SQL INJECTION PREVENTION

Pius Kawala
Student at Galgotias University, Uttar Pradesh, India

Siddhant Thakur
Student at Galgotias University, Uttar Pradesh, India

Abstract

SQL injection remains an existing security threat. It exploits vulnerabilities in web applications to execute malicious SQL commands. These attacks can disrupt operations, compromise sensitive data, and damage an organization's reputation. Despite significant improvements in web security, SQL injection remains a prevalent attack on most applications due to improper coding practices and input validation. SQL injections allow hackers to inject malicious code into web applications' input fields, manipulating databases. Successful SQL attacks enable users to execute admin tasks and recover files from the system, Sometimes, attackers can issue commands on the underlying database operating system. The primary problem this paper seeks to solve is persistent vulnerabilities in web applications to SQL injection attacks, which lead to data breaches, unauthorized access, and significant losses. Current solutions exist, but fail to protect due to their reliance on manual coding or inadequate automated tools. Therefore, sufficient knowledge is needed to have a unified framework to detect, prevent, and mitigate SQL injection attacks across various web platforms. The outcome of this project is crucial for enhancing the security of web applications, hence protecting sensitive data and maintaining the integrity of online services. By addressing the SQL injection problem, this paper aims to reduce the risk of data breaches, minimize potential damage to organizations, and improve cybersecurity practices.

Keywords: SQL injection, vulnerability, data breaches, cybersecurity.

Introduction

With the increase of web-based services, cyberattacks have grown significantly. SQL injection (SQLi) is a web application's most common and dangerous vulnerability. It occurs when malicious actors manipulate SQL queries through user inputs to gain unauthorized access to a database, steal sensitive data, or even alter and delete information. SQL injection attacks exploit poorly coded applications that fail to properly validate and sanitize user input, allowing attackers to bypass authentication mechanisms, access confidential data, and execute arbitrary SQL commands. This vulnerability remains a significant security threat, as highlighted in the OWASP Top 10 Vulnerabilities, which lists SQLi as one of the modern applications most critical security risks. The increasing reliance on web applications in fields such as e-commerce, finance, and healthcare makes these systems attractive targets for attackers. High-profile breaches, such as the attacks on Equifax and TalkTalk, exposed millions of user records,

underscoring the potential damage caused by SQL injection. As databases often store confidential information like customer credentials, financial records, and personal data, securing them is paramount.

This paper focuses on preventing SQL injection by adopting secure coding practices, parameterized queries, input validation, and the principle of least privilege. The objective is to understand the nature and mechanics of SQL injection and explore practical methods to mitigate it. Tools such as web application firewalls (WAFs), vulnerability scanners, and prepared statements will be demonstrated to show their effectiveness in protecting applications. By the end of this paper, various methods are used to develop a secure and robust strategy for query handling, providing practical insights into how developers can design secure applications resilient to SQL injection attacks. With the ever-growing number of cyberattacks, this project emphasizes the importance of a proactive security approach in software development.

Background Study

The most fundamental way of preventing SQL injection is by preventing SQL injection input parameters. Binning et al. (2015) highlighted the importance of rejecting user inputs with SQL-specific meta-characters like single quotes, comments, and semicolons. Even though attackers may find other ways to bypass these checks, their study showed that proper sanitization reduces the risk of SQL injection.

Web Application Firewalls (WAFs) filter traffic and monitor malicious HTTPS activities from the Internet to the user, as they act as intermediaries between users and web applications. Chen et al. (2019) showed how the WAFs can prevent SQL injection through signature-based and heuristic methods. WAFs are effective against known attack patterns, hence relying on pre-defined signatures makes them vulnerable to zero-day attacks or novel injection techniques. The authors proposed incorporating adaptive learning mechanisms to improve WAF efficiency. Static code analysis tools help analyze code during development to identify potential vulnerabilities. Gupta et al. (2020) proposed an automated static analysis framework to detect and identify SQL injection risks early in the development lifecycle. Their findings showed that the tools helped reduce vulnerabilities during the coding phase but often resulted in high false-positive rates.

Currently, the advancements in Artificial Intelligence have introduced machine learning models as a defense against SQL injection. Lee et al. (2022) developed a machine-learning system trained on datasets containing malicious SQL queries. Their model successfully detected injection attempts by identifying patterns and anomalies in query behaviour.

Brown & Tailor (2016) suggested developers should be trained on vulnerabilities associated with SQL injection. They indicated that the developer training programs are indispensable for minimizing the risks of SQL injection in the organization.

Automated Vulnerability Detection tools can scan and flag insecure code during development. Perez et al. (2017) suggested that the tools streamline secure coding practices but face adoption barriers due to integration challenges and perceived complexity.

Implementing the least privilege principle helps restrict user permissions, minimizing damage caused by SQL injection attacks and the potential for SQLi exploits. This strategy alone cannot single-handedly prevent attacks as suggested by the Secure Development Practices Guide (2021).

Overview of SQL injection.

In this section, we present a general overview of SQLI attacks. We discuss the sources of SQL attacks and classify their goals and types. SQL injection attack happens when an insertion of untrusted input in a valid query, which then maliciously alters the execution of the program. Here is a simple diagram explaining the SQL Injection attack.

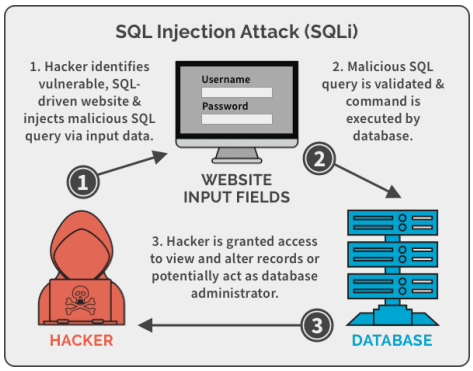

Figure 1

SQL attack sources.

- **Injection through user input:** Web applications normally provide forms for users to enter data. Such data can be registration details, login details, and so on. These forms in turn, can be exploited by hackers to inject malicious code and gain sensitive data or manipulate databases. Common loopholes are Login names, Passwords, credit card numbers, and so on.
- **Injection through cookies:** Cookies are used to store users' web preferences in web applications. These are files stored on the client machine that contain state information gathered in web applications. So, an attacker can embed malicious code in cookie files on a computer, making web applications using the content vulnerable to attacks.
- **Injection through server variables:** These are parameters that contain HTTP metadata, network headers, and environmental variables. Web applications use these variables to identify browsing trends. If these variables are stored in the database without validation, attackers may exploit this vulnerability.
- **Stored injection:** Attackers put malicious code into the database to indirectly launch an SQL injection attack each time the input is used. In this attack, the attacker injects

malicious code into the input field that stores data in the database. Therefore, whenever the application retrieves data, the code is executed.

SQL Injection Attack Goals

Hackers have different motives when performing attacks. Here are some goals that hackers have while performing SQLIA.

- **Identifying injectable parameters:** The attacker identifies which parameters could be used to inject malicious code. The parameters are inserted through the user input data, and then manipulation is done. The user input data can be a username in a form, card numbers stored in cookies, etc. Here is a simple illustration.

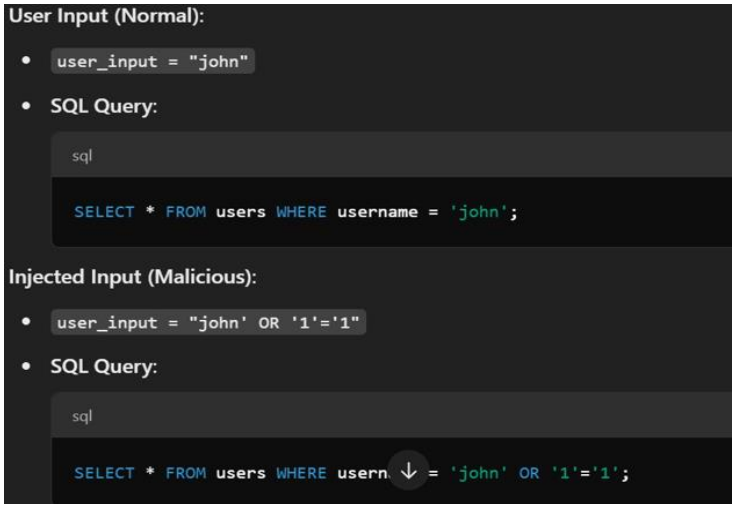

Figure 2

- **Performing database fingerprinting:** In this attack, hackers tend to use trial-and-error techniques to determine the version and type of database used. For instance, while performing this attack, usually an error message is obtained which shows the type and version of the database that is used, hence a loophole is found.

- **Determining database schema:** To successfully extract data from a database, the attacker needs to know the database schema information, such as table names, column numbers and names, and column data types. Hackers use the database schema to create an accurate consequent attack to extract or modify data from the database.

- **Extracting data:** These attacks aim to extract data values from the database. The attack poses a great risk to web applications as hackers may gain top-secret data. Such data can be a customer's bank credentials and private data.

- **Database alteration:** These attacks aim to alter information on the database. For example, a hacker may change price details on e-commerce sites to pay less than the

required amount.

- **Performing denial of service:** These attacks aim to disable services provided by web applications. This is done by shutting down the database of web applications, locking tables, dropping tables, etc.

- **Bypassing authentication:** This attack aims to bypass the authentication mechanisms of web applications. Therefore, a hacker can gain high privileges if he succeeds in bypassing authentication using an account of a user with many rights. Example admin rights.

How to detect SQL injection vulnerability?

The Burp Suite Web vulnerability scanner is mostly used to find SQL injection vulnerabilities. It can be detected manually by using systematic tests against every entry point in the application. This involves:

- Using single quote characters and looking for errors.
- Submitting Boolean conditions such as OR 1=1 and 1=2, then look for differences in application response.
- Submitting payloads designed to trigger time delays when executed within an SQL query and looking for differences in the time taken to respond.
- Submitting some SQL-specific syntax that evaluates to the base (original) value entry point, and to a different value, and looking for systematic differences in the resulting application response.
- By using automated tools for detecting and exploiting SQL injection. Example:
- sqlmap -u
- "http://example.com/index.php?i d=1" –batch
- By running the application in a controlled environment and performing dynamic testing to identify vulnerabilities. Tools like IBM Appscan, Veracode, and Netsparker are used.
- Through Penetration testing, we can simulate real-world attacks.
- Performing input fuzzing by sending unexpected inputs to identify vulnerabilities.
- Regularly scanning the databases and web applications to look for customizable payloads to test against different database systems.
- Analyzing the application's source code to identify vulnerable areas.

How can we prevent SQL injection?

As technology continues to grow, sql injection attacks cannot be prevented completely. Therefore, several measures can be taken into account to minimize and secure databases from SQL injection.

1. Using prepared statements.

Prepared statements separate SQL code from user input, thus preventing attackers from injecting malicious SQL commands. This happens when SQL commands use parameters

instead of inserting values directly into the command, hence preventing malicious queries from entering the backend and harming the database.

2. Input validation.

Here, we use strict rules for acceptable formats and values on the input. So we restrict the use of metacharacters such as “, ’, -- , ; , /*. In this example, the site doesn’t accept usernames with the symbols above, hence making it secure from SQL injection attacks.

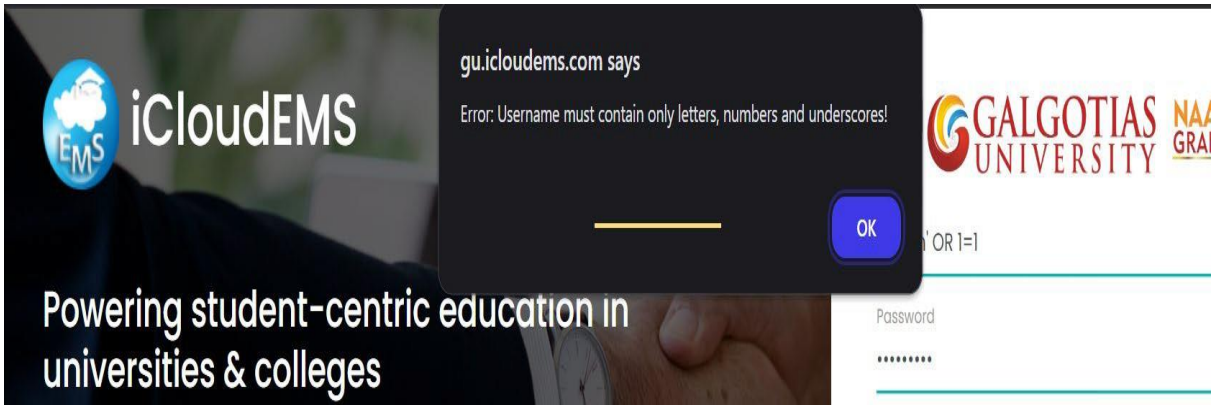

Figure 3

3. Limiting privileges.

Using the root account when connecting to the database is not advisable unless required, because once the attacker has root access, it means the entire database is vulnerable. Therefore, using an account with limited privileges is useful so that the scope of damage will be reduced.

4. Hiding info from the error message.

The database architecture is easily known using error messages. Therefore, attackers make use of this information to know the architecture of the database and exploit it. So, if something goes wrong, it’s important to show generic information that encourages users to call for technical support team if the problem persists.

5. Updating the database.

Applying patches and keeping the system up-to-date version is essential for maintaining the security of the database. SQL injection vulnerability is a frequent programming error, and the system should be regularly checked.

6. Keeping database credentials separate and encrypted.

Database credentials should be stored and encrypted so that attackers won’t benefit from the information once it falls into their hands. Also, don’t store sensitive data, and if you don’t need it, then delete information when no longer in use.

Conclusion

As technology continues to expand, it comes with new threats. With the use of AI and Machine Learning, some methods can be used to prevent SQL attacks. Preventing SQL injection attacks requires an in-depth strategy combining secure coding practices, proper database configuration, and active monitoring. If these techniques are adopted, then organizations can minimize vulnerabilities and protect their applications from malicious attacks.

References

Allen, R. (2019). Modern WAFs and the Challenge of Evolving Threats. Proceedings of the International Cybersecurity Symposium.

Binning, J., Kossmann, D., & Loesing, S. (2015). Advanced SQL Prevention Techniques. Journal of Secure Computing.

Brown, M., & Taylor, S. (2016). Enhancing Secure Coding Practices through Training. Journal of Information Security Education.

Martinez, L., & Lee, K. (2020). Machine Learning Applications in SQL Injection Defense. Journal of Artificial Intelligence and Security.

Perez, J., Wu, C., & Singh, T. (2017). Evaluating Automated Security Tools for Developers. ACM Transactions on Software Engineering.

Secure Development Practices Guide (2021).

Smith, A., & Johnson, B. (2018). The Role of Input Validation in Application Security. Cybersecurity Advances.