Historical and Modern Methods of Combating SQL Injection

Lilly Sharples
The University of Georgia
Athens, Georgia, USA
lillysharples@uga.edu

ABSTRACT

Consumers give their data across web applications daily, without realizing how insecure their data may be. It becomes a problem when a consumer expects this data to be kept private, but it is accessed or leaked beyond their knowledge. SQL injection is a common technique of accessing this data, by maliciously inputting queries to unprotected web applications. This paper highlights five known techniques developers can follow to prevent SQL injection. Additionally, three lesser-known techniques recently researched are described, relating to blockchain and machine learning. Ethical implications of incorporating machine learning when dealing with accessing private data are also considered.

CCS CONCEPTS

• **Information systems** → **Structured Query Language**; • **Security and privacy** → *Cryptography*; **Human and societal aspects of security and privacy**; **Intrusion detection systems**; • **Computing methodologies** → *Reinforcement learning*; • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

SQL Injection, Security, Blockchain, Machine Learning

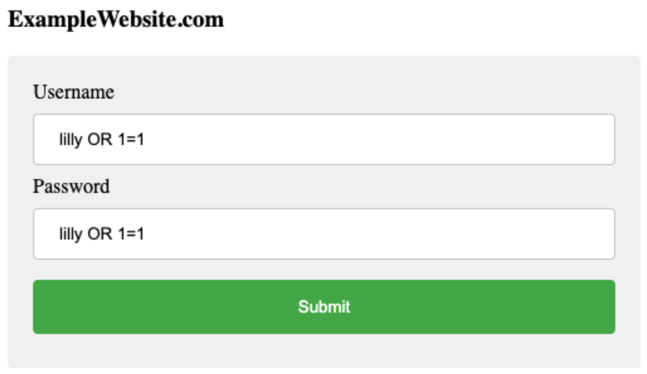
ACM Reference Format:
Lilly Sharples. 2021. Historical and Modern Methods of Combating SQL Injection . In *Proceedings of December*. ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

As the daily lives of consumers around the world become more reliant on the internet, it is no surprise that cyber attacks remain prevalent and threatening. One of the most widely used methods of hacking is SQL injection, where a hacker inputs SQL statements to text fields, dropdowns, pop-up windows, etc. on websites or web applications. This allows them to gain unauthorized access to the back-end database of a web application[2], and even manipulate the database or server. From there, the hacker can access and utilize private information such as usernames, passwords, addresses, credit card numbers, and more. Previously ranked by OWASP as the number one security risk, SQL injection is still one of the most popular methods of web hacks, now ranked as the third most popular[**?**]. With SQL injection moving down two spots, we can see that new breakthroughs in technology have resulted in hackers operating in different ways than before. Though modern technology provides hackers with new ways to steal data, it also equips developers with new methods of detecting and preventing these hacks. The remainder of this paper is organized as the following: Section II provides insight into how a hacker can perform SQL injection. Section III provides explanations and examples of historical, more widely known and used methods to combat SQL injection. Section IV introduces more recent methods to detect, prevent, and exploit SQL injection. This research is concluded and future work is considered within Section V.

2 SQL INJECTION BACKGROUND

There exist certain ways to format an SQL statement so that it always returns true. When a web application does not contain methods to detect and prevent these statements, hackers can easily enter a statement and return all possible rows in the database. One way to do this is the example of *OR 1=1* , a valid statement that will always return true. In the example below, the input *lilly OR 1=1* to the Username field results in the SQL statement *SELECT * FROM UserData WHERE Username = lilly OR 1=1 AND Password = lilly OR 1=1;*. If this statement is executed, all usernames and passwords from the UserData table would be returned.


ExampleWebsite.com

Username: lilly OR 1=1

Password: lilly OR 1=1

Figure 1: Example of SELECT * FROM UserData WHERE Username = lilly OR 1=1 AND Password = lilly OR 1=1

Another valid statement that will always return true is *" OR " "="*. In this example, SQL statement *SELECT * FROM UserData WHERE Username = " OR ""=" AND Password = " OR ""=";* will be sent to the server, again returning all usernames and passwords.

A final method of SQL injection is entering a batch, or group, of semicolon separated SQL statements to an input field. If the hacker

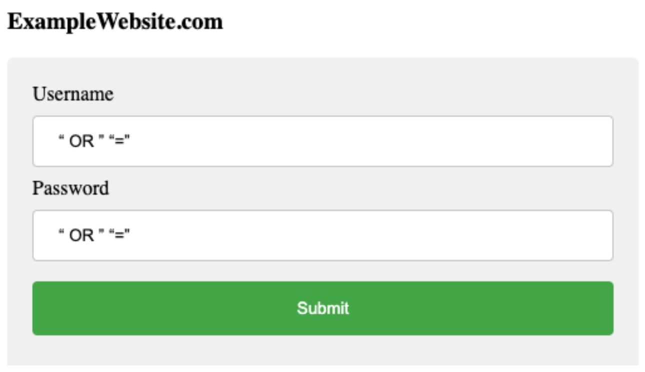


Figure 2: Example of SELECT * FROM UserData WHERE Username = " OR ""=" AND Password = " OR ""=";

has previously used SQL injection to access the database, they will know the name of the table. Once they know the name, there are commands that can be used to manipulate the database itself. Following the same example, the hacker has determined that the table name is 'UserData'. Now, they can enter 'lilly; DROP TABLE UserData', creating the SQL command *SELECT * FROM UserData WHERE Username = lilly; DROP TABLE UserData;* , resulting in the UserData table being deleted.

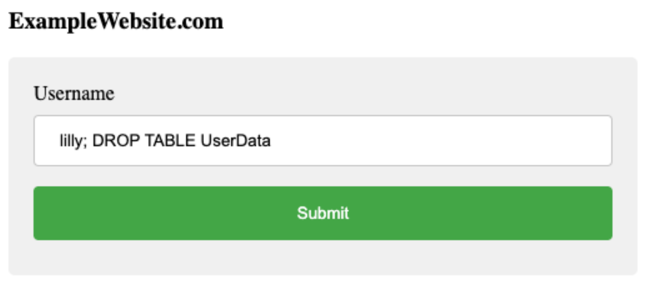


Figure 3: SELECT * FROM UserData WHERE Username = lilly; DROP TABLE UserData;

It is obvious to see how detrimental these seemingly basic hacks could be for a company, especially if confidential customer information is leaked and then deleted. Luckily, there are numerous ways to combat SQL injection hacks.

3 HISTORICAL METHODS TO COMBAT SQL INJECTION

Historically, programmers have used techniques such as prepared statements, parameterized queries, stored procedures, input validation, and escaping/encoding user input to combat SQL injection [8]. Companies have also been taught the importance of keeping their software up to date, error handling, and using the principle of least privilege. Security testing tools such as penetration testing additionally ensure that they have prevented SQL injection to the best of their abilities before deploying web applications. While these methods work to an extent, hackers have still been able to get around them. This section will further detail a few of these techniques.

3.1 Prepared Statements and Parameterized Queries

In a prepared statement, the developer prepares a template for the SQL statement to follow. This template can also be referred to as a base query, with different parameter values provided by the web application user. The parameters entered by the user have to exactly match the field to be searched for. Consider the previous example of an attacker attempting to perform an SQL attack in a web application where they are prompted to enter a username. Inputting *lilly OR 1=1* to the Username field would take away the injection vulnerability by searching the table for a literal Username of *lilly OR 1=1*. The use of a template or base query in a prepared statement is especially beneficial when similar SQL statements will be repeated, changing the parameters to reflect different users. Many programming languages even have built in methods for utilizing prepared statements and parameterized queries, such as Java.sql PreparedStatement(). By keeping all of the SQL code within the web application, it keeps the application database-independent for the most part. [5]

3.2 Stored Procedures

Stored procedures are similar to the SQL template in a prepared statement, where the developer puts in the parameterized query. The main difference is that unlike prepared statements, the stored procedure code is both defined and stored in the database, then called from the web application. For this reason, stored procedures are still able to be injected, though preventable with correct development. Again, multiple programming languages have their own implementation of stored procedures, such as java.sql CallableStatement.

3.3 Input Validation

When discussing input validation, it is important to differentiate between client-side and server-side. Client-side input validation is beneficial for improving the user experience, such as aiding non-malicious customers when entering input. If only client-side input validation is used, a hacker could, for example, alter the javascript code within their browser to remove it altogether[8].

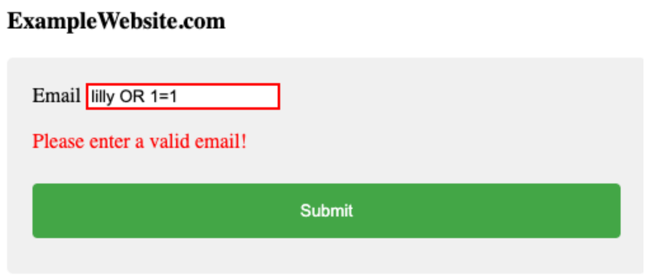


Figure 4: An example of client-side user input validation

Server-side input validation, however, ensures that user input matches what is expected within the query. This could be done using a regular expression, making it so that user input has to match the pattern of expected input in order to make a call to the database. It is recommended to use input validation on top of any other SQL injection defense method to maximize security [5].

3.4 Escaping or Encoding User Input

A lesser used method to defend against SQL injection is escaping or encoding user input. Different database management systems (DBMS) support schemas to escape characters, so by escaping all user input to match that schema vulnerabilities can be avoided. OWASP Enterprise Security API provides free, open-source database encoders for DBMS such as Oracle and MySQL (OWASP). However, escaping is not always guaranteed to work, and is dependent on the DBMS used. Encoding the user input, specifically hex-encoding, means each character provided as user-input is translated to hexadecimal.

3.5 Security Testing Tools

Before publishing a web application or program, code should be reviewed. A step further from human code review is software that automatically inspects code to look for security vulnerabilities. Source code analysis tools, commonly referred to as Static Application Security Testing (SAST) tools, are able to perform these inspections. They can be repeatedly run as a part of continuous integration. These tools are beneficial in identifying SQL injection vulnerabilities within code, though weaker in identifying other types of security issues, such as authentication. SAST tools do have limitations such as difficulty searching for vulnerabilities in one file without access to all libraries, files, or compilation instructions, as well as trouble proving why an identified vulnerability is actually an issue[6]).

4 RECENT METHODS TO COMBAT SQL INJECTION

Blockchain and machine learning are two modern technology buzzwords that have found applications in detecting, preventing, and even exploiting SQL injection. Blockchain encryption techniques can be used to provide a more secure encryption between the client and SQL database. Machine learning can be used to train a system to classify when there is an expected vulnerability. Another use case of machine learning can be to exploit SQL injection, potentially automating penetration testing. Incorporating machine learning to a database with private information raises an ethical concern, which will be discussed later in this section.

4.1 Stellar Blockchain Key Pair Encryption

The use of blockchain in SQL injection has been experimented with using the open-source and decentralized Stellar blockchain by researchers at the University of Science and Technology, Jilin, China. Stellar has a public and private key using Ed25519 public-key signatures, designed to be faster and more secure than previous encryption methods (Bernstein et al.).

Every valid SQL keyword for the database is given a public key, and stored in an array. Upon user input, the SQL statement is tokenized, and each token is checked to confirm if there is a public key with a private key to match. The tokenizing and checking process is done within an intermediate, proxy server, implemented with ASP.NET, as an additional layer of security. Upon confirmation that all SQL keywords have valid public and private keys, the SQL statement is sent to the database server to be processed. Within the study, the use of Stellar Key Pair was one of two methods able to successfully prevent all SQL Injection attempts [1].

4.2 Machine Learning for Detection

Machine learning applied to SQL injection has been seen to accurately detect hackers by classifying requests in a knowledge-based system. A knowledge-based system is based on a knowledge-based agent, using reasoning and an internal knowledge base to decide which action to take[7]. Detection involves training the system on a website log where SQL injection has occurred, creating a knowledge base for malicious and benign requests as part of the training[3]. In this case, the knowledge base contains scenarios of proven malicious or benign instances, specifically the features related to those instances. The knowledge base can then be compared to the log file of the website in question to classify whether malicious behavior is detected.

4.3 Machine Learning for Automated Testing

Reinforcement learning refers to an agent learning from rewards and punishments based on its behaviors. Over time, the agent has to learn which actions were likely responsible for the reinforcements and punishments, changing its methodology to aim for more rewards and less punishments in future attempts[7]. In the context of teaching an agent to perform SQL injection, a reward would be given if the agent was able to correctly exploit the database. Treating SQL injection like a game, autonomous agents have been trained with reinforcement learning to send queries, analyze the responses from the database, and alter their queries until achieving their goal- access to the database[4].

Researchers at the University of Oslo were able to train an agent by giving a +10 reward for an action that exploits the database or a -1 reward for any other action, meaning the agent has an overall positive reward so long as they can exploit the database in less than ten actions[4]. This incentive system has the goal of teaching an agent to take advantage of actions that may overlap, resulting in a reduction of the number of steps. This application of machine learning has the greatest use case in automated penetration testing, reducing the need for a human to perform this necessary task to ensure security before deploying a web application.

However, this application of machine learning also has the greatest level of ethical concern. If it is possible to develop automated SQL penetration testing, then hackers could have the ability to develop the same agent, but for malicious intent. It could be possible for these agents to become better than humans at performing SQL injections, as well as determining which web applications are more prone to them. A hacker could deploy these agents to a countless number of websites, retrieving private data from all over the world in a matter of minutes.

5 CONCLUSION AND FUTURE WORK

No consumer or company wants their personal information to be stolen, an obvious incentive to prevent SQL injection wherever possible. There exist a multitude of methods, such as those detailed above, available to developers. These range from basic to advanced, for purposes such as preventing, testing, and identifying SQL injection. New methods are being researched and developed constantly, showing that there is still a need to prevent this method of hacking, more advanced than what is currently available.

With regards to future work, an advancement could be the combination of blockchain encryption and machine learning detection or exploitation. If machine learning and high powered computers are able to decrypt Ed25519 encryption keys and exploit the database, it will be necessary to further advance this method of blockchain encryption.

These new methods additionally lead to new questions and concerns, specifically regarding the use of machine learning to automate security testing. It raises the question of trust, as to how we can ensure these agents are inherently good natured, without malicious intent.

REFERENCES

[1] O. Abimbola and Zhanfang Chen. 2020. Prevention of SQL Injection Attack Using Blockchain Key pair based on Stellar. *European Scientific Journal* 16 (12 2020), 92–104. https://doi.org/10.19044/esj.2020.v16n36p92

[2] Khaleel Ahmad. 2010. Classification of SQL Injection Attacks. *VSRD Technical Non-Technical Journal* 1 (05 2010), 236–242.

[3] Muhammad Amirulluqman Azman, Mohd Fadzli Marhusin, and Rossilawati Sulaiman. 2021. Machine Learning-Based Technique to Detect SQL Injection Attack. 17, 3 (Mar. 2021), 296–303. https://doi.org/10.3844/jcssp.2021.296.303

[4] Laszlo Erdodi, Åvald Åslaugson Sommervoll, and Fabio Massimo Zennaro. 2021. Simulating SQL Injection Vulnerability Exploitation Using Q-Learning Reinforcement Learning Agents. *CoRR* abs/2101.03118 (2021). arXiv:2101.03118 https://arxiv.org/abs/2101.03118

[5] Jim Manico and Jakub Mackowski. 2021. SQL injection prevention cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

[6] OWASP. 2021. Owasp Top Ten. https://owasp.org/www-project-top-ten

[7] Stuart J. Russell. 2020. *Artificial Intelligence: A Modern Approach* (4 ed.). Pearson Education.

[8] Brian Vermeer. 2021. SQL Injection Cheat Sheet: 8 best practices to prevent SQL injection. https://snyk.io/blog/sql-injection-cheat-sheet/