A Contemporary Method to Identify and Avoid SQL Injection in Weather Forecasting App Using MYSQL

Galgotias College of Computer Science and Technology

Galgotias University , Greater Noida 203201

Author – Ashwin Perti (Assistant Professor)

Prashant Kumar – prashantt11891@gmail.com

Dharmendra bhadauria – dharmendrabhadauria77@gmail.com

Tej Pratap Singh – tejpratap.tech@gmail.com

1. Abstract

As in today's international most of the builders make mistakes all through their internet development segment.This leads the attacker to gain unauthorized get right of entry to of corporation's touchy records.The commonplace vulnerability through which an attacker profits access is SQLi.web utility is excellent target for attacker to advantage get entry to to touchy information.If builders fails to installation defensive measures for records safety the attacker gets the facts.Developer have to think past conventional security measures for entire information protection in these days's world.The SQLi vulnerability impacts any websites or net application.SQLi is one of the oldest and perilous vulnerability of web application. . financial institution and authorities groups must take specific and incredible steps to shield themselves towards SQLi vulnerability.we will use penetration checking out to locate SQLi vulnerability in net programs earlier than attack.

The motive of this paper is to make developers privy to SQLi assault vulnerability and apprehend how work attacks and further implement extra security concerning SQLi and shield their corporations from touchy records leakage.

Keywords: SQL injection, Database security, Web application, SQLite, SQLIA(SQL injection attack)Internet, security of data.

2. Introduction

With the fast development of net generation, internet packages are facing increasingly more security issues. consistent with the statistics released through the open internet software protection undertaking (OWASP), sql injection is a web security vulnerability that permits an attacker to interfere with the queries that an software makes to its database. It normally allows an attacker to view information that they may be not normally capable of retrieve. this might include facts belonging to other users, or some other statistics that the software itself is capable of get admission to. in lots of cases, an attacker can adjust or delete this statistics, inflicting chronic changes to the software's content or conduct. In some situations, an attacker can boost

an sq. injection assault to compromise the underlying server or other lower back-cease infrastructure, or perform a denial-of-provider attack.
The sq. injection vulnerability is the number one vulnerability to internet protection [I] . within the beyond few years, many specialists and scholars have completed a whole lot of research on sq. injection vulnerabilities, and proposed many injection techniques and defense technology, however maximum of these research are confined to first-order square injection. second-order square injection is evolved on the idea of the traditional first-orderSQL injection generation. it's miles greater tough to find out, but has the equal protection hazard to net packages because the firstorder square injection. At gift, there's little studies on 2nd-order square injection technology. YAN [2] analyzes the principle of T-order square injection, and proposes a technique of mixing dynamic detection with static detection to stumble on 2d-order square injection vulnerabilities. Tian Yujie [3] put forward a form of 2nd-order square injection attack defense model primarily based on stepped forward parameterization, consisting of enter clear out module, index substitute module, syntax assessment module and parametric replacement module. however, these strategies are not perfect for the detection of second-order SQLinjection vulnerabilities.The paper analyzeV the precept of 2nd-order square injection, and provides a technique of detecting 2d-order sq. injection attacks based totally on ISR guidance Set Randomization . The experimental consequences show that the proposed approach can discover 2nd-order sq. injection as it should be and efficiently.

3. THE PRINCIPLE OF SQL INJECTION

SQL Injection includes entering sq. code into web bureaucracy, eg. login fields, or into the browser cope with subject, to get entry to and manipulate the database in the back of the site, device or utility.

while you enter text in the Username and Password fields of a login display, the records you input is typically inserted into an square command. This command checks the information you've got entered towards the relevant desk inside the database.if your enter suits desk/row data, you're granted get entry to (within the case of a login display screen). If now not, you are knocked backout.

The Simple SQL Injection Hack In its simplest form, this is how the SQL Injection works. It's impossible to explain this without reverting to code for just a moment. Don't worry, it will all be over soon.Suppose we enter the following string in a Username field:
' OR 1=1
Web application authenticates users by verifying whether the returned result set is empty, and saves the result set in memory. Web application uses the result set in memory to construct SQL statements for modifying user's passwords.The authorization SQL query that is run by the server, the command which must be satisfied to allow access, will be something along the lines of:

```
SELECT * FROM users WHERE username
   = ?USRTEXT ' AND password = ?PASSTEXT?
```

...where USRTEXT and PASSTEXT are what the user enters in the login fields of the web form.

So entering `OR 1=1 -- as your username, could result in the following actually being run:
SELECT * FROM users WHERE username = ?' OR 1=1 -- 'AND password = '?

Two things you need to know about this: ['] closes the [username] text field.
'
duble-dash-txt.png
' is the SQL convention for Commenting code, and everything after Comment is ignored. So the actual routine now becomes:
SELECT * FROM users WHERE username = '' OR 1=1

1 is always equal to 1, last time I checked. So the authorization routine is now validated, and we are ushered in the front door to wreak havoc.But the process does serve to illustrate just what SQL Injection is all about -- injecting code to manipulate a routine via a form, or indeed via the URL. In terms of login bypass via Injection, the hoary old ' OR 1=1 is just one option.If a hacker

thinks a site is vulnerable, there are cheat-sheets all over the web for login strings which can gain access to weak systems. Here are a couple more common strings which are used to dupe SQL validation routines:

username field examples:

admin'-- ') or ('a'='a ") or ("a"="a hi" or "a"="a

... and so on.familiar with using styles, do not worry; the template arranges everything for you in a user-friendly way.

The first-order SQL injection loadV attack payload directly to the SQL query, but the attack process of second-order SQL injection is divided into 2 stages , namely the stage of storing attack payload to database or file system and the stage of building the SQL query statement with the attack payload from database or file system. For example, the
common function of user registration and user information modification in Web application can carry out an second-order SQL injection attack,the user registration function loads the attack payload to the database, and then the user information modification function extract the attack payload from database to construct a SQL database query, which will cause a second-order SQL injection .

The key codes of user registration function .THE PRINCIPLE OF SECOND - ORDER SQL INJECTION

```
$cmd=$1ink->prepare($query)
$cmd->bind_param("sss",$username,$password,$email)
$cmd->execute
```

Web application authenticates users by verifying whether the returned result set is empty, and saves the result set in memory. Web application uses the result set in memory to construct SQL statements for modifying user's passwords, the key code is shown below.

```
$result=mysql_query($query)
$row=mysql_fetch_array($result)
$username=$row[" usemame"]
$email=$row["email"]
```

```
$password=$_POST[“newpwd”]
$query=“UPDATE Users SET username=‘$username’
password=‘$password’,email=‘$email’
WHERE
username=‘$username’将 ”将
$rs=mysql_query($query)
```

The second-order SQL injection defense system is composed of a randomization module, attack detection module and de-randomization module. Randomization modules append a random integer to SQL keywords in Web application, which creates dynamically SQL keyword sets, the attacker don't know the new SQL keyword sets, so standard SQL which attacker injects can be detected, but at the same time, DBMS can’t recognize the new keywords, too, a solution to solve the problem is to modify the database interpreter, but this method is very complex

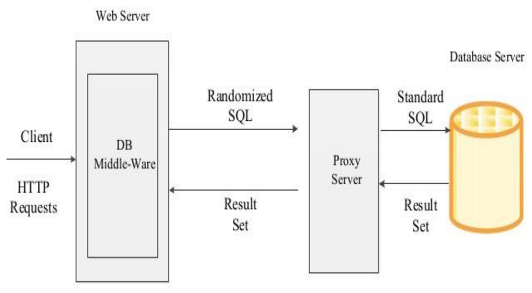
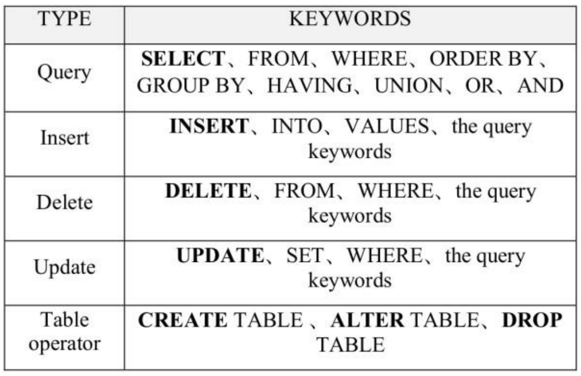


Fig. 1 SQL injection detect system

3.1. Keywords

The standard SQL statement can be described with regular expression, our method use regular expression to locate accurately the trusted constant strings containing SQL statements in the Web application, then randomize the keywords in an SQL statement with a random integer.

The keywords for each SQL statement is shown in Table 1,each type of SQL statement has a primary keyword which is formatted by bold font. Due to the WHERE expression in Insert, Delete, Update statement can contain subqueries, these types of SQL statements should involve the query keywords in addition to its own keywords.



TYPE	KEYWORDS
Query	**SELECT**、FROM、WHERE、ORDER BY、GROUP BY、HAVING、UNION、OR、AND
Insert	**INSERT**、INTO、VALUES、the query keywords
Delete	**DELETE**、FROM、WHERE、the query keywords
Update	**UPDATE**、SET、WHERE、the query keywords
Table operator	**CREATE** TABLE 、**ALTER** TABLE、**DROP** TABLE

TABLE 1 SQL KEYWORDS

3.2. Types OF Sql Injection

SQL injections typically fall under three categories: In-band SQLi (Classic), Inferential SQLi (Blind) and Out-of-band SQLi. You can classify SQL injections types based on the methods they use to access backend data and their damage potential.

3.2.1 Error Based SQL Injection:

The Error based technique, when an attacker tries to insert malicious query in input fields and get some error which is regarding SQL syntax or database.

For example, SQL syntax errors need to be like this:

you have an error to your sql syntax; take a look at the guide that corresponds in your MySQL server version for the right syntax to use near ''VALUE''.

The Error message gives records approximately the database used, where the syntax blunders befell in the query.

Error based totally technique is the easiest manner to find sql Injection.

3.2.2 Union-based Query:

This approach utilizes the UNION SQL operator to merge multiple select statements generated by the database into a single HTTP response. By analyzing this response, an attacker may discover exploitable data. If an error occurs, we attempt to exploit the SQL using a specially crafted SQL query that incorporates the "UNION" operator.

The UNION operator is commonly employed to combine two tables or execute two select queries simultaneously. It eliminates duplicate rows or columns when executing these queries together.

3.2.3 Blind SQLI

During a standard SQL injection (SQLI) attack, if an error occurs, the application typically displays an error message indicating an incorrect syntax in the SQL query. In contrast, Blind SQLI is a different technique that involves injecting SQL queries into the database blindly and inferring the output by observing changes in the response behavior.

Blind SQLI differs from error-based SQLI, where the attacker intentionally triggers specific error messages by inserting certain SQL queries into the database. In Blind SQLI, the attacker attempts to gather information by posing true or false queries to the database and determining the output based on the observed behavior. As a result, the attacker needs to predict and interpret the output accurately.

Blind SQL injections rely on analyzing server responses and behavioral patterns, which often results in slower execution compared to other injection techniques. However, they can be equally detrimental. Blind SQL injections can be categorized as follows:

- Boolean—In this technique, the attacker sends a SQL query to the database, causing the application to generate a response. The response will differ based on whether the injected query evaluates to true or false. By analyzing the resulting information within the

HTTP response, the attacker can deduce whether the injected message produced a true or false outcome."

- **Time-based**—In a time-based Blind SQL injection attack, the attacker sends a SQL query to the database that deliberately introduces a delay before the database responds.By observing the response time, the attacker can deduce whether the injected query evaluates to true or false. This determination influences the generation of an HTTP response, which may occur immediately or after the specified delay. Consequently, the attacker can infer the true or false outcome of the injected message without relying on data directly obtained from the database.

3.2.4 Out-of-band SQLi

Out-of-band SQL injection is a type of attack that relies on specific features being enabled on the targeted database server. It serves as an alternative to in-band and inferential SQL injection techniques.

Out-of-band SQL injection is employed when the attacker cannot utilize the same communication channel for both launching the attack and gathering information. This can occur in situations where the server's performance is slow or unstable. In such cases, these techniques leverage the server's capability to generate DNS or HTTP requests to transfer data to the attacker.

3.2.5 Time-based SQLI:

Time based SQl in which attackers insert SQL query causing database pause for a specified amount of time and then returning the results(just delaying the output). This is helpful when the attacker does not have any kind of answer (error/output) from the application because the input validation has been sanitized.

4. SQLI prevention and mitigation

The application uses an authentication query to check for the registered uses of the application. The authentication query matches the user entered credentials to the credentials stored in the database during user registration. In this implementation for each user authentication query to access the database, a unique fingerprint is generated using a hashing algorithm, based on the authentication credentials of the user, provided during the user registration. This unique fingerprint of the user is stored against his access credentials in the database.

The application utilizes an authentication query to verify the legitimacy of users. To prevent SQL injection attacks and mitigate their impact, there are various effective measures that can be implemented.

One fundamental step is input validation or sanitization, which involves developing code that can identify and reject illegitimate user inputs. While input validation is generally regarded as a best practice, it is important to note that it is not always foolproof. It can be challenging to define all possible legal and illegal inputs without causing a significant number of false positives. Striking the right balance is crucial to avoid negatively affecting user experience and the overall functionality of the application.

For this reason, a web application firewall (WAF) is commonly employed to filter out SQLI, as well as other online threats. To do so, a WAF typically relies on a large, and constantly updated, list of meticulously crafted signatures that allow it to surgically weed out malicious SQL queries. Usually, such a list holds signatures to address specific attack vectors and is regularly patched to introduce blocking rules for newly discovered vulnerabilities.

Modern web application firewalls are also often integrated with other security solutions. From these, a WAF can receive additional information that further augments its security capabilities.

For example, a web application firewall that encounters a suspicious, but not outright malicious input may cross-verify it with IP data before deciding to block the request. It only blocks the input if the IP itself has a bad reputational history.The authentication query matches the user entered credentials to the credentials stored in the database during user registration. In this implementation for each user authentication query to access the database, a unique fingerprint is generated using a hashing algorithm, based on the authentication credentials of the user, provided during the user registration. This unique fingerprint of the user is stored against his access credentials in the database.

4.1 Parameterized Statements

Programming languages talk to SQL databases using database drivers. A driver allows an application to construct and run SQL statements against a database, extracting and manipulating data as needed. Parameterized statements make sure that the parameters (i.e. inputs) passed into SQL statements are treated in a safe manner.

You should always use parameterized statements where available, they are your number one protection against SQL injection.

4.2 Object Relational Mapping

Many development teams find it beneficial to utilize Object Relational Mapping (ORM) frameworks to facilitate the translation of SQL result sets into code objects. By using ORM tools, developers can often avoid directly writing SQL statements in their code, as these tools employ parameterized statements behind the scenes.

One widely recognized ORM framework is Ruby on Rails' Active Record, which offers seamless data retrieval from the database. However, it is important to note that employing an ORM does not automatically guarantee immunity against SQL injection attacks. While ORM frameworks handle basic database operations safely, they may still provide mechanisms for constructing SQL statements or SQL fragments when dealing with more intricate database operations. Care must be taken to ensure that these custom SQL operations are properly parameterized to avoid SQL injection vulnerabilities.

4.3 Escaping Inputs

When parameterized statements or SQL-writing libraries are not available, an alternative approach is to properly escape special string characters in input parameters to prevent SQL injection attacks.

Injection attacks exploit the ability to manipulate inputs in a way that prematurely closes the argument string within the SQL statement. This is why attempted SQL injection attacks commonly include characters like ' or ". These characters are used by attackers to craft inputs that can disrupt the intended structure of the SQL statement.

To mitigate this risk, programming languages typically provide methods to handle strings containing quotes, including SQL. By properly escaping special string characters, such as doubling up the quote character (''), you can ensure that these characters are treated as part of the string itself rather than as the termination of the string.

By implementing proper escaping techniques, you can reduce the likelihood of SQL injection vulnerabilities. It is important to apply these measures when parameterized statements or SQL-writing libraries are not feasible options.

- Using proper character escaping is a common method to defend against SQL injection attacks and is supported by various programming languages through standard functions. However, it is important to be aware of the limitations and drawbacks associated with this approach:
- Careful character escaping: When relying on character escaping, it is crucial to ensure that all SQL statements throughout your codebase are properly escaped. Failing to escape characters consistently can introduce vulnerabilities.

Not limited to quote characters: It's essential to understand that SQL injection attacks can exploit other techniques beyond the abuse of quote characters. While escaping symbols is a helpful practice, it may not provide comprehensive protection against all types of injection attacks.

For instance, even if a SQL statement expects a numeric ID where quote characters are not required, the code can still be vulnerable to injection attacks. Simply manipulating the input in unexpected ways can lead to security vulnerabilities, regardless of the presence or absence of quote characters.

To address SQL injection risks effectively, it is recommended to utilize techniques such as parameterized statements or SQL-writing libraries that handle input validation and parameter binding. These methods provide more robust protection against a broader range of injection attack vectors.

4.4 Sanitizing Inputs

Sanitizing inputs is a good practice for all applications. In our example hack, the user supplied a password as ' or 1=1--, which looks pretty suspicious as a password choice.

Developers should always make an effort to reject inputs that look suspicious out of hand, while taking care not to accidentally punish legitimate users. For instance, your application may clean parameters supplied in GET and POST requests in the following ways:

- Check that supplied fields like email addresses match a regular expression.
- Ensure that numeric or alphanumeric fields do not contain symbol characters.
- Reject (or strip) out whitespace and new line characters where they are not appropriate.

Client-side validation (i.e. in JavaScript) is useful for giving the user immediate feedback when filling out a form, but is no defense against a serious hacker. Most hack attempts are performed using scripts, rather than the browser itself.

Language specific recommendations for Prepared Statement:

1. Java EE—	use Prepared Statement() with bind variables
2. .NET —	use parameterized querielike SqlCommand() or OleDbCommand() with bind variables
3. PHP —	use PDO with strongly typed parameterized queries (using bindParam())
4. Hibernate —	use createQuery()with bind variables (called named
	parameters in Hibernate)
5. SQLite —	use sqlite3_prepare()to create a statement object

5. CONCLUSION AND FUTURE WORK

This report discussed web application security principles and fundamental information that can help us to prevent web exploits in our system. Web applications are considered.the most exposed and least protected, thereafter vulnerable because the standards.somehow are not focused on security but more on the server's need for functionality.Security threats are more common than before because the internet has become today's economy the most valuable tool for everyone. So there is indeed a need to protect our resources, data and user privacy information. As technology moves forward and brings new strategies, tools, models and methods to increase security levels, hackers will be part of this never end game.

The proposed system is developed to detect the vulnerabilities like SQLi, Cross Site Scripting, Local and Remote File Inclusion, Command Injection in web applications and it will also provide information about remediation of vulnerable URL and its vulnerability. Our formalization goes at the heart of the problem and captures seemingly different types of above mentioned vulnerabilities. The simple and effective strategy is meant to be cost effective and is openly targeted toward large commercial applications. The results of the proposed systems are satisfactory.

References

[1] https://cheatsheetseries.owasp.org/cheatsheets/ SQL_Injection_Prevention_Cheat_Sheet.html

[2] YAN L.LI X H˙FENG R T˙et al˴Detection method of the second-order SQL injection in Web applications[J]˴ Lecture Notes inComputer Science˙2014˙8332∶154-165˴.

[3] TIAN Y J,ZHAO Z M ˙ ZHANG H C ˙ et al ˴Second-order SOL injection attack defense model[J] ˴ Netinfo Security,2014,(11)70-73˴

[4] Sqli-labs Project[EB/OL].https://github.com/Audi-1/sqli-labs.

[5] Protecting Against SQL Injection https://www.hacksplaining.com/prevention/sql-injection