World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Check for updates

Enhancing website security through prevention of SQL and data inference injections

Raghavendra Sridhar *

Department of ECE, Visvesvaraya Technological University, Belagavi, Karnataka, India.

World Journal of Advanced Research and Reviews, 2022, 13(03), 673-678

Publication history: Received on 11 February 2022; revised on 24 March 2022; accepted on 29 March 2022

Article DOI: https://doi.org/10.30574/wjarr.2022.13.3.0238

Abstract

The widespread use of web applications for handling sensitive personal information makes them a primary target for cyberattacks. SQL Injection and Data Inference attacks, in particular, represent critical vulnerabilities that exploit an application's database layer. This paper proposes a multi-layered security framework to mitigate these threats through advanced optimization and encryption techniques. We introduce a query tree mechanism for real-time attack detection, designed to analyze the structure of user queries and identify malicious patterns. To further enhance security, we propose the implementation of Fully Homomorphic Encryption (FHE), an advanced cryptographic method that allows for computations on encrypted data. This dual approach of proactive detection and robust encryption aims to create a resilient defense, significantly improving the security posture of web applications against sophisticated data-level attacks.

Keywords: Web Application Security; SQL Injection; Data Inference Attacks; Query Tree Mechanism; Real-Time Attack Detection; Fully Homomorphic Encryption; Encryption Techniques; Cybersecurity Defense

1. Introduction

In today's digital era, web applications have become indispensable across a wide range of organizations and industries, including business, commerce, education, and healthcare. Their ability to provide reliable, efficient, and scalable solutions for online communication and data management has led to their widespread adoption. However, much like other rapidly evolving technologies, web applications and their underlying databases have attracted significant attention, not only from developers and legitimate users but also from malicious actors seeking to exploit system vulnerabilities for personal gain. The security of databases remains a pressing and ever-evolving concern, especially given the increasing frequency of reported breaches and cyber incidents. One of the most prevalent threats is SQL injection attacks (SQLIAs). In these attacks, adversaries manipulate database queries by injecting malicious SQL code into input fields, often at points where the application interacts directly with the database layer. Such attacks can result in unauthorized access, data leakage, or even complete compromise of sensitive information.

Typically, SQL injection attempts are characterized by the presence of suspicious SQL tokens and patterns within user inputs, which can be flagged as potential threats. In contrast, legitimate web requests generally contain expected data formats and structures generated by the application itself. Attackers often leverage publicly available resources and detailed online guides, making it easier for even non-experts to orchestrate a variety of web-based attacks, with SQL injection being among the most common and damaging.

To combat these threats, a range of traditional detection and prevention techniques have been developed. Many standard SQL injection attempts can be identified and mitigated using established procedures and security best practices. Furthermore, modern organizations increasingly rely on robust Database Management Systems (DBMS) to safeguard their data, especially in cloud storage environments. These systems are equipped with advanced access

* Corresponding author: Raghavendra Sridhar

control mechanisms, ensuring that only authorized users can access or manipulate sensitive information, thereby reducing the risk of unauthorized data breaches. Despite these advancements, the landscape of web application security continues to evolve. Ongoing vigilance, regular updates to security protocols, and the adoption of emerging technologies are essential to stay ahead of increasingly sophisticated cyber threats. By fostering a culture of security awareness and leveraging the latest tools and methodologies, organizations can better protect their digital assets and maintain the trust of their users.

2. Literature Review

The detection of SQL injection attacks presents a significant challenge in the field of cloud database security, primarily due to the extreme heterogeneity of the attack vectors employed by adversaries. To address this, novel methodologies are being developed that move beyond traditional signature-based detection. One such advanced approach involves representing SQL queries as a graph of tokens, where centrality measures are calculated and then used to train a Support Vector Machine (SVM) classifier. While our research primarily focuses on web applications built with PHP and MySQL, this graph-based methodology is inherently platform-agnostic and can be readily adapted to other technology stacks.A complementary technique, proposed by researchers such as Paul R. McWhirter et al., utilizes a Gap-Weighted String Subsequence Kernel. This method is designed to recognize shared character subsequences between different query strings, yielding a powerful similarity metric. An SVM algorithm is subsequently trained on this similarity measure, enabling it to accurately classify unknown queries by comparing them to a baseline of known legitimate and malicious queries.

A key advantage of these approaches is their ability to derive all necessary information directly from the query strings themselves, eliminating the need for additional data from the source application. This self-contained analysis simplifies deployment and reduces dependencies. Given that a successful SQL injection attack poses a grave threat to the integrity of the database, the web application, and the entire server infrastructure, the development of such sophisticated detection mechanisms is paramount. These attacks remain one of the most common and effective methods for manipulating data flow and exfiltrating or corrupting sensitive database content.

2.1. Problem Statement of Current Strategies

Despite ongoing efforts to secure web applications, SQL injection and data inference (DI) attacks continue to pose significant threats, particularly when attackers successfully exploit vulnerabilities within the web server's database. While numerous defensive mechanisms have been introduced in the literature to protect websites, these solutions often struggle to keep pace with the evolving landscape of attack techniques. In many cases, new SQL injection attacks are not entirely novel but are instead creative variations of previously known attack vectors. This constant evolution makes it challenging for traditional defenses to remain effective. Current filtering models typically focus on sanitizing user inputs by removing special characters, operators, and brackets from SQL queries. The end goal is to standardize queries into a plain text format, which can then be analyzed using document similarity measures to detect anomalies. While this approach can catch many common attack patterns, it is not foolproof. Attackers often find ways to bypass these filters by exploiting overlooked query structures or encoding techniques.

In addition to filtering, various encryption methods such as AES, DES, RSA, and ECC have been implemented to safeguard sensitive data. However, these encryption strategies are not always integrated with robust authentication mechanisms. As a result, attackers may still gain unauthorized access to confidential information if proper user identity verification is not enforced. Without strong security parameters and comprehensive identity checks, these vulnerabilities can be exploited, allowing malicious actors to inject harmful SQL queries and compromise the integrity of the database.

In summary, the current strategies, while helpful, have notable limitations in detecting and preventing sophisticated SQL injection attacks. There is a pressing need for more adaptive, intelligent, and integrated security solutions that can effectively address the dynamic nature of modern web threats.

3. Methodology

Over decades of evolution, databases have become highly robust and sophisticated structures. However, like all developing technologies, their increased complexity has also made them a prime target for new and creative attacks. This research introduces an innovative, multi-layered security mechanism designed to protect databases in distributed cloud environments, with a particular focus on countering SQL injection and Data Inference (DI) attacks. Our framework guarantees the secure execution of all requested queries, preventing unauthorized database access. The system operates through several integrated components:

- **Initial Defense and Deception:** The web server security model employs a technique to generate dummy SQL queries, which act as a decoy to confuse and mislead potential attackers.
- **Intelligent Filtering:** After initial processing, a proxy filtering method is used to remove extraneous server information from web data, securing the web server database. This filter is designed to recognize and neutralize queries containing suspicious forms, such as unusual special characters or malformed IDs.
- **Advanced Attack Detection:** A query tree mechanism is used for the final detection phase. This allows an administrator to analyze the structure of a query and trace it back to the user, identifying malicious intent based on the query's composition.
- **State-of-the-Art Encryption:** To provide the highest level of security, we utilize a Fully Homomorphic Encryption (FHE) algorithm. This allows identified queries to be encrypted and securely processed. The system can then compare a suspicious encrypted query against a stored library of malicious patterns using a similarity measure, effectively filtering out threats without ever exposing the raw data.

3.1. The Database Inference Problem

Database inference attacks are particularly challenging to classify and defend against because they leverage human intuition and logical deduction rather than exploiting a simple technical flaw. An inference attack occurs when a malicious user pieces together multiple pieces of non-sensitive information to logically deduce a protected or private fact without ever directly accessing it. The defense against such attacks is rooted in the three core principles of database security:

- **Confidentiality:** Ensuring that data is not disclosed to unauthorized individuals. Inference attacks directly violate this principle.
- **Integrity:** Protecting data from unauthorized modification.
- **Availability:** Guaranteeing that data and services are accessible to authorized users when needed.

3.2. Query Preprocessing

As a first line of defense, our system employs a robust query preprocessing step at the front-end. This process is designed to prevent a wide range of attacks by enforcing strict rules on incoming queries.

- **Restricting Query Types:** Inquiries are restricted to aggregate functions only. Each item selected for a query is automatically checked to ensure that only a pre-approved function is being used. This prevents attackers from executing arbitrary or harmful commands.
- **Enhancing Analysis:** A "count" is automatically added to each chosen item, which assists in later analysis and makes it easier to spot anomalies or patterns of abuse.
- **Improving Performance:** This preprocessing system automatically neglects and blocks non-essential or malformed queries from reaching the security model. By reducing the number of complex operations, the overall performance of query processing is greatly improved.

This proactive approach is crucial in environments where data is accessed by many different users, as it significantly reduces the threat of data breaches by blocking unauthorized query types at the source.

3.3. Proxy Server and Intelligent Filtering

At the heart of our detection strategy is a proxy server that acts as an intelligent filter. This server intercepts all web requests before they reach the database, allowing for a deep and careful investigation of all incoming traffic. This proxy-based filtering model is designed to identify and neutralize optimal queries used in attacks by:

- **Removing Malicious Characters:** The filter is configured to strip out uncommon or dangerous characters that are hallmarks of injection attacks.
- **Decoding Obfuscated Inputs:** Attackers often attempt to hide their malicious code by converting normal strings and characters into hexadecimal, ASCII, or Unicode formats. Our proxy is capable of decoding this obfuscated traffic to analyze its true intent.
- **Enhancing Anonymity:** The proxy server can also provide Network Address Translation (NAT), which makes the individual computers and users on the internal network anonymous when they are using the internet, adding another layer of security.

By capturing and analyzing web requests at a proxy, our system can effectively detect and prevent sophisticated SQL and DI attacks that might otherwise bypass traditional security measures.

3.4. Fully Homomorphic Encryption (FHE)

To achieve the highest standard of data protection, our framework utilizes Fully Homomorphic Encryption (FHE), a revolutionary form of encryption that enables specific calculations to be performed directly on encrypted data. When a query is received, it is encrypted and processed in its encrypted state. The result, which is also encrypted, is then returned. Only the user with the correct secret key can decrypt the final result, which will be identical to the result of performing the same operation on the original, unencrypted data. This process ensures that the underlying sensitive data is never exposed on the server during processing. To manage the computational demands of FHE, the operations can be distributed across multiple nodes to reduce processing time. The FHE security process includes four key stages:

- **Key Initialization:** A secret encryption key is generated. This key consists of a set of relatively prime moduli and other mathematical vectors, which are associated with the optimal queries and the database.
- **Encryption:** Using a public key, the sensitive data and the incoming query are encrypted. Each byte of the query is encoded into ciphertext before being processed.
- **Decryption:** This is the final step where the secret key is used to decrypt the resulting ciphertext, returning the original, readable information to the authorized user within the web database. This guarantees that the evaluation of a permitted circuit yields the correct and secure result.

4. Implementation and Performance Evaluation

To validate the effectiveness of our proposed security model against SQL injection and Data Inference (DI) attacks, we developed a proof-of-concept implementation. The system was built using the Java programming language within the NetBeans Integrated Development Environment (IDE). To simulate a realistic deployment environment, we utilized the CloudSim toolkit, which served as the web server and allowed us to model the large-scale cloud infrastructure where these applications typically operate. The performance of our innovative approach was rigorously evaluated and benchmarked against other established security algorithms. We assessed the model's detection and prevention accuracy across a diverse set of five common web application types: job portals, event management systems, online bookstores, payment gateways, and gaming platforms. The results of this comparative analysis are detailed in Figure 1, illustrating the model's efficacy in various real-world scenarios. For each application, we examined key performance indicators, including the accuracy of identifying malicious SQL and DI pages, as well as the overall success rates for attack detection and prevention.

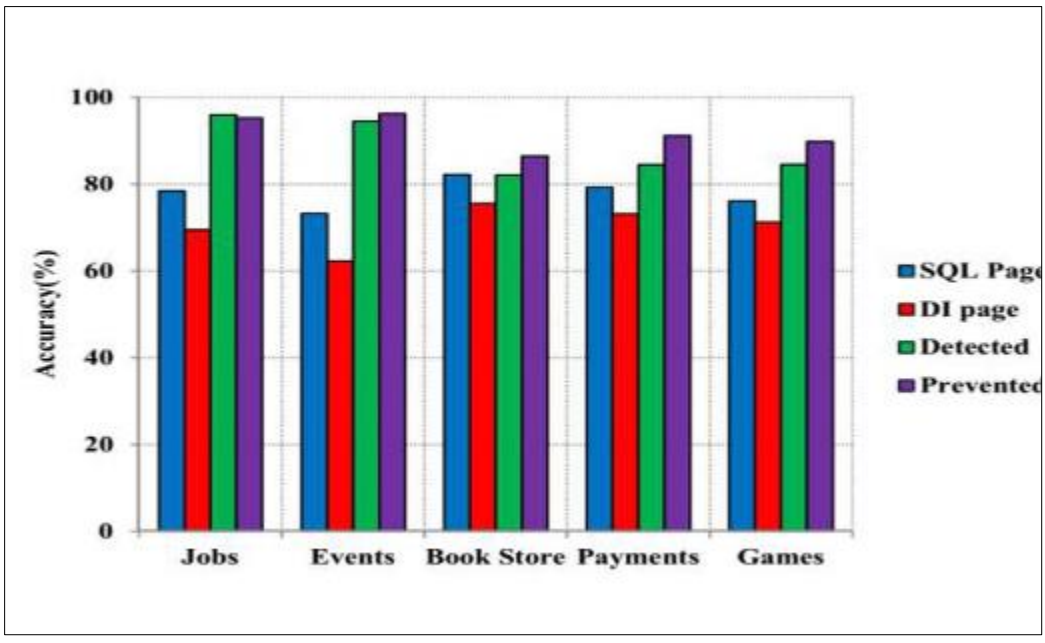

Figure 1 Detection Accuracy Vs Web application

Our findings indicate a high degree of effectiveness. For instance, in the context of a job application portal, the model achieved a 79% accuracy in identifying malicious SQL pages and 70% for DI pages. More critically, it successfully detected 97.23% of attack queries and prevented 96.89% of them from executing. When this analysis was extended to

the other application types, a notable trend emerged: the model demonstrated its highest prevention rate when applied to the events web application, highlighting its particular robustness in that context.

5. Conclusion

The proposed method operates on the database server side within a distributed cloud environment, establishing a robust security control framework. At the core of this approach is the integration of Fully Homomorphic Encryption (FHE), which is leveraged to rigorously evaluate and enhance the security posture of the system. SQL and Data Inference (DI) injection attacks remain among the most common techniques employed by malicious actors to compromise databases. These attacks work by altering SQL queries and manipulating the intended behavior of the database system, often to gain unauthorized access or to extract sensitive information. Our technique systematically analyzes all SQL and DI queries generated in response to user inputs. By capturing and examining the original structure of each query statement, the system is able to identify and intercept potentially harmful modifications before they can impact the database. This proactive analysis ensures that only legitimate queries are executed, significantly reducing the risk of successful attacks. Implementation results demonstrate the effectiveness of this approach. The system achieved a security prevention rate of 93.56% for web applications utilizing database backends. This high level of protection highlights the practical value of combining advanced encryption with intelligent query analysis, offering organizations a powerful tool to safeguard their data assets in dynamic cloud environments.

References

[1] Halfond, W. G. J., Viegas, J., and Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. Proceedings of the IEEE International Symposium on Secure Software Engineering, 13–15.

[2] Su, Z., and Wassermann, G. (2006). The essence of command injection attacks in web applications. ACM SIGPLAN Notices, 41(1), 372–382. https://doi.org/10.1145/1113361.1113409

[3] Bertino, E., Sandhu, R. (2005). Database security—concepts, approaches, and challenges. IEEE Transactions on Dependable and Secure Computing, 2(1), 2–19. https://doi.org/10.1109/TDSC.2005.9

[4] Buehrer, G. T., Weide, B. W., and Sivilotti, P. A. G. (2005). Using parse tree validation to prevent SQL injection attacks. Proceedings of the 5th International Workshop on Software Engineering and Middleware, 106–113.

[5] Shin, Y., Meneely, A., Williams, L., and Osborne, J. A. (2011). Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Transactions on Software Engineering, 37(6), 772–787. https://doi.org/10.1109/TSE.2010.81

[6] McClure, R. A., and Krüger, I. H. (2005). SQL DOM: Compile time checking of dynamic SQL statements. Proceedings of the 27th International Conference on Software Engineering, 88–96.

[7] Wassermann, G., and Su, Z. (2007). Sound and precise analysis of web applications for injection vulnerabilities. Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation, 32–41.

[8] Xie, Y., and Aiken, A. (2006). Static detection of security vulnerabilities in scripting languages. USENIX Security Symposium, 179–192.

[9] Shar, L. K., Tan, H. B. K., and Le, A. H. (2013). Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis. Information and Software Technology, 55(7), 1188–1202. https://doi.org/10.1016/j.infsof.2012.12.010

[10] Srivastava, S., and Kumar, S. (2018). A comprehensive study on SQL injection: Vulnerabilities, attacks, and prevention techniques. Information Security Journal: A Global Perspective, 27(4), 162–175. https://doi.org/10.1080/19393555.2018.1492897

[11] Doupe, A., Cova, M., and Vigna, G. (2010). Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. Detection of Intrusions and Malware, and Vulnerability Assessment, 111–131. https://doi.org/10.1007/978-3-642-14246-7_6

[12] Ray, I., and Ray, I. (2006). An approach for secure SQL query processing in a cloud environment. Proceedings of the 2006 International Conference on Information Technology: Coding and Computing, 2, 141–146.

[13] Li, Y., Chen, X., and Li, X. (2019). A survey on the security of cloud computing systems. Future Generation Computer Systems, 86, 914–929. https://doi.org/10.1016/j.future.2016.11.009

[14] Homoliak, I., Toffalini, F., Guarnizo, J. D., Elovici, Y., and Ochoa, M. (2019). Insight into insiders and IT: A survey of insider threat taxonomies, analysis, modeling, and countermeasures. ACM Computing Surveys, 52(2), 1–40. https://doi.org/10.1145/3303771

[15] Ghosh, A. K., and Swaminatha, T. M. (2001). Software security and privacy risks in mobile e-commerce. Communications of the ACM, 44(2), 51–57. https://doi.org/10.1145/359205.359226