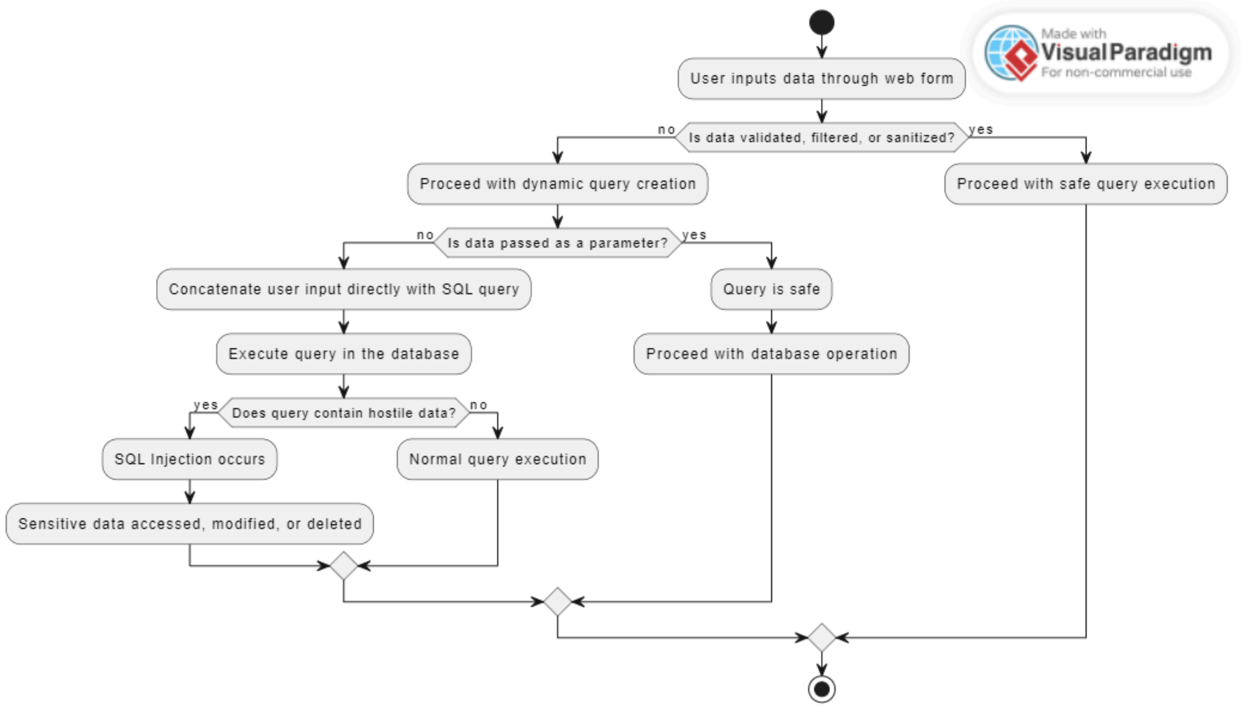
The Open Web Application Security Project (OWASP) is a project that identifies several common coding weaknesses that can lead to vulnerabilities in web software applications. According to the latest ranking from OWASP Top 10 these weaknesses includes broken access control, cryptographic failures, injection, insecure design, etc. (OWASP, 2021). Each of these weaknesses represents a potential risk that can be exploited by malicious actors, resulting in data breaches, unauthorized access, and other security incidents. In this discussion, I will focus on Injection particularly SQL Injection.

Understanding SQL Injection

Injection vulnerabilities, including SQL Injection, are a critical concern in web application security. According to OWASP, Injection vulnerabilities occur when untrusted data is sent to an interpreter as part of a command or query. This allows attackers to manipulate the interpreter into executing unintended commands or accessing unauthorized data. SQL Injection specifically targets databases by injecting malicious SQL code into queries and occurs when an application fails to properly sanitize user inputs, allowing attackers to alter the SQL queries executed by the database.

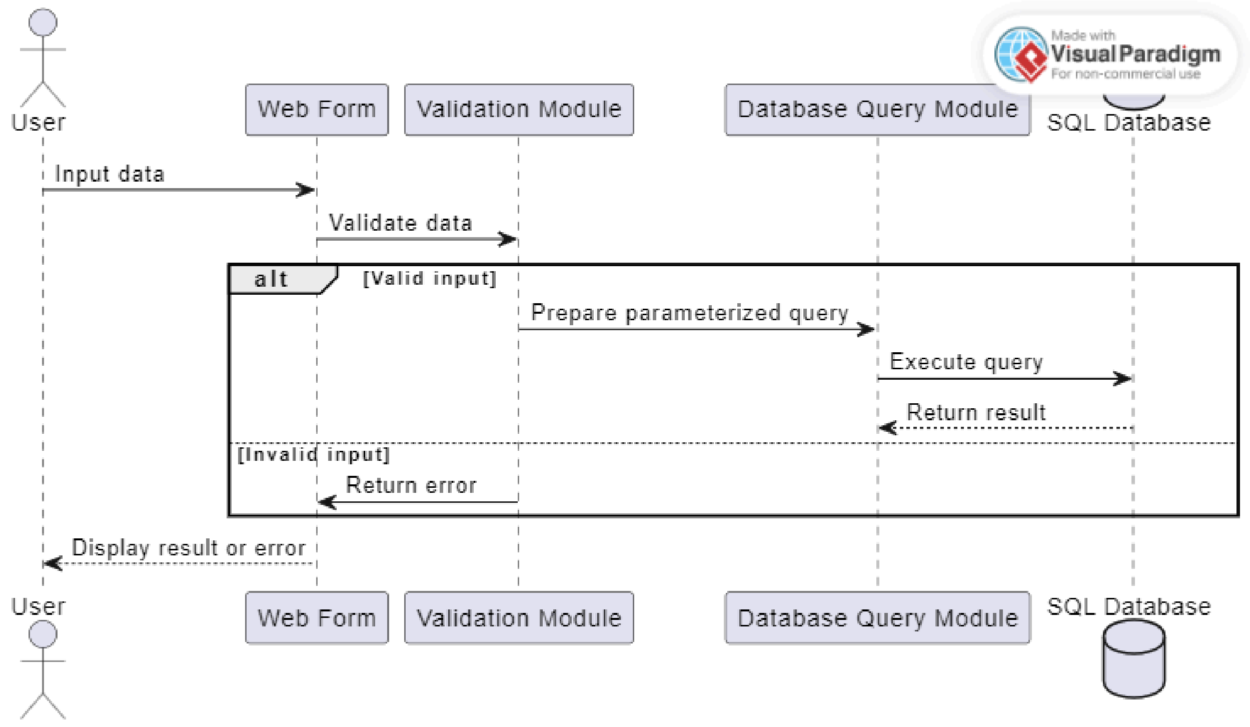
Flowchart of Steps Leading to SQL Injection:


Made with Visual Paradigm For non-commercial use
User inputs data through web form
Is data validated, filtered, or sanitized?
no
yes
Proceed with dynamic query creation
Proceed with safe query execution
Is data passed as a parameter?
no
yes
Concatenate user input directly with SQL query
Query is safe
Execute query in the database
Proceed with database operation
Does query contain hostile data?
yes
no
SQL Injection occurs
Normal query execution
Sensitive data accessed, modified, or deleted

Preventing SQL Injection

One important step to preventing SQL injection is ensuring that user input is never directly used in queries without proper validation and sanitization (Galluccio, et al, 2020). The use of parameterized queries or prepared statements is essential, as they keep data separate from commands. Additionally, positive server-side input validation and the use of secure APIs can help prevent such vulnerabilities.

In designing a software as this, many UML models can be used, but, when it comes to the case of SQL injection in particular, sequence diagram seems to be the best to use. This is because sequence diagram illustrates the sequence of operations and how objects interact in a particular sequence. It is useful for mapping out the flow of data from user input to database query execution, highlighting where vulnerabilities might eventually occur. Do find below the sequence diagram showing the user interaction with the software and also highlighting where SQL injection vulnerabilities might occur if validation is skipped:



Conclusion

Injection vulnerabilities, specifically, SQL Injection vulnerabilities are still popular in web software applications. Mitigating these risks requires a combination of both secure coding practices, proper input validation and the use of safe database interaction methods. Using UML models, such as Sequence Diagrams, to demonstrate the flow of data will help developers to easily identify where things can go wrong and thereby be able to better design secure applications.

REFERENCES:

OWASP Foundation (2021) *OWASP Top 10*. Available from: https://owasp.org/www-project-top-ten/ [Accessed 11 August 2024].

Galluccio, E., Caselli, E. and Lombari, G., 2020. SQL Injection Strategies: Practical techniques to secure old vulnerabilities against modern attacks. Packt Publishing Ltd.