Initial Post

One of the most significant OWASP coding weaknesses is **A03: Injection**, with SQL injection remaining a prevalent cause of data breaches. Injection vulnerabilities typically arise through a sequence of development decisions in which untrusted user input is treated as executable instructions rather than data (OWASP, 2021). As emphasised in security-driven software development, such weaknesses are rarely isolated coding errors but instead reflect shortcomings in design, process, and governance (Olmsted, 2024).

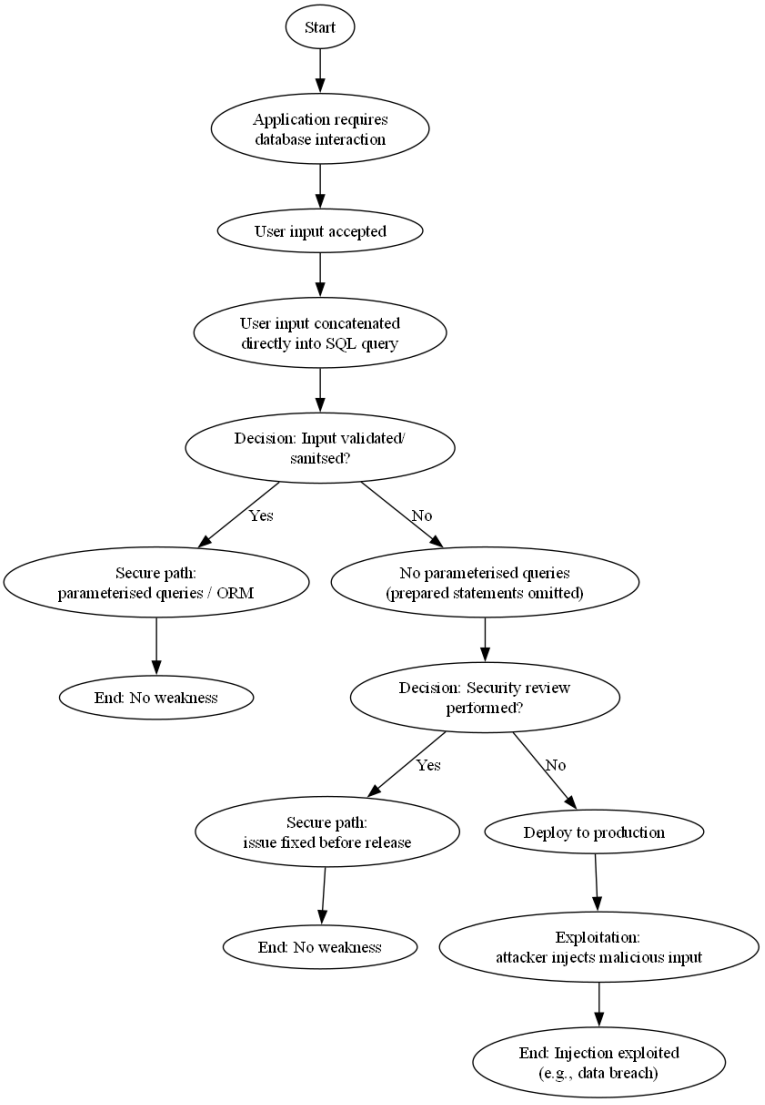

Figure 1 presents a **UML activity diagram** created using Python to illustrate the steps that may lead to an SQL injection vulnerability. The flow begins when an application requires database interaction and accepts user input. The vulnerability is introduced when user input is concatenated directly into an SQL query. A critical decision point follows: whether input validation and sanitisation are applied. If secure practices such as parameterised queries or ORM mechanisms are used, the flow terminates without a weakness. If not, the process continues toward deployment. A second decision point addresses whether a security-focused code review is performed. Where such review is absent, the

application is deployed with the vulnerability intact, enabling an attacker to inject malicious input and exploit the flaw. This flowchart demonstrates how SQL injection emerges from cumulative development and review failures rather than a single mistake.

To present the design of proposed software that mitigates this weakness, several UML models are appropriate. A **class diagram** is well suited to demonstrate secure separation of responsibilities, such as isolating database access logic from input handling, in line with established architectural principles (Naghdipour, Hasheminejad and Barmaki, 2023). A **sequence diagram** is particularly effective for illustrating runtime behaviour, showing that validated input is passed to the database exclusively via parameterised queries. A **use case diagram** may additionally be used to capture security-critical functionality at the requirements level. Together, these UML models provide a coherent representation of a design that directly addresses the causes illustrated in the activity diagram.

References

Naghdipour, A., Hasheminejad, S.M.H. and Barmaki, R.L. (2023) 'Software design pattern selection approaches: A systematic literature review', *Software: Practice and Experience*, 53(4), pp. 935–958.

Olmsted, A. (2024) *Security-Driven Software Development: Learn to Analyze and Mitigate Risks in Your Software Projects*. Birmingham: Packt Publishing.

OWASP (2021) *OWASP Top 10: A03 – Injection*. Open Web Application Security Project.