

Overview of SQL Injection Attacks and Prevention Methods

1st Alex McMoil
University of Pittsburgh
Pittsburgh, PA
alm470@pitt.edu

2nd Ralph Gonsalves
University of Pittsburgh
Pittsburgh, PA
rvgl2@pitt.edu

3rd Peter Smith
University of Pittsburgh
Pittsburgh, PA
pes71@pitt.edu

Abstract—Structured Query Language (SQL) is a programming language that is used to access and manipulate databases. SQL is used in almost every modern database as the universal option for relational databases. Recent popularity in cloud storage and computing due to machine learning has drastically increased the need for secure databases. In this overview, we focus on SQL, its vulnerability to injection attacks, and methods to prevent injection attacks. Our overview found 3 main types of SQL injection attacks: error-based, union based, and blind. We demonstrate each of these attacks with simulations.

Index Terms—Structured Query Language, SQL, SQL Injection

I. INTRODUCTION

SQL injections have been a part of some of the largest cybercrimes to date. Web applications that are vulnerable to SQL injections are at risk of having their entire databases exposed. These databases often store private information, leaving consumers vulnerable to identity theft and fraud. Notable cases of SQLi include 130 million credit cards being stolen [1], the Swedish general election being tampered with [2], and one million Sony accounts being leaked [3].

A. What is SQL?

Structured Query Language (SQL, pronounced *S-Q-L* or *sequel*) is a programming language designed to interact with databases. The language also allows the user to perform various operations on the data within the databases [4]. The language was initially developed in the 1970s, for use by database administrators and developers to write data integration scripts and run analytical queries. SQL is an extremely versatile language, that is used by everyone from data scientists to engineers [5]. A SQL query is a way of requesting a bit of data from the database [6].

SQL queries can also perform various operations. A SQL query can be used to create a new database or insert data into the database. It can not only retrieve data from the database, but it can also modify and update existing data within the database [6]. It can delete data from the database and delete tables from the database. Queries can be used to set permissions for the tables, views, and procedures [6].

One of the other benefits of SQL queries is that they maintain consistency in database transactions. They follow the basic rules of atomicity, consistency, isolation, and durability. It ensures that all instructions occur successfully, or none of

them will execute at all. All transactions that are running concurrently should execute independently of each other and not affect the other transactions [7].

II. BACKGROUND

A. SQL Injections

A SQL injection (SQLi) is an attack that consists of an injection of a SQL query via the input data from the client to the application [8]. Informally, SQLi attacks happen when an attacker changes an SQL query with by inserting SQL commands into the query [9]. SQLi's can have numerous effects on the database that they are attacking. One of the most common applications of a SQLi is to read sensitive data from a database that an attacker may not be authorized to use. A SQLi can modify databases by inserting, updating, or deleting data. They can perform administrative operations, such as a shutdown of the database [8].

A successful SQLi attack allows the attacker to manipulate the SQL queries that an application makes to the database. The first step is that it identifies vulnerable inputs. These inputs could be anything from fields in a form, URL parameters or any other input mechanisms [10]. Once a vulnerable input is found, the attacker will create a SQL statement intended to be inserted into the query that will be executed by the application. These statements are designed to modify the original SQL query and cause it to perform unintended actions by the application. To successfully inject the SQLi attack the attackers must bypass security measures. These security measures consist of things like input validation and escaping special. These can be achieved through methods such as string concatenation or commenting out part or the original query [10].

Once security is bypassed the application then executes the SQL query that includes the SQLi. These modified queries will then perform any number of attacks, such as the unauthorized viewing of data, the deletion of data, the extraction of sensitive information, altering or adding data. Some SQLi's can exploit vulnerabilities in the server itself, extending the attack beyond the original database, and manipulate data at the server level. They can execute commands on the operating system and access unauthorized portions of the server's file system [10]. By leveraging the dynamic execution of SQL in applications, it exploits the way SQL queries are constructed [10].

B. SQL Injection Types

There are 4 main SQL injection types: 1) error-based; 2) tautology 3) union-based; and blind, which is split into 4) blind Boolean-based and 5) time-based blind.

1) *Error-based SQL Injection*: The first type of injection is called an error-based SQL injection. This type of injection obtains information about the structure of the database from error messages issued by the database server [11]. This method can only be used against MS-SQL Servers. The data that the attacker is after is returned in the error message. This error enables the attacker to understand the entire database structure [10].

2) *Tautology SQL Injection*: The tautology SQL injection attack utilizes an always true Boolean statement to obtain information from the database.

For example, adding an always true statement to password will allow an attacker to bypass the password check in the database and access any account.

3) *Union-Based SQL Injection*: Union based SQL injections use a specific operator to implement the attack. The Union operator is used to aggregate the results of two or more select queries into a single result. The result is then returned as part of the HTTP response [11]. Union base SQL injection attacks are the most popular type of SQLi. There are two important needs that must be met for a union-based injection to work. The first need is that the query must return the same number of columns as before. The second thing is that the data types that are returned are not changed after the query executions [11].

4) *Blind Boolean-Based SQL Injection*: Blind Boolean-Based SQL injection submits a query to the database and forces the application to produce a different response depending on if the query returns a true or false response [11]. The attacker never actually sees error messages or different data, hence the blind portion of the attack, but the attacker can ascertain the result of the true/false query through how the website responds. Generally, if the statement is true, the site functions normally; if the statement is false, the site reacts differently [12].

5) *Time-based Blind SQL Injection*: This leads to time-based blind SQL injections. Much like a blind Boolean-based SQL injection the time-based injection relies on a true or false response. The attacker uses branching statements, like if-else statements, to determine if the query is true/false. The length of the response time depends on the result of the branch, which tells the attacker the result of the query [12].

C. Implementation of SQLi Attacks

There are four main mechanisms for SQL injection: 1) user input; 2) cookies; 3) server variables; and; 4) second order attacks.

1) *User Input*: Many web applications accept inputs through forms, which pass inputs to the database for processing. If the application doesn't sanitize the inputs, an attacker will be able to inject malicious SQL statement [10].

2) *Cookies*: Cookies are files that contain information from web applications that are stored on the user's machine. Web applications can use cookies to restore the user's preferences, data, etc. A malicious user can modify the information in the cookies to perform SQL injection [9].

For example, a cookie can save the account information of a user. The server would process this cookie as an SQL query to the account database to automatically log in the user. An attacker can modify this cookie to send arbitrary SQL queries to the account database, such as obtaining other people's account information.

3) *Server Variables*: Server variables are a group of variables like HTTP, network headers, and environment variables. Web applications process and store these variables in their databases. Attackers can forge HTTP and network header to run SQLi attacks directly in these headers. These variables are sent to the server which are then executed as SQL queries [9].

4) *Second Order Injection*: The previous three mechanisms are all first-order injections. They all execute SQL queries immediately. Second-order SQLi rely on future uses of malicious seed input during other operations [9]. These kinds of attacks lie dormant for a period of time before being executed at a later time.

For example, a user uses a seed user name "admin' --". The ' -- is the escape for the SQL comment operator. So, the application properly escapes the comment when storing it in the database, but when accessing it again, it will comment any SQL after the username. For this example, the database runs the commands below to update passwords.

```
UPDATE users SET password='newpwd'
WHERE userName='admin'--' AND
password='oldpwd'
```

When the attacker updates their password, the ' -- escapes into a comment, which comments the rest of the query. As such this allows the attacker to change the password of the admin account to whatever they desire.

D. What Can SQLi Injection Attacks Do?

Attacks perform a large range of things they can do.

1) *Extract Data*: The main goal of this attack is to extract private information from the backend of a database. This includes credit cards, Social Security numbers, and personal information.

2) *Gain More Control*: Attackers bypass authentication of web applications and gain access to their database. Additionally, they can escalate their privilege, allowing them to act as admins of the system.

One of the most dangerous attacks is when attacker attempt to gain complete control of a system, allowing them to remotely execute commands. The server is considered completely compromised. [13].

3) *Denial of Service*: SQLi attacks can execute commands in the database to shutdown the web application. Additionally, attackers can lock or drop (delete) the database to deny service [13].

4) *Set Up Further Attacks:* Attackers often probe many input fields of a website with SQL injections to find vulnerabilities. These are often done with automated tools, scanning input fields, cookies, server variables, etc.

Attackers also aim to gain information about the database itself. Different databases are vulnerable to different attacks, and this information is invaluable to attackers.

Some injection attacks are used to evade various prevention methods. They cover up other attacks with injected SQL. [13]

E. Prevention Methods

here are numerous methods to prevent SQLi attacks. In this section we will describe some of the most popular and effective methods of prevention. Many of these methods can be deployed alongside each other to allow for more effective prevention.

1) *Input filtering:* Input filtering involves not allowing user input to effect the database with potential SQL queries. Input filtering is the first line of defense against SQLi attacks. While input filtering can prevent the most trivial attacks, it is not able to fix any underlying vulnerabilities within the system itself [14]. In most cases input filtering can be evaded by attackers. Many attackers can exploit extended URLs and special character handling to gain unauthorized access to a database [15]. One way to filter inputs is to deny extended URLs. An attacker can use scripts to probe a database to search for vulnerabilities. The downside of denying extended URLs is that there are legitimate uses of extended URLs, making it hard to outright deny all extended URLs [15].

2) *Sanitize Data:* Another filtering method is to sanitize data and limit special characters. SQLi attacks can be introduced by abusing special characters, using a web interface to inject SQL code into the database. The data must be sanitized to prevent concatenation and prevent user inputs being used as commands [15]. One method of sanitization is to use typecasting. By typecasting the data, inputs will be restricted to a format that is expected in the specific field and that's it. Another method would be to convert all inputs into strings. This can ensure that any dangerous characters are not passed to a SQL query [15].

3) *Restrict Database Code:* If a database has the code available to it limited to the bare minimum necessary to operate it becomes harder for attackers to exploit vulnerabilities. These restrictions include reducing functionality, using stored procedures, whitelisting user inputs, and enforcing prepared statements [15].

By reducing available functionality, the operator is limiting the available entry points for an attacker to exploit. Using stored procedures isolates the users from the database, preventing some exploitations. Instead of executing the code directly in the database, the application will use stored procedures and return the results of those procedures [15]. Whitelisting user inputs allows the operator to validate user inputs. The operator creates a list of all valid SQL statements, leaving any unvalidated statements out of the query. Inputs should also be

configured for user data by the context [15]. By enforcing prepared statements, the database can easily distinguish from user inputs and code without risking an injection. By implementing prepared statements, the database will treat any malicious SQL statements as data and not as commands [15].

4) *Restrict Database Access:* The most common restriction is by using a firewall. Firewalls deploy a set of default rules and configurations to protect against vulnerabilities. These firewalls help filter out malicious data and attacks [15]. You can also establish appropriate privileges and strict access to the database. If a website only needs to select information, you can restrict its ability to access more information. Setting administration privileges further limits who can access and make changes to a database. Limiting shared databases and user accounts limits the amount of cross traffic accesses which will increase the vulnerabilities in the system [15].

5) *Encryption:* Encryption is an obvious method to protect against attacks. With the appropriate levels of encryption even if an attacker were to be able to see the information, they wouldn't be able to decrypt it. Because the intruder does not have access to the cryptographic keys, there is little chance they would be able to decipher the information [15].

6) *Active Monitoring:* Monitoring all inputs and communications to the database allows an operator to identify possible SQLi attacks. Once a system administrator identifies an attack, they can delete any unauthorized SQL queries and disable any unauthorized users of the system. Monitoring can be assisted with the use of machine learning systems to quickly identify and remove unauthorized accesses [15].

An operator may want to test their own system for SQLi vulnerabilities. By using tools created to test for vulnerabilities, an operator is able to find vulnerabilities without the risk of a security breach. Once a vulnerability is identified the operator can work quickly to fix the vulnerability before it becomes a larger security issue [15].

Maintaining applications and databases is a great way to prevent against vulnerabilities. Ensuring that you are continually updating and patching your software so that it stays current with new security trends is key to maintaining a secure database. Investing in software maintenance is key for any database operator. All components of a web application must be maintained, such as software, libraries, and plug-ins [15].

III. SIMULATION

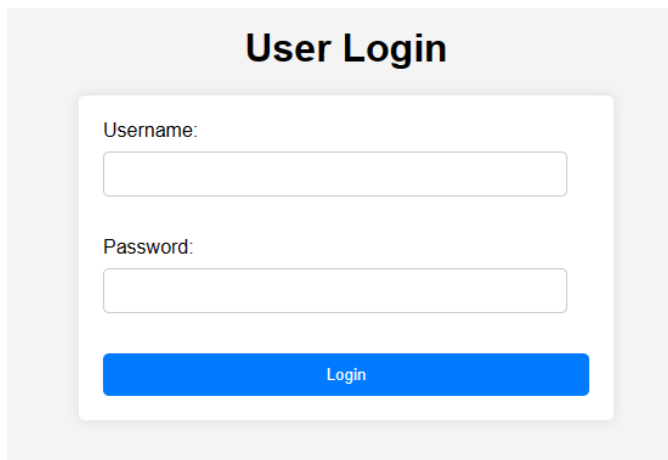
A. Methodology

To demonstrate the various types of SQL injection attacks, we need to prepare a test environment that allows for easy use of SQL injection queries to manipulate a backend database instance. For the purpose of this simulation, the environment requires lenient security policies and an unsafe SQL injection script, both of which are to not be deployed in a regular production setting.

The simulation utilizes the AWS cloud environment for deployment of an EC2 instance and a MYSQL relational database (RDS) to create a simple login page that stores login information as well as other information tied to each user.

Fig. 1 shows the user interface of the simulation. To note, the deployment of the infrastructure for this simulation is done through terraform resource modules. The MYSQL database is populated through SQL commands that can be run after connecting to the database. Furthermore, the instance hosts a basic webpage that is tied to a PHP script that runs each time a login attempt is made. Connecting to the instance through ssh allows for further configuration of the instance. This is where we installed the apache2 library, a few PHP libraries, and added the front-end files. While our simulation configures the resources for the instance manually, this can be done using an Amazon Machine Image (AMI).

With the environment deployed and configured, we are now able to test a variety of SQL injection attacks to crack into the backend database.



The image shows a web form titled "User Login". It has two input fields: "Username:" and "Password:". Below the "Password:" field is a blue button labeled "Login". The form is set against a light gray background.

Fig. 1. UI of the simulation

B. Procedures

1) *Error-Based SQLi*: The first attack that we simulated was an error-based attack. As noted previously, this attack may provide the bad actor with information about the database. In the simulation's case, we learned that the database utilizes SQL and is a MYSQL database. We also got a hint at the syntax that is used in the SQL query.

username: test'
password: w

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'w' at line 1

Fig. 2. Error from injection faulty SQL query

2) *Tautology*: Next, we simulated a tautology attack. This attack utilizes an always true Boolean statement to simulate a successful login condition. In our simulation of this attack, we were able to manipulate the SQL statement to provide a list all the entire user database containing information about each user. Fig. 3 shows what information was exposed through a tautology attack.

Name: Peter Smith

Username: pes71
Password: irrelevant
Date of Birth: 1999-07-25
Email: psmith@gmail.com
Phone Number: 417-123-4567
Address: 458 Oak St, Springfield, USA

Name: Alex McMoil

Username: alm470
Password: Ilovecats!
Date of Birth: 2003-11-30
Email: amcmoil3@outlook.com
Phone Number: 412-555-1180
Address: 123 Elm St, Pittsburgh, USA

Name: Emily Johnson

Username: emj133
Password: StrongPass123
Date of Birth: 1992-03-12
Email: ejohnson@yahoo.com
Phone Number: 617-909-6512
Address: 1491 Maple Ave, Boston, USA

Name: Luiz Santos

Username: lfr39
Password: messifan
Date of Birth: 1985-09-01
Email: luiz.santos@zoho.com
Phone Number: 305-963--0842
Address: 345 Pine St, Miami, USA

Fig. 3. Result from Tautology Injection

username: hithere

password: something' or '1' = '1

3) *Blind Time-Based SQLi*: Continuing with the simulation, we chose to simulate a blind time-based SQL injection attack. As previously mentioned, this attack is designed to create a delay in server response using a sleep function call. As expected, after clicking the login button, we had a delay in response from the database server that kept us waiting at the login page. Upon the completion of the delay, we received the resulting page from our query.

username: alm470' -SLEEP(2) or '1' = '1
password: anything

4) *Union-Based SQLi*: Lastly, we wanted to simulate a union-based SQL injection attack- an attack that is utilized to collect information about multiple databases that may exist in the backend. For our simulation, we are assuming that through brute force tactics, the bad actor has stumbled upon the correct name of another database table containing employee information. Having a database with employee information is standard in any company, so it's feasible that the bad actor has stumbled upon the correct name. Adding a 'union select' to the password field enables the bad actor to pull information from another database called 'employees.' Upon submitting the following login query, we landed on a page that contained database information from both the 'user' and 'employees' database. Fig. 4 shows what information can be exposed with a union-based attack.

username: hiagain

password: w' or '1' = '1' union select * from employees where '1'='1'

```
Name: Peter Smith
Username: pet71
Password: irrelevant
Date of Birth: 1999-07-25
Email: psmith@gmail.com
Phone Number: 417-123-4567
Address: 458 Oak St, Springfield, USA

Name: Alex McMill
Username: alex470
Password: Illovecats!
Date of Birth: 2003-11-30
Email: alexcmill3@outlook.com
Phone Number: 412-555-1180
Address: 123 Elm St, Pittsburgh, USA

Name: Emily Johnson
Username: emj133
Password: StrongPass123
Date of Birth: 1992-03-12
Email: ejohnson@yahoo.com
Phone Number: 617-899-6512
Address: 1491 Maple Ave, Boston, USA

Name: Luiz Santos
Username: ls39
Password: messifan
Date of Birth: 1985-09-01
Email: luiz.santos@zoho.com
Phone Number: 305-963-0842
Address: 345 Pine St, Miami, USA

Name: John Doe
Username: JD123456
Password: 789 Pine St, Oakville
Date of Birth: 534-120-4439
Email: john.doe@company.com
Phone Number: 123-45-6789
Address: 65000.00

Name: Priya Patel
Username: PP189012
Password: 456 Elm St, Maplewood
Date of Birth: 216-754-8980
Email: priya.patel@company.com
Phone Number: 987-65-4321
Address: 87000.00

Name: Mohammed Ali
Username: MA654321
Password: 321 Cedar St, Riverside
Date of Birth: 317-009-2372
Email: mohammed.ali@company.com
Phone Number: 654-32-1098
Address: 100000.00
```

Fig. 4. Result from union-based injection

IV. CONCLUSION

In this paper, we have reviewed what SQL and SQL injections are, the different types of SQLi attacks, and how to prevent them. We have discussed the dangers of SQLi attacks and how these attacks are delivered.

As seen in the simulation, there are a variety of attacks that bad actors can utilize, individually or collaboratively, to identify and extract weaknesses in database infrastructure. Starting with error-based attacks to understand the syntax and language used to query the database, the bad actor can utilize brute force methods to crack into the database to extract, modify, or even delete information in the database causing serious infrastructure damage.

Companies deploying production environment applications consisting of web and database servers must be extremely cautious of vulnerabilities that may exist within their database infrastructure. SQL injections attacks are one of the most common forms of database attacks, and as such, emphasizing the importance for those deploying production level databases to be well-informed of the risks associated with SQL during the database implementation process.

REFERENCES

[1] "Us man 'stole 130m card numbers'," *news.bbc.co.uk*, Aug. 2009. [Online]. Available: <http://news.bbc.co.uk/2/hi/americas/8206305.stm> (visited on 09/04/2020).

[2] J. Elfström, *Did little bobby tables migrate to sweden?* Alicebobandmallory.com, 2010. [Online]. Available: <https://alicebobandmallory.com/articles/2010/09/23/did-little-bobby-tables-migrate-to-sweden> (visited on 12/15/2019).

[3] *Lulzsec hacks sony pictures, reveals 1m passwords unguarded* — *electronista*, web.archive.org, Jun. 2011. [Online]. Available: <https://web.archive.org/web/20110606051745/http://www.electronista.com/articles/11/06/02/lulz.security.hits.sony.again.in.security.message> (visited on 04/13/2024).

[4] P. Loshin and J. Sirkin, *What is structured query language (sql)?* TechTarget, Feb. 2022. [Online]. Available: <https://www.techtarget.com/searchdatamanagement/definition/SQL>.

[5] S. Jaiswal, *Sql query examples and tutorial*, DataCamp.com, Jul. 2022. [Online]. Available: <https://www.datacamp.com/tutorial/sql-query-examples-and-tutorial> (visited on 04/12/2024).

[6] D. Pandey, *What is query in sql?* Scaler Topics, Oct. 2022. [Online]. Available: <https://www.scaler.com/topics/what-is-query-in-sql/>.

[7] V. Kanade, *What is sql? definition, elements, examples, and uses in 2022*, Spiceworks, Jul. 2022. [Online]. Available: <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-sql/>.

[8] OWASP, *Sql injection*, OWASP, 2024. [Online]. Available: https://owasp.org/www-community/attacks/SQL_Injection.

[9] W. G. Halfond, J. Viegas, and A. Orso, "A Classification of SQL-Injection Attacks and Countermeasures," in *Proceedings of the IEEE International Symposium on Secure Software Engineering*, Arlington, VA, USA, Mar. 2006.

[10] B. Hofesh, *Sql injection attack: How it works, examples and prevention*, Bright Security, Apr. 2022. [Online]. Available: https://brightsec.com/blog/sql-injection-attack/#impact_attack.

[11] *Types of sql injection (sqli)*, GeeksforGeeks, Aug. 2022. [Online]. Available: <https://www.geeksforgeeks.org/types-of-sql-injection-sqli/>.

[12] L. Ma, D. Zhao, Y. Gao, and C. Zhao, "Research on sql injection attack and prevention technology based on web," in *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)*, 2019, pp. 176–179. DOI: 10.1109/ICCNEA.2019.00042.

[13] L. Muhammad Aminu, A. Sultan, and A. Shakiru, "Systematic literature review on sql injection attack," *International Journal of Soft Computing*, vol. 11, pp. 26–35, Jan. 2016.

[14] U. Berkeley, *How to protect against sql injection attacks* — *information security office*, security.berkeley.edu, 2023. [Online]. Available: <https://security.berkeley.edu/education-awareness/how-protect-against-sql-injection-attacks>.

- [15] C. Kime, *How to prevent sql injection attacks — esecurity planet*, eSecurityPlanet, May 2023. [Online]. Available: <https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks/>.