EasyChair Preprint

№ 12821

Investigation into Methods for Detecting SQL Injection Technology

Bhargav Reddy Dumpa

March 28, 2024

Investigation into Methods for Detecting SQL Injection Technology

Dumpa Bhargav Reddy

Student at Parul university, Limda, Vadodara, Gujarat,- 391760

Abstract: SQL injection stands out as one of the most prominent and perilous security vulnerabilities in web applications. This research scrutinizes the characteristics and methodologies underlying SQL injection attacks while also presenting a detection mechanism. Furthermore, a comprehensive defense and remediation model against SQL injection is proposed, focusing on non-intrusive approaches. Enhancements in server resilience against SQL injection are achieved through the implementation of security measures targeting operating systems, IIS, databases, and related components. The efficacy of these strategies is demonstrated through practical implementation in real-world projects.

Keywords: SQL injection vulnerability; Web application security; Non-intrusive detection; Defensive measures

INTRODUCTION

With the rapid evolution of internet technology and the emergence of new online platforms such as Web 2.0, social networking sites (SNS), and microblogging platforms like Weibo, web-based applications have become pervasive. Many businesses rely on web platforms for their digital transformation efforts. However, this increased reliance on web technologies has also attracted the attention of malicious actors, leading to various security challenges. Among these challenges, SQL injection stands out as the most prominent and hazardous. According to the OWASP (Open Web Application Security Project), SQL injection consistently ranks as the top security risk in web application programs, posing significant threats to data integrity and confidentiality. Typically, attackers exploit SQL injection vulnerabilities to manipulate webpage contents, extract sensitive data from backend databases, or even implant malicious code to compromise user interactions. For instance, in 2011, numerous websites fell victim to the Liza Moon SQL injection attack, and major websites belonging to SONY Corporation experienced frequent SQL injection attacks during the same period. These incidents underscore the pervasive nature of SQL injection threats, with attacks continuously evolving and impacting a vast number of web properties.

According to a report published by Imperva Corporation in October 2012, SQL injection-related topics accounted for 19% of discussions in hacker forums, highlighting the persistent interest and engagement in exploiting these vulnerabilities. Given the widespread impact and severity of SQL injection attacks, it is imperative to develop robust detection and prevention mechanisms to safeguard web environments. Consequently, numerous researchers have undertaken extensive efforts in this area, focusing on various aspects such as:

- Establishing coding standards and program specifications to mitigate vulnerabilities.
- Developing defense models to thwart automated attack vectors.
- Designing detection tools and secure algorithms for proactive defense.
- Implementing data encryption techniques to protect sensitive information stored in databases.

- Exploring innovative protection mechanisms, including Base64-data-encoding, to counter SQL injection attacks effectively.
- Utilizing static and dynamic analysis techniques for intelligent detection and mitigation strategies.

These research endeavors collectively contribute to enhancing the resilience and security posture of web applications against SQL injection threats, thereby safeguarding critical data and ensuring the integrity of online interactions.

In this research, the resilience against SQL attacks was significantly bolstered by focusing on enhancing server security measures and optimizing database policy settings. Additionally, the defense and remediation model developed in this study proved invaluable in mitigating SQL injection attacks post-incident. For users facing constraints in promptly modifying source code, this model serves as a practical remedial measure or preventive measure against SQL injection attacks.

INTRODUCTION OF SQL INJECTION

SQL injection represents a significant threat to web applications, stemming from vulnerabilities within their codebase. This attack vector operates by leveraging the interface of a database to inject user data directly into the underlying database manipulation language. Consequently, attackers can exploit this vulnerability to gain unauthorized access to the database and potentially compromise the entire operating system. The root cause of SQL injection attacks often lies in the oversight of developers who prioritize business logic implementation over code robustness and security. Insufficient consideration of input data validation during code editing leaves applications susceptible to security breaches.

SQL injection attacks pose a pervasive and substantial risk to networks, as they can target any database server supporting SQL command batch processing, including popular platforms such as MySQL, MS SQL, Oracle, DB2, and Sybase. Furthermore, the simplicity of the attack process and the minimal technical expertise required make SQL injection accessible to a wide range of attackers. Even without in-depth knowledge of SQL injection techniques, attackers can exploit vulnerabilities using readily available tools such as HDSI, SQLmap, Bobcat, BSQL, NBSI, and Domain.

One of the key challenges in mitigating SQL injection threats lies in their deceptive nature. Since SQL injection attacks typically originate from HTTP service ports (commonly port 80), they can masquerade as legitimate web access requests, evading detection by standard network firewalls that typically permit HTTP/HTTPS traffic. Consequently, SQL injection attacks often remain undetected until they have already caused significant damage, underscoring the importance of proactive security measures and thorough code auditing to prevent exploitation.

ANALYSIS OF PROCEDURE OF SQL INJECTION

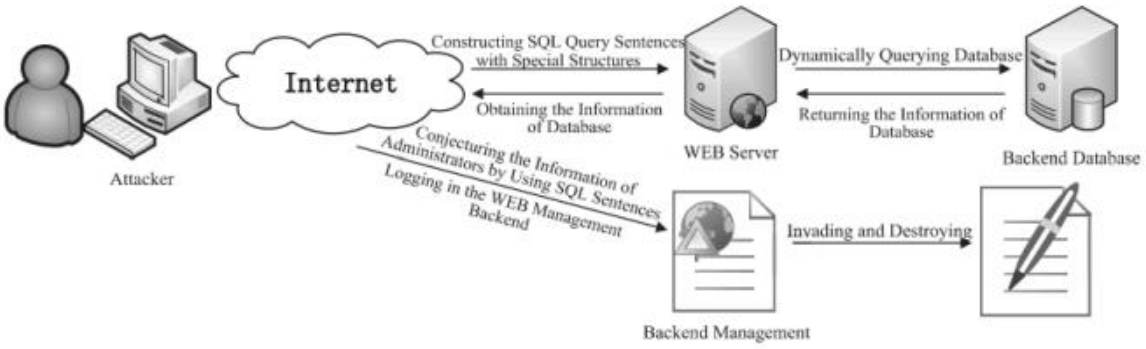

FIGURE1. Procedure of SQL injection

SQL INJECTION VULNERABILITY DECTION

Manual finding of SQL injection vulnerabilities

Based on the complexity of leak detection, SQL injection vulnerabilities within web application systems are categorized into three levels, as outlined in Table 1.

(1) First Level SQL Injection Vulnerabilities:

The detection of these vulnerabilities involves constructing SQL clauses dynamically using single quotes, double quotation marks, and the condition "1=1". For instance, consider a web login page with the following SQL validation code:

```
SELECT * FROM admin WHERE username='" + userInput.Text + "' AND userPass='" + passInput.Text + "'
```

In this example, the vulnerability arises from the direct concatenation of user inputs without proper sanitization, potentially allowing attackers to manipulate the SQL query through malicious input.

When inputting "user" or "1-1--" into the textbox, the authentication process succeeds. This occurs because "--" denotes a SQL comment, causing the subsequent code to be ignored. Consequently, "1-1" effectively becomes the identity, resulting in a condition that always evaluates to true within the SQL clause. As a result, the authentication process bypasses without proper validation.

TABLE 1. Three levels of SQL injection vulnerabilities

level	description	example
first level	error triggering	'
	always true condition	1' or '1'='1
	Always false condition	1' and '1'='2
	no condition	value' or '1'='2
	SQL comment	- -
	Microsoft SQL Server concatenation	1' or 'ab'='a'+'b
second level	variant keyword	uPdate as update
	ASCII conversion	a=chr(97)
	restructuring	aandnandd as and
	UNICODE coding	and%201%3Dl as and 1=1
third level	Use stored procedures to complete access to the database	';exec master.xp_cmdshell 'ping 127.0.0.1'- -

(2) Second Level SQL Injection Vulnerabilities:

In addition to constructing SQL clauses dynamically, detecting SQL injection vulnerabilities can involve more advanced techniques such as variant keywords, ASCII conversion, restructuring, and UNICODE coding. For instance, utilizing ASCII representations of input characters can bypass web application filters that are based on keywords or special characters. This is possible due to the one-to-one correspondence between characters and ASCII codes. For example, representing "or 1=1" using ASCII conversion would be "chr(111)chr(114) 1=1". Similarly, encoding special characters using UNICODE in URLs allows for automatic decoding by web servers. For instance, the UNICODE for a space is "%20", and for an equal sign, it is "%3D". UNICODE coding provides a more concealed approach to bypassing filters.

(3) Third Level SQL Injection Vulnerabilities:

At this level, attackers typically exploit specific SQL procedures stored within the database to gain complete access. For example, the payload ""; exec master.xp_cmdshell 'ping 127.0.0.1'--" can execute the ping command, demonstrating the ability to execute arbitrary commands through SQL injection.

Automatically finding SQL injection

In cases where web applications comprise extensive codebases, manual detection of SQL injection vulnerabilities can be arduous and inefficient. Automated tools offer a systematic and thorough approach to this task. While these tools may lack an understanding of the underlying web application logic, they excel at rapidly testing numerous potential injection points—a task that is challenging for humans to perform consistently and comprehensively. Various commercial and open-source tools are available for automated vulnerability detection, including HP WebInspect, IBM Rational AppScan, HP Scrawl, SQLIX Paros Proxy, and others. While automated discovery tools may not identify every existing vulnerability, they nonetheless provide valuable security assessments of web applications, including comprehensive testing for SQL injection vulnerabilities.

DETECTION AND DEFENSE OF SQL INJECTION ATTACK

Detection of SQL injection attack

SQL injection attacks can be executed through either manual techniques involving the deliberate insertion of abnormal inputs or through the utilization of automated tools [3]. Regardless of the method employed, successful SQL injection attacks typically leave traces within the system. Four effective methods exist for determining whether a web system has been subjected to SQL injection attacks [4].

(1) Examination of Backend Database Tables:

A key indicator of a potential SQL injection attack involves checking for the presence of abnormal data tables within the backend database. SQL injection attacks, particularly those conducted using tools like HDSI and NBSI, may result in the creation of temporary tables within the database. Identifying these anomalous tables can provide insight into potential security breaches.

(2) Examination of Backend Database Logs:

Analyzing logs generated by the backend database is crucial. Initiation of log management by the backend database ensures that database accesses are recorded. Specifically, SQL injection attacks may be identified within these logs if erroneously executed SQL statements are logged.

(3) Inspection of IIS Logs:

Reviewing logs generated by the Web server, particularly IIS logs, provides detailed information such as visitor IP addresses, access times, accessed files, and access methods. In the case of SQL injection attacks, pages containing injection points are typically accessed frequently. Given the large size of log files, anomalies in size or content can indicate potential SQL injection attacks.

(4) Assessment of System Information:

Furthermore, intrusion detection can involve examining various system parameters, including system administrator account activity, status of open ports, recently generated files, presence of viruses, and firewall logs. Anomalies in any of these areas may suggest a security breach.

Prevention of non-intrusive SQL Injection Attack

A prudent web system administrator should cultivate the practice of regularly backing up data and routinely examining the log information of the database, IIS, and firewall. Any unusual activities detected should prompt swift investigation by the administrator to identify the injection point and ascertain the cause. If the issue stems from a security vulnerability in the operating system, timely installation of updates and patches is imperative. Conversely, if the problem lies within the web system program itself, the administrator should promptly implement defensive measures and proceed with code modifications. When enhancing and modifying program codes, programmers should implement rigorous filtering, detection, and parameterized SQL queries on dynamically constructed SQL statements within the codebase. This approach necessitates source code modification and therefore qualifies as an intrusive solution [5], which is not within the scope of this study. Alternatively, this study proposes a non-intrusive approach for preventing SQL injection attacks, using Windows+SQL SERVER+ASP.NET as an example. This approach focuses on

enhancing security configurations on the server and implementing defensive remedies to thwart potential attacks.

PREVENT SQL INJECTION USING STORED PROCEDURES

Stored procedures provide a robust defense against SQL injection attacks by enforcing parameterization and encapsulating database logic. Here's how stored procedures help mitigate SQL injection vulnerabilities:

1. **Parameterization**: Stored procedures allow developers to pass parameters securely, separating user input from SQL code. Parameters are treated as data rather than executable code, significantly reducing the risk of injection attacks.

2. **Prevention of Dynamic SQL**: Stored procedures execute predefined SQL statements, eliminating the need for dynamically generated SQL queries where injection vulnerabilities often arise.

3. **Input Validation**: Developers can implement input validation within stored procedures to ensure that only valid data is processed. This helps prevent malicious input from compromising the integrity of SQL queries.

4. **Access Control**: Stored procedures can be assigned specific permissions, restricting access to sensitive database operations. This limits the impact of potential SQL injection attacks by controlling the actions that can be performed.

5. **Encapsulation of Logic**: By encapsulating database logic within stored procedures, developers can limit the attack surface exposed to potential vulnerabilities. This reduces the risk of unauthorized access to critical database resources.

6. **Audit Trails**: Stored procedures facilitate the creation of audit trails, allowing administrators to track and monitor database activity. This helps detect and respond to SQL injection attacks in a timely manner.

Overall, stored procedures provide a comprehensive defense mechanism against SQL injection attacks, making them a crucial component of secure database development and management practices.

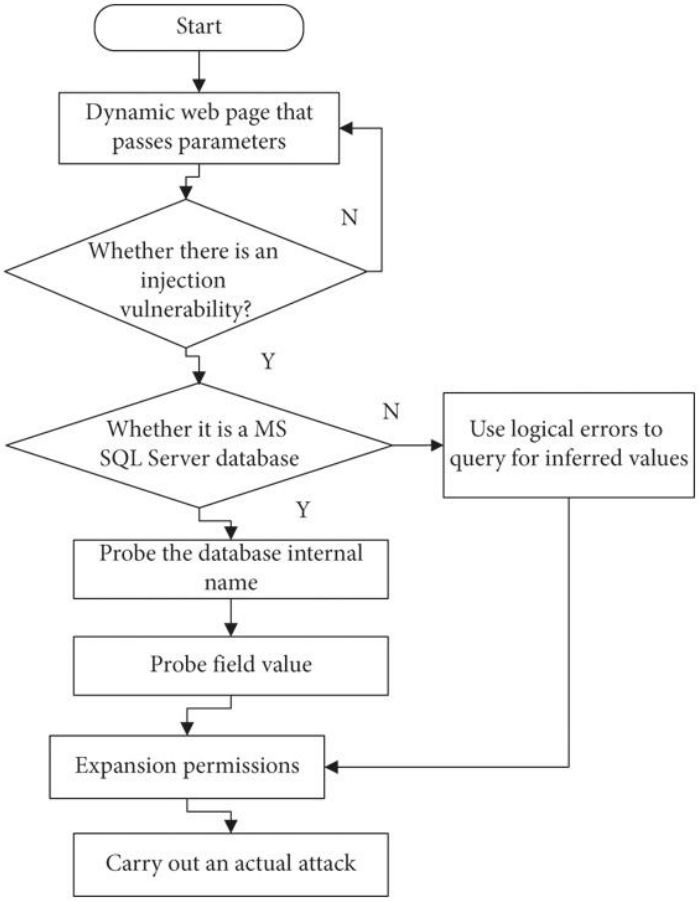

Figure 2. Steps in Stored Procedure

Example

Certainly! Here's an example of a simple stored procedure in SQL Server that demonstrates how parameterization can help mitigate SQL injection attacks:

Let's say we have a table named `Users` with columns `Username` and `Password`, and we want to create a stored procedure for user authentication.

```SQL
CREATE PROCEDURE AuthenticateUser
@Username VARCHAR(50),
@Password VARCHAR(50)
AS
BEGIN
SET NOCOUNT ON;
```

```
DECLARE @IsValidUser BIT;

-- Check if the username and password match
SELECT @IsValidUser = CASE WHEN EXISTS (
SELECT 1 FROM Users
WHERE Username = @Username AND Password = @Password
) THEN 1 ELSE 0 END;

-- Return 1 if the user is valid, otherwise return 0
SELECT @IsValidUser AS IsValidUser;
END;
```

In this stored procedure:

- We define two input parameters `@Username` and `@Password`.

- We use these parameters directly in the SQL query to compare against the `Users` table.

- The SQL query is parameterized, meaning that user input is treated as data rather than executable code.

- The procedure returns a value indicating whether the user authentication was successful (`1` for valid user, `0` for invalid user).

By using stored procedures with parameterized queries, we eliminate the risk of SQL injection attacks because the input values are treated as data, not as part of the SQL code.

References

1. M. Praveen Kumar Reddy. "Research on SQL Injection Attack and Defense Technology of Power Dispatching Data Network: Based on Data Mining." 1Department of Computer Science and Technology, Shenyang University of Chemical Technology, Shenyang, Liaoning 110142, China. 30 Jul 2022.
2. Limei Ma,Dongmei Zhao, Yijun Gao, Chen Zhao. "Research on SQL Injection Attack and Prevention Technology Based on Web". International Conference on Computer Network, Electronic and Automation (ICCNEA). 28 November 2019.
3. Shobana R., Dr M Suriakala. "A Thorough Study On SQL Injection Attack-Detection And Prevention Techniques And Research Issues". Part time Research Scholar, University of Madras, Assistant Professor, Department of Computer Science and Applications, D.K.M. College for Women, Vellore- 1. Volume 10 Issue 5 – 2020.
4. Wubetu Barud Demilie, Fitsum Gizachew Deriba . "Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques". Journal of Big Data volume 9, Article number: 124 (2022). 30 December 2022.
5. Xue Ping-Chen . "SQL injection attack and guard technical research". Chongqing College of Electronic Engineering Chongqing 401331,China. 6 December 2011.
6. Edmond Jajaga, Ferihane Nijazi Nuhiji . "Evaluation of triggers and stored procedures on relational databases". Conference: University for Business and Technology International Conference. October 2018.
7. Jagdish Halde . "SQL Injection analysis, Detection and Prevention". San Jose State University. 2008.
8. Mohd Amin Bin Mohd Yunus, Muhammad Zainulariff Brohan, Nazri Mohd Nawi, Ely Salwana. "Review of SQL Injection : Problems and Prevention". JOIV International Journal on Informatics Visualization 2(3-2):215. June 2018.
9. Bhuvana, Bindu, Chandan H, Brijesh Reddy KH, Mr. Pradeep V. "Review Paper on a Study on SQL Attacks and Defense". Department of Information Science and Engineering Alvas Institute of Engineering and Technology, Mijar, Moodbidri, Karnataka, India. Volume 2, Issue 1, August 2022