SQL Injection Prevention Techniques

Angela Darden
Computer Science Department
Hampton University
Hampton, VA

Abstract – SQL injections are one of the most common and dangerous vulnerabilities found in web applications, even though they have been well documented for decades. This paper explores the effectiveness of common prevention techniques against SQL injection attacks, including input validation, parameterized queries, and prepared statements. To demonstrate, a vulnerable web environment was created using Damn Vulnerable Web Application (DVWA) to simulate attacks and observe how each defense method withstands different injection attempts. The results will show the strengths and weaknesses of each approach when tested against real-world attack patterns. In addition to testing, this research highlights the relevance of SQL injections in today's cybersecurity environment, shown by their inclusion in the OWASP Top 10 [4]. By demonstrating how easily unsecured applications can become victims of attacks and how effective proper countermeasures can be, this paper highlights the importance of implementing secure coding practices in modern web development.

I. Introduction

Having accounts to services that you commonly use to make accessing their services more convenient is something that we don't even think about. But what if you knew that the organization behind the storage might not have proper security measures in place? Your information could be at danger of being captured during an SQL injection attack. An SQL injection attack allows for an attacker to input various "queries" or commands to see what the service will tell them about what is in the database. It can also allow for them to gain administrative privileges. This type of attack is very dangerous and requires proper safeguards to be in place to protect customers and their stored sensitive data. How can this be combatted? There are different ways that this problem can be combatted, such as input validation, parameterized queries, and prepared statements.

Input validation is ensuring that an attacker cannot enter in a query that is outside of the specified format. This prevents from custom queries from being entered and accepted.

Parameterized queries are allowing user entered values to be passed as parameters into queries. This allows for what has been entered to be separated from the SQL query, what has been entered can't contain malicious code since the parameter will be treated as a value and is checked.

Prepared statements are reusable queries that allows for an SQL command to be executed safely. This is also another way for user inputted information to be separated from the SQL query.

The average person will not think about if their information is safe against an SQL injection attack, let alone think of how to determine if it is. Attackers will pick web applications either at random or one of their choice and start inputting queries to see what the application will output. Many won't know that their information has been compromised until that organization sends out a notice or until they get a message stating that their account has been accessed.

This paper will look into different prevention techniques to determine their effectiveness at protecting customers and their data. As well as the importance of having secure coding practices implemented into web applications.

A. Problem Statement

SQL injections are common vulnerability that is exploited that most don't have proper protections in place to prevent. But do the recommended protection methods protect as well as they claim to? Web application developers implement these safeguards in hopes that their testing and secure coding practices hold up to the damage that can be done by attackers. The actions that are taken by attackers are random and cannot be predicted accurately. Sometimes by ensuring that one action can't be successfully completed, puts the potential that other action can be successfully completed.

II. Methodology

This study will use a combination of literary reviews and experiments to gather information that is directly related to the thesis. Each stage of the methodology will be explained in the order that follows:

A. Literary Review

We will discuss the dangers of SQL injection attacks and the importance of prevention techniques by referencing scholarly articles. The importance of having secure coding practices will also be discussed, also by referencing scholarly articles. These articles will give a basis to build off of with additional research and to either support or contradict the thesis and findings through multiple methods.

B. Experiments

The experiment will be myself using a vulnerable web environment to demonstrate how SQL injection attacks can be carried out and how to defend against them. The web environment will be used to simulate a website that has been created for normal use. In addition to the vulnerable web environment, a software that allows for web application security testing will be used. The protection techniques will be analyzed to determine if the results support or contradict the thesis.

III. Results/Raw Data

This section will cover the results defined in Section II, Methodology.

A. Literary Review

Many organizations use databases to store information, most commonly sensitive information such as login credentials. When you input your login information on a form and hit submit, it turns your information into an SQL statement. SQL (Structured Query Language) is the language that is used to communicate with databases. SQL allows for developers to retrieve, add, remove, update, and modify data in databases [1].

SQL injections occur when an attacker inputs queries with the intentions of trying to capture data in the database [3]. SQL injection attacks are very dangerous and should be taken seriously. Attackers can typically do this without encountering any security measures.

The impact of an SQL injection attack is felt on the user side and the organization side. An SQL injection attack damages the reputation of an organization and can cost them financially to rebuild a more secure database. The users no longer trust the company to secure their data and if their compromised financial data was used, the users must deal with an attacker having their information [2].

Ensuring that your database is protected from SQL injection attacks is just as important as securing the rest of your application. It is true that no web application is completely secure, but implementing defense techniques such as input filtering, input sanitization, and the use of prepared statements.

B. Experiment Results

To conduct an experiment on how effective different SQL injection prevention techniques are, I will be testing out the various techniques using a vulnerable web application and a software for web application security in a virtual machine. The vulnerable web application used is Damn Vulnerable Web Application (DVWA) and the web application security software is Burp Suite.

Damn Vulnerable Web Application or DVWA is a web application that is used to gain a better understanding of the importance of secure

coding practices. It allows for users to simulate various attacks in a simulated environment, such as brute force, command injection, cross-site scripting (XSS), and many more. DVWA also has four security levels, low, medium, high, and impossible. Each changes the vulnerability level of DVWA, low has no security measures and impossible is secure against all vulnerabilities.

DVWA has a default database that is created with user ID numbers, first names, last names, usernames, passwords, and avatar pictures. This created database will be used to simulate how organizations store user information.

Burp Suite allows for users to conduct security tests using toolkits. These toolkits allow for users to intercept and forward HTTP traffic, view HTTP traffic, modify HTTP requests, and various other security testing actions.

user_id	first_name	last_name	user	password	avatar
1	admin	admin	admin	5f4dcc3b5aa765d61d8327deb882cf99	/DVWA/hackable/users/admin.jpg
2	Gordon	Brown	gordonb	e99a18c428cb38d5f260853678922e03	/DVWA/hackable/users/gordonb.jpg
3	Hack	Me	1337	8d3533d75ae2c3966d7e0d4fcc69216b	/DVWA/hackable/users/1337.jpg
4	Pablo	Picasso	pablo	0d107d09f5bbe40cade3de5c71e9e9b7	/DVWA/hackable/users/pablo.jpg
5	Bob	Smith	smithy	5f4dcc3b5aa765d61d8327deb882cf99	/DVWA/hackable/users/smithy.jpg

Figure 1: Default DVWA Database

Low Level

```
ID: ' OR '1'='1
First name: admin
Surname: admin

ID: ' OR '1'='1
First name: Gordon
Surname: Brown

ID: ' OR '1'='1
First name: Hack
Surname: Me

ID: ' OR '1'='1
First name: Pablo
Surname: Picasso

ID: ' OR '1'='1
First name: Bob
Surname: Smith
```

Figure 2: First returned output of Low Level

Fatal error: Uncaught mysqli_sql_exception: You have an error in your SQL syntax;

Figure 3: SQL Syntax Error

```
ID: ' UNION SELECT NULL, NULL #
First name:
Surname:
```

Figure 4: Second returned output of Low Level

```
ID: ' UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Figure 5: Third returned output of Low Level

Attack Attempt	**Defense Technique**	**Result**
‘ OR ‘1’=’1	None	Returns all user data
‘ UNION SELECT NULL, NULL --	None	SQL syntax error
‘ UNION SELECT NULL, NULL #	None	Returned format
‘ UNION SELECT username, password FROM users #	None	SQL error, no column named username found
‘ UNION SELECT user, password FROM users #	None	Returned list of usernames and passwords

Figure 6: Low Level Results

Medium Level

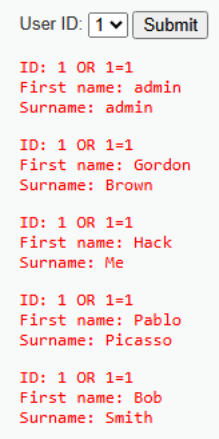

Figure 7: First returned output of Medium Level

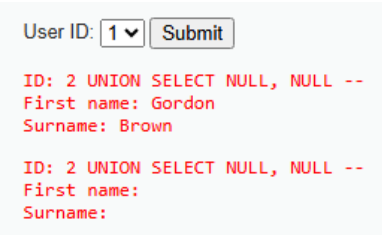

Figure 8: Second returned output of Medium Level

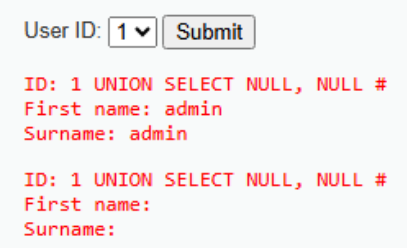

Figure 9: Third returned output of Medium Level

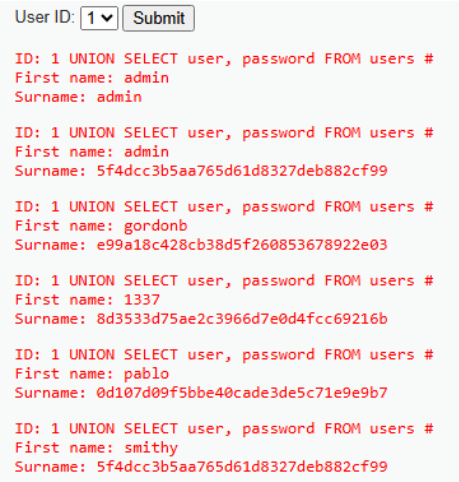

Figure 10: Fourth returned output of Medium Level

Attack Attempt	Defense Technique	Result
‘ OR ‘1’=’1	Input Filtering	SQL syntax error
OR 1=1	None	Returns all user data
‘ UNION SELECT NULL, NULL --	Input Filtering	SQL syntax error
UNION SELECT NULL, NULL --	None	Returns user data for entered ID
‘ UNION SELECT NULL, NULL #	Input Filtering	SQL syntax error
UNION SELECT NULL, NULL #	None	Returns user data for entered ID
‘ UNION SELECT username, password FROM users #	Input Filtering	No column named username found
‘ UNION SELECT user, password FROM users #	Input Filtering	SQL syntax error
UNION SELECT user, password FROM users #	None	Returns all user data

Figure 11: Medium Level Results

High Level

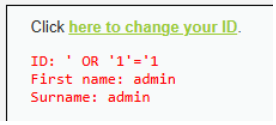

Figure 12: First returned output of High Level

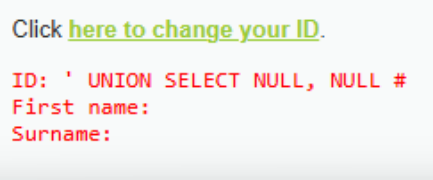

Figure 13: Second returned output of High Level

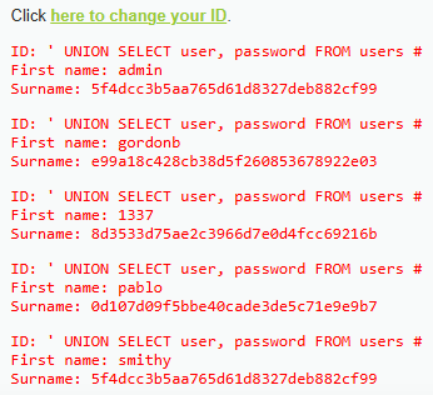

Figure 14: Third returned output of High Level

Attack Attempt	**Defense Technique**	**Result**
‘ OR ‘1’=’1	Limited filtering	Returns admin data (unexpected)
OR 1=1	Input sanitizing	Blank page, no data returned
‘ UNION SELECT NULL, NULL --	Input filtering	SQL syntax error
UNION SELECT NULL, NULL --	Input sanitizing	Blank page, no data returned
‘ UNION SELECT	Input filtering	Returns format but a blank page
NULL, NULL #		
UNION SELECT NULL, NULL #	Input sanitizing	Blank page, no data returned
‘ UNION SELECT username, password FROM users #	Input filtering	No column named username found
‘ UNION SELECT user, password FROM users #	Input filtering	Returns all user data (unexpected)
UNION SELECT user, password FROM users #	Input sanitizing	Blank page, no data returned

Figure 15: High Level Results

Impossible Level

Attack Attempt	**Defense Technique**	**Result**
‘ OR ‘1’=’1	Prepared statements	Blank page, no data returned
OR 1=1	Prepared statements	Blank page, no data returned
‘ UNION SELECT NULL, NULL --	Prepared statements	Blank page, no data returned
UNION SELECT NULL, NULL --	Prepared statements	Blank page, no data returned
‘ UNION SELECT	Prepared statements	Blank page, no

NULL, NULL #		data returned
UNION SELECT NULL, NULL #	Prepared statements	Blank page, no data returned
‘ UNION SELECT username, password FROM users #	Prepared statements	Blank page, no data returned
‘ UNION SELECT user, password FROM users #	Prepared statements	Blank page, no data returned
UNION SELECT user, password FROM users #	Prepared statements	Blank page, no data returned

Figure 16: Impossible Level Results

IV. Analysis

An analysis of the results of the various prevention techniques on the different DVWA security levels. Below is a description of each of the security levels of DVWA:

Figure 17: Description of DVWA Security Levels

A. Low Level

The results from the low level shows that basic SQL injection queries be executed, if they follow the syntax rules. The low level has no protections in place, making it very vulnerable. The low level allows for users to enter in their user ID and upon submission, their entry appears in the URL bar. This slight mistake allows for users to see how their request is being processed and allows for malicious entries.

As seen in the table, there were various queries that allowed for the stored user data to be displayed. The query *‘ OR ‘1’=’1*, essentially creates a true statement that returns all the user data.

It was seen that some queries returned syntax errors, the syntax errors indicate that the database used does not match the query syntax used. Syntax errors give you clues about what database is used and allows you to change your statements so that they will go through with no errors. Some of the attack attempts use -- or #, these are comments. Since the # symbol worked, this can suggest that the database used is MySQL. The -- comment format cause errors, this can indicate that it was not used correctly for the database. These syntax errors allow for attackers to tune their attacks specifically to the database.

B. Medium Level

The medium level introduces more protection techniques that were in the low level. As seen in the table, there were some queries that were able to return data. But some of the commands that were working on the low level now require some modifications to work on the medium level. The medium level attempts to remove the slight vulnerability in the URL bar and allowing users to enter in their own ID. There is now a dropdown menu that lists out the different user IDs and upon submission, the URL bar does not show the user’s input. With this vulnerability gone, a new method has to be introduced to input queries.

Burp Suite allows for users to intercept and modify HTTP traffic requests. This will be used to submit custom queries in attempt to get the user IDs and passwords. The number that is selected from the dropdown, with intercept

enabled, will allow for the request to be modified. For example, the user ID 1, will be shown as id=1, this can be modified to be a custom query and forwarded instead to see the response.

As seen in the table, the queries that include quotations are failing due to input filtering. This is because the SQL query already has quotations around the input. For the first query for example, *‘ OR ‘1’=’1* worked when there were no quotes, but now that there are quotes already inputted, it results as this *‘ ‘ OR ‘1’=’1’*; leading to the syntax error. The query *OR 1=1*, works due to the removal of quotations.

C. High Level

The high level attempts to provide more protections than the medium level. Instead of a dropdown menu there is a button to click that will prompt a secondary window to open. In that secondary window there is the ability to enter in your own queries. There are some queries that return data, some that give syntax errors, and some that return nothing.

A query that was used to return all data now only returns the admin data, *‘ OR ‘1’=’1*. Returning the admin data can be more detrimental than returning regular users. This query returning the admin data was unexpected, potentially indicating an issue with the security measures in place on the high level.

Interestingly, the only query that returned all the data was *‘ UNION SELECT user, password FROM users #*, which is different from the other levels. This being the only query that returned information suggests that input sanitization was not applied thoroughly, allowing some UNION statements to succeed.

D. Impossible Level

The impossible provides the most protections against SQL injections. As seen in the table, none of the queries were able to produce any output for the user to see, for both quotation-based and UNION-based queries. Every attack attempt resulted in a blank page with no data returned, indicating that the input was securely handled and the structure of the query was not affected.

The results suggest that the web application is secure by using secure coding practices such as prepared statements and parameterized queries, these separate user input from SQL logic. Unlike the previous levels, where the user input could change the query and return unintended data, the impossible level ensures that all input data is treated as literal data. The provides the most protections against SQL injection attacks.

With the lack of output across all attempt attacks, this proves that the impossible level provides the most protection, serving as the standard that all developers should follow when securing interactions with databases.

V. Conclusion

In conclusion, SQL injection attacks still remain as a relevant threat to web applications. Organizations still continue to use databases to store your information, such as login credentials. If an attacker is able to successfully access the information stored in the database, all user information is at risk.

By using Damn Vulnerable Web Application (DVWA) to simulate different versions of a web application, from highly vulnerable to secure, it shows the importance of ensuring that a web application is secure.

The low level allowed for basic SQL injection queries to execute, proving that this level is the most vulnerable. The medium level provided more protection against basic SQL injection queries, but to successfully return data it required more complex queries.

The high level, it was slightly more secure than the previous level. A cosmetic change to how input is handled does not guarantee that the input is more securely handled unless security is upheld on the backend.

The last level, impossible, this level provided the most protections against SQL injection attacks.

The impossible level is how all developers should strive for when developing their backend.

Overall, the results show that strong backend security are crucial for keeping your information safe against SQL injection attacks. But you should not just rely on the developers to keep your information safe, you must remain aware of where and who you trust your information with.

VI. References

[1]. Abdullayev, V., & Chauhan, A. S. (2023). SQL injection attack: Quick view. *Mesopotamian journal of Cybersecurity*, *2023*, 30-34. https://doi.org/10.58496/MJCS/2023/006

[2]. Alenezi, M., Nadeem, M., & Asif, R. (2021). SQL injection attacks countermeasures assessments. *Indonesian Journal of Electrical Engineering and Computer Science*, *21*(2), 1121-1131. http://doi.org/10.11591/ijeecs.v21.i2.pp1121-1131

[3]. Kareem, F. Q., Ameen, S. Y., Salih, A. A., Ahmed, D. M., Kak, S. F., Yasin, H. M., ... & Omar, N. (2021). SQL injection attacks prevention system technology. *Asian Journal of Research in Computer Science*, *6*(15), 13-32. https://doi.org/10.9734/ajrcos/2021/v10i330242

[4]. *OWASP Top 10:2025*. (n.d.). OWASP. https://owasp.org/Top10/2025/