Unit 1

# Reflection

## OWASP

This unit introduced the idea of what secure software development is all about as well as a number of standards used by the industry to develop secure software. Of importance, was the introduction of OWASP (Open Web Application Security Project) top ten web application security risks (OWASP, 2021). Referencing OWASP frequently helps development teams stay focused on addressing the most common vulnerabilities present in web applications, for example Cross-site Scripting (also known as XSS). Not only does OWASP highlight *what* a particular vulnerability entails, but they also provide various recommendations (OWASP_1, 2021) on how to address the particular vulnerability.

I found particularly interesting the references to the Common Weaknesses Enumeration (CWE) database (Mitre, 2021) as they categorise vulnerabilities by topics such as software development, hard or research. In addition, the CWE database also makes reference to OWASP top ten as well (Mitre_1, 2021), lending itself as a rather detailed companion to the standard OWASP definitions.

## SDLC and Security

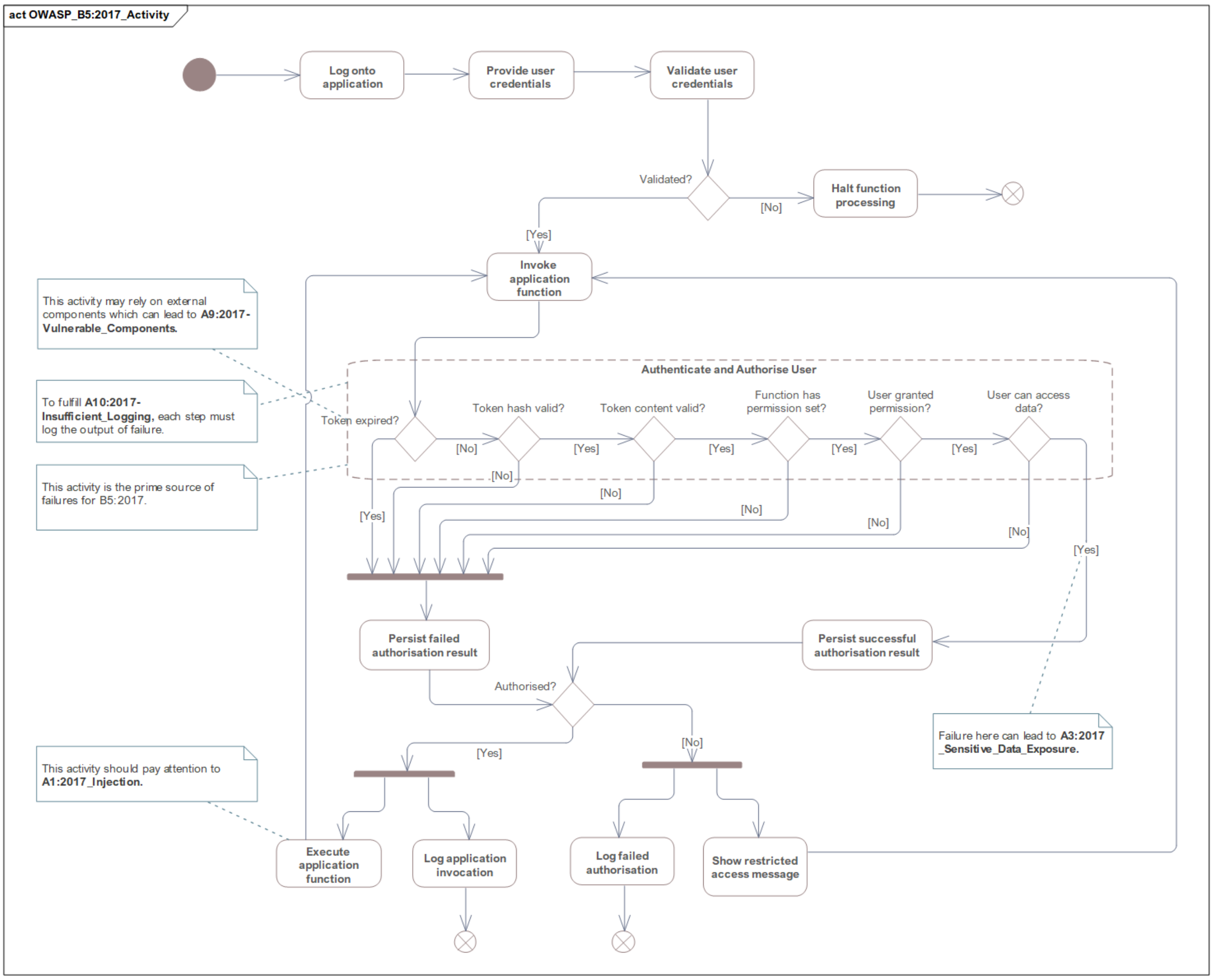
Merely knowing of vulnerabilities is insufficient to develop secure software because it is necessary to incorporate security into the software development lifecycle (SDLC). This unit highlighted the challenges of integrating security into an SDLC (especially Agile) because Agile does not necessarily consider security in each “phase”. This is because Agile quickly delivers on requirements using a flexible development methodology. One resolution can be found in the work expressed by Moyón et al. (2020) who propose a framework of continuous security compliance where security compliance is satified on a continual basis as opposed to compliance at release time. In addition, I found the tabulated data in the Secure Software Development Framework (NIST, 2020) to useful in understanding the various concerns to consider in developing secure systems.

## UML and Secure Software

Lastly, we looked at how UML can aid secure software development. Despite that UML has broad applicability through use case diagrams, class diagrams or sequence diagrams, its usefulness to aid secure software development is found in the *detail* they contain. Alshayeb et al. (2020) consider how to improve the security in UML sequence diagrams by leveraging model refactoring to eliminate security issues via a generic algorithm and associated correction techniques. They do this by classifying issues according to abtraction, encaptulation, moularisation and hierarchy concerns. Whereas Gosh (ND) proposes a method to uncover security policy violations in a UML model  using a scenario-based analysis technique that uses OCL to specification to determine whether or not a given scenario is permitted in a UML model.

## UML FlowChart

As an example of leveraging UML, students we tasked to identify an OWASP vulnerability and depict how it could occur using UML. I chose Broken Authentication and depicted it as follows:


act OWASP_B5:2017_Activity
Log onto application
Provide user credentials
Validate user credentials
Validated?
[No]
Halt function processing
[Yes]
Invoke application function
Authenticate and Authorise User
Token expired?
Token hash valid?
Token content valid?
Function has permission set?
User granted permission?
User can access data?
[No]
[Yes]
[Yes]
[Yes]
[Yes]
[No]
[No]
[No]
[No]
[No]
[Yes]
[Yes]
Persist failed authorisation result
Persist successful authorisation result
Authorised?
[Yes]
[No]
Execute application function
Log application invocation
Log failed authorisation
Show restricted access message

This activity may rely on external components which can lead to **A9:2017-Vulnerable_Components.**

To fulfill **A10:2017-Insufficient_Logging**, each step must log the output of failure.

This activity is the prime source of failures for B5:2017.

This activity should pay attention to **A1:2017_Injection.**

Failure here can lead to **A3:2017_Sensitive_Data_Exposure.**

For this flowchart, I homed in on the *details* of what a generic flow would like like and *why* it would be possible for a flaw to occur. In the scenario depicted, the flaw exists because each decision node could potentially fail to correctly determine a “Yes” or “No” state.

# References

Alshayeb, M., Mumtaz, H., Mahmood, S. &  Niazi, M. (2020). Improving the security of UML sequence diagram using genetic algorithm. IEEE Access, 8: 62738-62761.

Ghosh, S. (ND) A rigorous approach to uncovering security policy violations in uml designs.

Mitre (2021). Common Weaknesses Enumeration. Available from https://cwe.mitre.org/data/ [Accessed 16 Aug. 2021]

Mitre_1 (2021). Common Weaknesses Enumeration. Available from https://cwe.mitre.org/data/definitions/1026.html [Accessed 31 Aug. 2021]

Moyón, F., Méndez, D., Beckers, K. & Klepper, S. (2020) How to Integrate Security Compliance Requirements with Agile Software Engineering at Scale? Lecture notes in Computer Science: 69–87.

OWASP (2021) OWASP Top Ten. Available from https://owasp.org/www-project-top-ten/ [Accessed 31 Aug. 2021]

OWASP_1 (2021) OWASP: A1:2017-Injection. Available from https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.