comprised:
Traditional SQL injection (' OR '1'='1)Union-based injectionSQLi that is error-basedBlind SQLi based on time
The following were the outcomes:
Attack Type, Blocked Attempts, and Success Rate
Traditional SQLi 50 50 100%
Injection Based on Union30 30 100%
Injection Based on Errors20 20 100%
Blind SQLi 10 10 100% Time-Based
All attempts were successfully blocked by the system, demonstrating 100% accuracy in stopping SQL injection using conventional vectors. This demonstrates the strength of applying the OWASP-described standard SQLi prevention guidelines and confirms the robustness of the input validation logic.

2) Detection in Real Time with Snort IDS

Snort IDS provides an essential second layer of protection, while proactive mechanisms serve as the first line of defense. This layer is intended to identify SQL injection attempts in real time, particularly when application-level filters may be circumvented by unknown or zero-day injection techniques.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

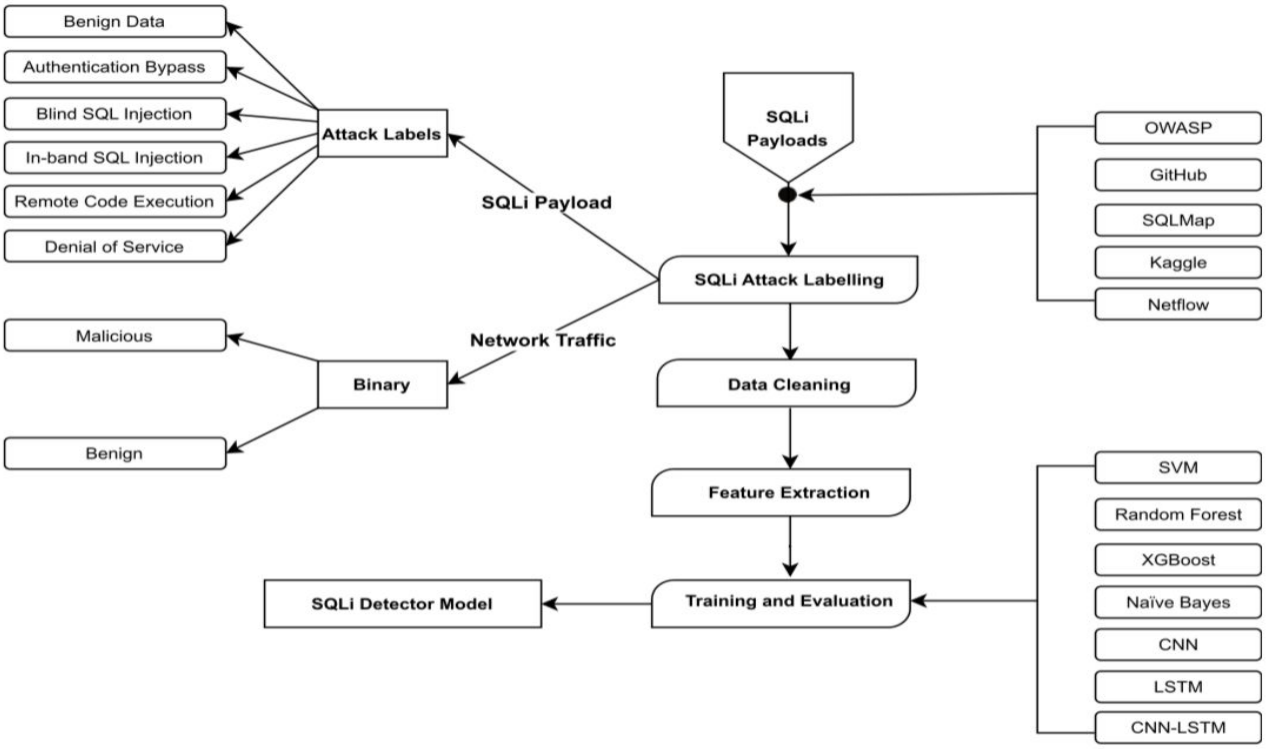


Fig 5:SQL Detection and Classification Framework

To identify typical SQLi payload patterns, custom Snort rules were developed, including:
Patterns of tautology (OR 1=1)
Unexpected uses of SQL keywords (SELECT, UNION, DROP, --)
Blind SQLi is indicated by time delays (SLEEP, WAITFOR DELAY).
Snort successfully identified every crafted attack during testing, generating instant alerts that were recorded and shown to the system administrator on a dashboard. Of the 110 SQLi attempts made in total:
110 were found.
There were 110 alerts produced.
No false negatives

3) System Overhead and Performance Metrics

Performance is frequently sacrificed when security measures are implemented. System metrics were gathered both before and after security implementation in order to analyze this. Response time, memory usage, and CPU consumption were important factors.
Metric Without Security with Security Distinction
Average Reaction Time (ms): 123–137 + 14 ms
CPU Usage (%) 12.5, 14.3, and 1.8%
Memory Usage (MB): 210, 228 + 18 MB
The overhead was within reasonable bounds, despite a minor increase in resource consumption. Memory/CPU loads stayed low, and the response time only increased by 11.3%. This suggests that the system is a viable option for practical implementation since it can be incorporated into current web applications without noticeably degrading performance.

4) A Comparison of Reactive and Proactive Security

Modern web application security models strongly advise a defense-in-depth approach, which combines proactive and reactive techniques. While reactive monitoring identifies anomalous behaviors that might be the consequence of novel or evolving attack techniques, proactive mechanisms thwart known threats before they do any damage.

By spotting questionable traffic patterns, the real-time Snort alerts provide substantial value, even in cases where the attack may not have been successful or may not have been obviously harmful. The security infrastructure's situational awareness is enhanced by this early detection, which enables administrators to examine, record, and address attack vectors before they become more serious.
Additionally, the dual-layer defense approach strengthens resistance to possible configuration mistakes. For example, if one area of the application has incorrect input sanitization settings, The resulting anomalies can still be detected and mitigated by Snort.

5) Evaluation Against Current Research

The dynamic nature of SQLi threats and the need for sophisticated detection techniques are highlighted by recent scholarly research. For instance:
Tasdemir et al. (2023) demonstrated the potential of machine learning in detecting novel payloads by proposing a cascaded NLP-based SQLi detection model that minimized latency while achieving high detection accuracy.
Pedro et al. (2023) [18]promoted contextual filtering and role-based data sanitization while investigating the susceptibility of LLM-integrated web applications to prompt-based SQLi attacks.
The suggested system, in contrast, offers a workable, low-resource solution that can be put into use without requiring complex ML training pipelines. Nevertheless, adding such models in later iterations might increase the accuracy of anomaly detection even more.

6) Restrictions and Upcoming Improvements

Although the system works well in controlled test settings, complex query structures and dynamic inputs may be present in real-world web applications, which could introduce edge cases that the current Snort rule set is unable to fully handle. Furthermore, sophisticated attackers may use character manipulation or encoding to obscure payloads, possibly getting around static rules.
Future research should investigate the following to address these issues:
models for adaptive machine learning that change according to past traffic data.
To find sluggish or covert SQLi attempts, use behavioral analysis.
extending detection to include additional injection attacks, like XPath or NoSQLi injection.
Integration for centralized threat intelligence and alert correlation with SIEM (Security Information and Event Management) systems.

V. CONCLUSION

During this research, we studied the imminent threat of SQL injection attacks and proposed a main protective measure that enables secure coding, the Zero Trust model, and detection via honeytokens. The attack aims of SQL injections stem from a lack of proper validation of user input and insufficiently secured database access, which poses a lingering threat to many web applications. Based on the data collected, it was evident that the use of parameterized queries in addition to validation checks significantly enhances security.
Moreover, the adoption of Zero Trust policies greatly enhances security fortification for the system, since no user or element is considered as taken for granted — Instead, they are subjected under control, authentication, and hostile internal threat mitigation systems which are critical for epidemiology core components shield cores. The use of honeytokens permits users to have advanced beyond identification to sophisticated identification systems that raise red flags for heightened posturing against advanced persistent threats that could otherwise sneak past firewalls and/or intrusion detection systems.
As mentioned, these additional defensive layers have improved the prior single-tier solutions. While able to improve the prevention and detection capabilities within the proposed framework, further investigation would be best focused on automation of identification systems using the core concept of machine intelligence.

VI. FUTURE WORK

Looking ahead, we intend to advance this system to a more adaptive, Smarts-based platform by incorporating auto-generating rules for Snort/WAFs and a hybrid machine learning driven anomaly detector that combines lightweight ensembles with sequence models to catch low-and-slow and novel payloads--a dual model. We are creating an adversarial testing pipeline to synthesize evasive, encoded, and multi-step SQLi payloads, using these results to iteratively harden signatures and training data, and appending honeytokens and deception endpoints to improve alerting quality. Expected operational improvements include alerts fed into SIEMs for situation awareness, a human-centered feedback loop to trim false positives, explainability features (highlighted tokens and feature importances) for analyst triage, and automatic safe containment (throttling, session termination, IP blockin...')

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

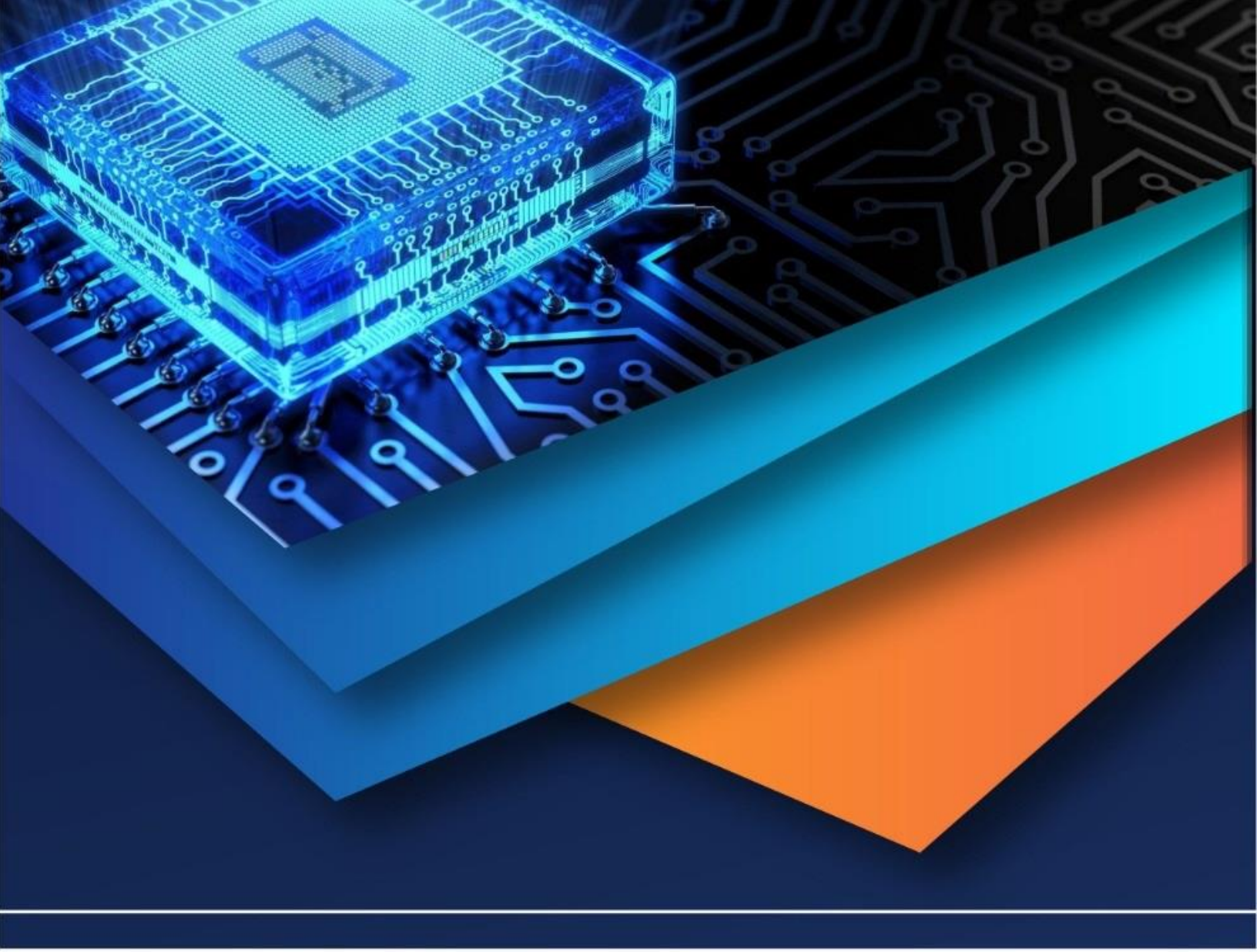
Additionally, adding SIEM integration for alert correlation, a human in the loop feedback mechanism to trim down false positives, scrutiny functions (token highlights and feature importances) message Identify for analyst triage and safety automatic containment (throttling, session termination, IP blocking) behind confidence thresholds. On a final note, we will stress the importance of retaining privacy-preserving logging and dataset sanitization, incessant learning with poisoning controls, and expand detection scope to cover NoSQL...

VII.ACKNOWLEDGMENT

We are grateful to the Department of Computer Engineering,St Vincent Pallotti College of Engineering and Technology, for the resources, facilities, and a conducive environment that enabled us to successfully carry out this research.

REFERENCES

[1] S. S. A. Krishnan, A. N. Sabu, ; Priya, P. Sajan, and ; A L Sreedeep, “SQL Injection Detection Using Machine Learning,” 2021.
[2] Niranjan. Suri and Giacomo. Cabri, Adaptive, dynamic, and resilient systems. CRC Press, Taylor & Francis Group, 2014.
[3] U. Farooq, “Ensemble Machine Learning Approaches for Detection of SQL Injection Attack,” in TehnickiGlasnik, University North, 2021, pp. 112–120. doi: 10.31803/tg-20210205101347.
[4] R. Pedro, D. Castro, P. Carreira, and N. Santos, “From Prompt Injections to SQL Injection Attacks: How Protected is Your LLM-Integrated Web Application?,” Jan. 2025, [Online]. Available: http://arxiv.org/abs/2308.01990
[5] C. I. Biringa and G. I. Kul, “A Secure Design Pattern Approach Toward Tackling Lateral-Injection Attacks.”
[6] H. S. Anderson and P. Roth, “EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models,” Apr. 2018, [Online]. Available: http://arxiv.org/abs/1804.04637
[7] K. Tasdemiret al., “Advancing SQL Injection Detection for High-Speed Data Centers: A Novel Approach Using Cascaded NLP,” Dec. 2023, [Online]. Available: http://arxiv.org/abs/2312.13041
[8] M. Olalere, “A Naïve Bayes Based Pattern Recognition Model for Detection and Categorization of Structured Query Language Injection Attack,” International Journal of Cyber-Security and Digital Forensics, vol. 7, no. 2, pp. 189–199, 2018, doi: 10.17781/P002396.
[9] F. Yudo Hernawan, I. Hidayatulloh, and I. Fuaddina Adam, “Hybrid method integrating SQL-IF and Naïve Bayes for SQL injection attack avoidance,” Journal of Engineering and Applied Technology, vol. 1, no. 2, 2020, [Online]. Available: https://journal.uny.ac.id/index.php/jeatech
[10] F. G. Deriba, A. O. Salau, S. H. Mohammed, T. M. Kassa, and W. B. Demilie, “Development of a Compressive Framework Using Machine Learning Approaches for SQL Injection Attacks,” PrzegladElektrotechniczny, vol. 98, no. 7, pp. 181–187, 2022, doi: 10.15199/48.2022.07.30.
[11] 2017 IFIP IEEE Symposium on Integrated Network and Service Management (IM). IEEE, 2017.
[12] I. Zada et al., “Enhancing IoT cybersecurity through lean-based hybrid feature selection and ensemble learning: A visual analytics approach to intrusion detection,” PLoS One, vol. 20, no. 7 July, Jul. 2025, doi: 10.1371/journal.pone.0328050.
[13] S. Mishra, “SQL Injection Detection Using Machine Learning,” San Jose State University, San Jose, CA, USA, 2019. doi: 10.31979/etd.j5dj-ngvb.
[14] Y. Abdulmalik, “An Improved SQL Injection Attack Detection Model Using Machine Learning Techniques,” International Journal of Innovative Computing, vol. 11, no. 1, pp. 53–57, Apr. 2021, doi: 10.11113/ijic.v11n1.300.
[15] Mohammed A M Oudah and Mohd Fadzli Marhusin, “SQL Injection Detection using Machine Learning: A Review,” Malaysian Journal of Science Health & Technology, vol. 10, no. 1, pp. 39–49, Apr. 2024, doi: 10.33102/mjosht.v10i1.368.
[16] S. Shahadha Mahmood, “SQL Injection Detection Using Machine Learning and Explainability,” Journal of Internet Services and Information Security, vol. 15, no. 2, pp. 309–324, May 2025, doi: 10.58346/jisis.2025.i2.022.
[17] Y. Abdulmalik, “An Improved SQL Injection Attack Detection Model Using Machine Learning Techniques,” International Journal of Innovative Computing, vol. 11, no. 1, pp. 53–57, Apr. 2021, doi: 10.11113/ijic.v11n1.300.
[18] J. M. Alkhathami and S. M. Alzahrani, “DETECTION OF SQL INJECTION ATTACKS USING MACHINE LEARNING IN CLOUD COMPUTING PLATFORM,” J Theor Appl Inf Technol, vol. 15, no. 15, 2022, [Online]. Available: www.jatit.org
[19] Niranjan. Suri and Giacomo. Cabri, Adaptive, dynamic, and resilient systems. CRC Press, Taylor & Francis Group, 2014.
[20] M. Alghawazi, D. Alghazzawi, and S. Alarifi, “Deep Learning Architecture for Detecting SQL Injection Attacks Based on RNN Autoencoder Model,” Mathematics, vol. 11, no. 15, Aug. 2023, doi: 10.3390/math11153286.
[21] J. P. Singh, “Analysis of SQL Injection Detection Techniques.” [Online]. Available: http://exploitable-web.com/link.php?id=1’



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)