

11/4/2024

SQL Injection



Rachel Henninger & Kaci Craycraft

A SQL injection or SQLi attack is the insertion of a SQL query into a user input on an application or website that exploits vulnerabilities and allows hackers to read, modify, execute, and recover information that the hacker should not have access to. By using this method, existing data can be tampered with such as deleting transactions or changing balances and could even expose sensitive information about the website's users. This form of attack is fairly common with older PHP and ASP applications due to their interface and close connection to a database. These attacks can be injected through user inputs, cookies, or HTTP headers. The severity of the attack is limited to the attacker's skill, imagination, and the amount of security measures a developer has instituted.

Here is an example of vulnerable PHP code. The username and password variables are placed directly from the POST request into the query to be sent to the database. This allows malicious SQL injections inside of the username or password field that allow a hacker to gain access to accounts whose username is known.

```
1      $username = trim($_POST['username']);
2      $password = $_POST['password'];

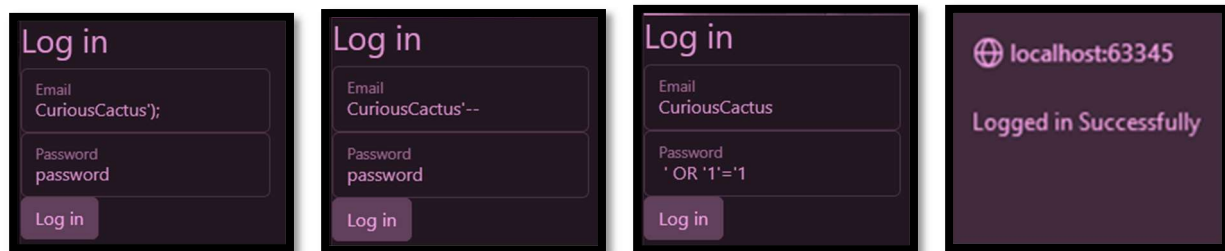
3      $query = "SELECT * FROM LOGINS WHERE (USERNAME = ?) AND (PASSWORD = ?)";
4      $conn->real_query($sql);
5      $result = $conn->use_result();

6      if($result)
7      {
8          echo "<script>alert('Logged in successfully');</script>";
9      }

10     else
11     {
12         echo "<script>alert('Invalid Credentials');</script>";
13     }
```

There are three different types of SQL injection attacks, Classic, Inferential, and Out-of-Band. Classic attacks are extremely common as they are the easiest to perform and they come in two forms. The first is an Error-Based attack, where the attacker will try to directly change the query tied to whatever input they are using. For example, they could type “%’--“ into the

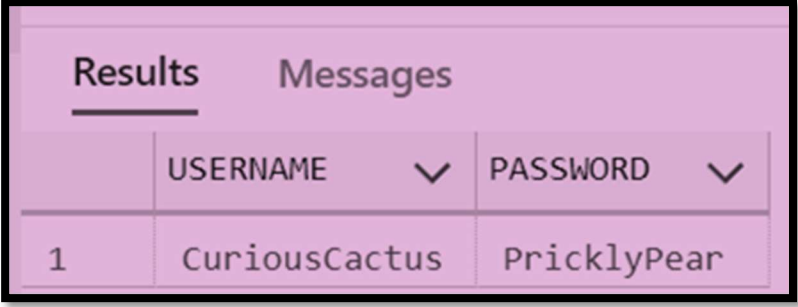
username input to attempt to yield a full list of users with the wildcard symbol and the single quote will end the query. The “--” syntactically represents the beginning of a comment in SQL and in this case escapes the remainder of the line, which validates password input. The second method of Classic injection attacks is known as a Union-Based attack, where the attacker uses a union operator to append to the existing query, usually to retrieve sensitive information about the database itself, including, but not limited to - the table names, the administrator’s username, and the SQL version being used. With this information, the attacker would be able to exploit database vulnerabilities via the determined administrator account.



The first two examples above demonstrate error-based attacks, resulting in any password yielding results. The third example demonstrates a Boolean inferential attack and how an individual can manipulate the SQL command using the password input field. In SQL, 1=1 always returns true for matching characters; therefore, this query also returns CuriousCactus from the database and allows the hacker access to their account, as demonstrated below. This method can be used to find usernames in a database by fuzzing the username input field with different strings.

```
1 SELECT * FROM LOGINS WHERE (USERNAME = 'CuriousCactus');') AND (PASSWORD = 'password');
2 SELECT * FROM LOGINS WHERE (USERNAME = 'CuriousCactus'--') AND (PASSWORD = 'password');
3 SELECT * FROM LOGINS WHERE (USERNAME = 'CuriousCactus') AND (PASSWORD = ' ' OR '1'='1');
```

Here, the code's query is run in Azure, where we can see the list of results returned. While today, SQL Injection is very difficult to replicate in a



Results		Messages	
	USERNAME		PASSWORD
1	CuriousCactus		PricklyPear

PHP environment, due to advancements across versions that resulted in built-in protections, these methods should not be relied on as they are not well understood, and don't exist in every environment. It is important to protect inputs from malicious SQL manipulation inside of the PHP code and the best way to achieve this lies in PDO. PDO prepares are very secure when utilized properly.

Inferential attacks take longer to execute than Classic attacks and thus are used less often. These attacks analyze the behavioral patterns of a database rather than directly gathering the contents of the database itself. This type of attack also comes in two forms, a Boolean and a Time-Based attack. A Boolean inferential attack takes place when an attacker sends a query to a database looking for either an OK or failed response. This can be found in the HTTP response itself, where modifications to the will be present upon success, and absent upon failure. By using Booleans in queries when probing the database, hackers may slowly acquire information such as the names of tables letter-by letter. They will test each index for every letter until an 'OK' response is received and repeat the process until the entire table name has been acquired. A time-based attack on the other hand, queries a *wait* function to the database. Much like the Boolean type of attack, it is used to probe the database for information; however, rather than looking through the content of the HTTP response received, the attacker calculates the Boolean manually

based on the time difference between request and response, and whether this number matches the request's *wait*.

Out-of-Band attacks are the last type of SQLi attack. These types of attacks can only be used if certain features are enabled in a database server such as the ability to trigger DNS or HTTP requests. It requires the attacker to receive a response from a database on a different communication channel other than the application or website itself. For example, occasionally there are functions built into a database that allow developers to gather specific information about their data by having the application send a DNS request to the domain itself. If an attacker knows the name of this function, they can gather the same information a developer would, by capturing that DNS request and exposing whatever sensitive information that function would return.

While there are now many ways to prevent these attacks, some of the methods to do so are relatively new and their creation can be attributed to attacks on various organizations in 2012. A hacker group called *GhostShell* was rumored to have been formed due to dissatisfaction with the status of online security measures instituted by large corporations. Many of *GhostShell*'s members had differing motivations, some political, some angry about unfair or oppressive institutions, but they all had one thing in common, and that was their desire to show-case the vulnerabilities in these large institutions to show the public that mega-corporations cannot be trusted with their data.

GhostShell's method of attack used SQLi to inject malicious code into hundreds of websites including government agencies, military organizations, and highly regarded financial institutions. They utilized a popular automated injection tool known as SQLmap to identify and efficiently exploit vulnerabilities in web applications. This open-source tool's wide availability

made it the perfect addition to both novice and advanced hackers' repertoires. SQL map allows hackers to obtain user and staff credentials, credit card numbers, and other personal identifying information. *GhostShell* claims to have hacked into the European Space Agency, the United States Department of Defense, and the Chicago Federal Reserve, in the process, acquiring over 1.6 million usernames and passwords. After the attack, customers and the general public started to lose faith in these companies and institutions. The attack was a rude awakening to many businesses, effectively showcasing just how insecurely their data was stored. Many companies started implementing more preventative measures in response in order to ensure an attack like this could not happen again.

While these attacks pose significant threats and are something every developer should be aware of, there are plenty of ways to minimize the risks. The first step is to make sure everyone associated with building an online application, (i.e. the developers, the QA staff, the DevOps team, as well as the system administrators), is adequately trained on expected security measures. The next step involves an overarching distrust of user input. Hackers will use input fields to launch attacks, thus, a developer may never assume the input will be free of malicious code. It is always important to use various degrees of input validation to ensure that injected statements are never used as prepared statements. Another great way to increase security with minimal effort is through the use of the latest versions of technologies, languages, and plugins. The latest versions of PHP prevent many forms of SQLi attacks without the developer using any input validation measures. The final act a developer can institute to protect user data involves scanning their application regularly for vulnerabilities. If done safely, a tool like SQL map can be perfect for testing against these attacks, as long as the developer is aware of how effective this tool can be at altering the data they are trying to protect.

```
1    $query = "SELECT * FROM LOGINS WHERE (USERNAME = ?) AND (PASSWORD = ?)";
2    $stmt = $pdo->prepare($query);
```

These two statements demonstrate a stored procedure (line 1) with a prepared statement via variable binding (line 2). PDO does not have prepared statements by default, it emulates them for MySQL interactions, building the query string internally instead, using the MySQL `real-escape-string` call to escape any special characters in the user's input text. This code to the right executes the prepared statement with the user's inputs and returns the result to the `$result` variable. If results are returned, it means that the user's credentials matched those in the database, and they have now logged in successfully.

```
3    $stmt->execute([$username, $password]);
4    $result = $stmt->fetchAll();
```

SQL injection attacks can be extremely powerful in collecting or altering user data. It has been used in many attacks over the years due to the ease of use, but as long as a developer is aware of the dangers, they can ensure their application includes up-to-date security measures to prevent these attacks. Scanning their application or website regularly, ensuring permissions are as strict as they can be, and ensuring input validation is always used in many forms (client and server-side), a developer can greatly decrease the chances that their application will be targeted.

Reference Page

1. Cheat Sheet Series - Query Parameterization :
https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html
2. Cheat Sheet Series - SQL Injection Prevention :
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
3. GitHub - PayloadsAllTheThings MySQL Injection :
[https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL Injection/MySQL Injection.md](https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/MySQL%20Injection.md)
4. Medium - Ghost Shell Attack Using SQL Injection : <https://medium.com/@20dcs137/ghost-shell-attack-using-sql-injection-d607456fd5b0>
5. Microsoft - SQL Injection : <https://learn.microsoft.com/en-us/sql/relational-databases/security/sql-injection?view=sql-server-ver16>
6. MySQLI stmt→get_result() : <https://www.php.net/manual/en/mysqli-stmt.get-result.php>
7. OWASP - SQL Injection : https://owasp.org/www-community/attacks/SQL_Injection
8. PHP Manual - PDO Prepared Statements : <https://www.php.net/manual/en/pdo.prepared-statements.php>
9. PHP Manual - SQL Injection : <https://www.php.net/manual/en/security.database.sql-injection.php>
10. PHP Manual - mysqli Prepared Statements :
<https://www.php.net/manual/en/mysqli.quickstart.statements.php>
11. PHP Manual - Stored Procedures : <https://www.php.net/manual/en/mysqli.quickstart.stored-procedures.php>
12. PortSwigger - SQL Injection : <https://portswigger.net/web-security/sql-injection>
13. SIEMxpert - SQL Injection Prevention and Mitigation : <https://www.siemxpert.com/blog/sql-injection-prevention-and-mitigation/>
14. W3Schools - PHP MySQL Prepared Statements :
https://www.w3schools.com/php/php_mysql_prepared_statements.asp
15. W3Schools - SQL Injection : https://www.w3schools.com/sql/sql_injection.asp
16. Invicti – Types of SQL Injections: <https://www.invicti.com/learn/sql-injection-sqli/>