

# Play to Secure: Gamified Platform for Secure Web Development

by

Inan Karim Chowdhury  
22101377

Mimia Mozammel Mimu  
22101088

Sal-Sabil  
22101376

Ayaz Elahi Mullick  
22101069

Faiyaz Ahmed Chowdhury  
22101848

A thesis submitted to the Department of Computer Science and Engineering  
in partial fulfillment of the requirements for the degree of  
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering  
BRAC University  
January 2026.

© 2026. BRAC University  
All rights reserved.

# Declaration

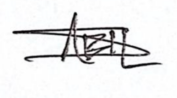
It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

**Student's Full Name & Signature:**



**Inan Karim Chowdhury**  
22101377



**Sal Sabil**  
22101376



**Mimia Mozammel Mimu**  
22101088



**Ayaz Elahi Mullick**  
22101069



**Faiyaz Ahmed Chowdhury**  
22101848

# Approval

The thesis titled “Play to Secure: Gamified Platform for Secure Web Development” submitted by

1. Inan Karim Chowdhury (22101377)
2. Mimia Mozammel Mimu (22101088)
3. Faiyaz Ahmed Chowdhury (22101848)
4. Ayaz Elahi Mullick (22101069)
5. Sal-Sabil (22101376)

of [Spring], [2022] has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science in [Summer] 2025.

## Examining Committee:

Supervisor:  
(Member)



---

Dr. Farida Chowdhury

Professor  
Computer Science And Engineering  
Brac University

Thesis Coordinator:  
(Member)

---

Dr. Md. Golam Rabiul Alam

Professor  
Computer Science And Engineering  
Brac University

Head of Department:  
(Chair)

---

Dr. Sadia Hamid Kazi

Associate Professor & Chairperson  
Department of Computer Science and Engineering  
Brac University



# Abstract

Novice developers often focus on quick progress and fast learning at the intermediary stage of web development, commonly disregarding the security aspects that need to be considered. Many are not aware of key concepts such as input validation, parameterized queries, and access control, which have adverse consequences such as leakage of confidential information and loss of reputation. To explore this further, we conducted 32 semi-structured interviews of Bangladeshi developers at novice, mid, and senior levels working across both large and small enterprises. The interviews were then transcribed, coded, and analysed using thematic analysis. Our interview findings include several themes: developers frequently make tradeoffs between functionality and security, code reviews are vital in ensuring safe coding practices, and, while in the learning process, security concepts need to be taught in an effective and motivating manner. We have developed and designed a narrative-based gamified learning platform using the Octalysis framework to improve learning motivation. The platform promotes security awareness and embeds security as a fundamental competency alongside development skills. The learning content is structured around seven common web security attacks, presented as progressive challenges within the game narrative. The gamified solution was evaluated through the application of the MEEGA+ model with a sample size of 30 participants. The IRT analysis classified the gamified platform as GOOD QUALITY with a mean  $\theta$  score of 60.11 ( $42.5 \leq \theta < 65$ ). This shows that the gamified solution successfully meets the requirement for an effective and motivating security education for novice developers.

**Keywords:** XSS, SQLi, Dom Clobbering, IDOR, NoSQLi, Clickjacking, Insufficient Authentication, Gamification, Octalysis, MEEGA+

# Table of Contents

|                                                                                           |            |
|-------------------------------------------------------------------------------------------|------------|
| <b>Declaration</b>                                                                        | <b>i</b>   |
| <b>Approval</b>                                                                           | <b>ii</b>  |
| <b>Abstract</b>                                                                           | <b>iv</b>  |
| <b>Table of Contents</b>                                                                  | <b>v</b>   |
| <b>Table of Contents</b>                                                                  | <b>v</b>   |
| <b>List of Figures</b>                                                                    | <b>x</b>   |
| <b>List of Tables</b>                                                                     | <b>xii</b> |
| <b>Nomenclature</b>                                                                       | <b>xii</b> |
| <b>1 Introduction</b>                                                                     | <b>1</b>   |
| 1.1 Background . . . . .                                                                  | 1          |
| 1.1.1 Importance of Secure Web Applications and the Threat Land-<br>scape . . . . .       | 2          |
| 1.1.2 Insecure Development Practices by Developers and their Knowl-<br>edge Gap . . . . . | 2          |
| 1.1.3 Gamification as a Tool for Educational Purposes . . . . .                           | 3          |
| 1.1.4 Gaps in Traditional Security Training and Real-World Conse-<br>quences . . . . .    | 4          |
| 1.2 Rational of the Study or Motivation . . . . .                                         | 5          |
| 1.3 Problem Statement . . . . .                                                           | 6          |
| 1.4 Research Objectives . . . . .                                                         | 7          |
| 1.5 Scopes and Challenges . . . . .                                                       | 8          |
| 1.5.1 Scopes . . . . .                                                                    | 8          |
| 1.5.2 Challenges . . . . .                                                                | 9          |
| 1.6 Our Contribution . . . . .                                                            | 10         |
| 1.7 Thesis Structure . . . . .                                                            | 11         |

|          |                                                                        |           |
|----------|------------------------------------------------------------------------|-----------|
| <b>2</b> | <b>Literature Review</b>                                               | <b>13</b> |
| 2.1      | Preliminaries . . . . .                                                | 13        |
| 2.1.1    | Secure Coding Concepts and Importance . . . . .                        | 13        |
| 2.1.2    | Common Web Vulnerabilities . . . . .                                   | 15        |
| 2.1.3    | Gamification and its use in Education . . . . .                        | 17        |
| 2.1.4    | Gamification Frameworks . . . . .                                      | 18        |
| 2.2      | Review of Existing Research . . . . .                                  | 20        |
| 2.2.1    | Security Awareness of Developers and the Impact of Tools . .           | 21        |
| 2.2.2    | Attacks in Focus . . . . .                                             | 22        |
| 2.2.3    | Gamification System and Frameworks . . . . .                           | 28        |
| 2.3      | Summary of Key Findings . . . . .                                      | 30        |
| <b>3</b> | <b>Methodology</b>                                                     | <b>32</b> |
| 3.1      | Phase 1: Vulnerabilities Identification and Gamification Exploration . | 33        |
| 3.1.1    | Identification of Vulnerabilities . . . . .                            | 33        |
| 3.1.2    | Gamification Exploration . . . . .                                     | 33        |
| 3.2      | Phase 2: Gamification Design and User Study . . . . .                  | 34        |
| 3.2.1    | Interview for Validation and Additional Insight . . . . .              | 34        |
| 3.2.2    | Framework-Based Game Design . . . . .                                  | 34        |
| 3.3      | Phase 3: Implementation and Evaluation of the Gamification . . . . .   | 34        |
| 3.3.1    | Creation of the Gamified Platform . . . . .                            | 34        |
| 3.3.2    | Evaluation Criteria . . . . .                                          | 35        |
| <b>4</b> | <b>Semi-Structure Interview</b>                                        | <b>36</b> |
| 4.1      | Interview Approach and Participants' Engagement . . . . .              | 36        |
| 4.1.1    | Overview of Participants and Study Approach . . . . .                  | 36        |
| 4.1.2    | Participant Recruitment . . . . .                                      | 39        |
| 4.1.3    | Pilot Testing . . . . .                                                | 40        |
| 4.1.4    | Interview Protocol and Initialization . . . . .                        | 41        |
| 4.2      | Procedure Analysis . . . . .                                           | 42        |
| 4.3      | Ethical Considerations . . . . .                                       | 43        |
| 4.4      | Study Limitations . . . . .                                            | 44        |
| <b>5</b> | <b>Interview Findings</b>                                              | <b>45</b> |
| 5.1      | Analysis of Novice Developer Responses . . . . .                       | 45        |
| 5.1.1    | Security Awareness and Knowledge Gaps . . . . .                        | 47        |
| 5.1.2    | Learning Sources and Framework Dependency . . . . .                    | 47        |
| 5.1.3    | Security Practices and Real-World Experience . . . . .                 | 48        |
| 5.1.4    | Attitudes Toward Security and Learning . . . . .                       | 49        |
| 5.2      | Analysis of Mid-Level Developer Responses . . . . .                    | 50        |
| 5.2.1    | Project Experience and Security Awareness . . . . .                    | 52        |

|          |                                                              |           |
|----------|--------------------------------------------------------------|-----------|
| 5.2.2    | Secure Development Practice . . . . .                        | 53        |
| 5.2.3    | Responsibility Sharing . . . . .                             | 53        |
| 5.2.4    | Credential Management and Security Practices . . . . .       | 54        |
| 5.2.5    | Vulnerability Awareness and Security Knowledge . . . . .     | 55        |
| 5.2.6    | Gamification and Security Training . . . . .                 | 57        |
| 5.3      | Analysis of Senior Developer Responses . . . . .             | 58        |
| 5.3.1    | Security Practices and Mentorship . . . . .                  | 62        |
| 5.3.2    | Security-Specific Topics . . . . .                           | 64        |
| 5.3.3    | Gamification and Motivation in Security Training . . . . .   | 66        |
| 5.4      | Discussion: The Developer Security Lifecycle . . . . .       | 67        |
| 5.4.1    | From Novice to Senior: The Professional Journey . . . . .    | 67        |
| 5.4.2    | Security Awareness Across Experience Levels . . . . .        | 68        |
| 5.4.3    | Gamification as a Cross-Level Learning Approach . . . . .    | 69        |
| <b>6</b> | <b>Play to Secure Design and Development Process</b>         | <b>71</b> |
| 6.1      | Introduction . . . . .                                       | 71        |
| 6.2      | Design Objectives and Learning Goals . . . . .               | 72        |
| 6.3      | Game Conceptualization . . . . .                             | 72        |
| 6.3.1    | Our Gamification System through Octalysis Process . . . . .  | 73        |
| 6.4      | Game Design and Development Process . . . . .                | 76        |
| 6.4.1    | Game Design Objectives and Learning Principles . . . . .     | 76        |
| 6.4.2    | Hybrid Approach for Developing the Gamified System . . . . . | 78        |
| 6.4.3    | Development Workflow . . . . .                               | 79        |
| <b>7</b> | <b>Design Implications and Game Functionality</b>            | <b>81</b> |
| 7.1      | Design Implications . . . . .                                | 81        |
| 7.2      | How the Game Works . . . . .                                 | 82        |
| 7.2.1    | Admin Panel Design Implications . . . . .                    | 86        |
| 7.2.2    | Mystery Gifts and Reward Design . . . . .                    | 86        |
| 7.2.3    | Design Details per Vulnerability . . . . .                   | 86        |
| 7.2.4    | AI-Based Automated Secure Code Assessment . . . . .          | 93        |
| <b>8</b> | <b>Game Evaluation Methodology</b>                           | <b>96</b> |
| 8.1      | The MEEGA+ Evaluation Framework . . . . .                    | 96        |
| 8.1.1    | MEEGA+ . . . . .                                             | 96        |
| 8.1.2    | MEEGA+ Evaluation Approach . . . . .                         | 97        |
| 8.1.3    | MEEGA+ Application in This Study . . . . .                   | 98        |
| 8.2      | Alignment with Research Objectives . . . . .                 | 101       |
| 8.3      | Ethical Considerations . . . . .                             | 101       |
| 8.4      | Evaluation Results . . . . .                                 | 102       |
| 8.5      | Evaluation Methodology . . . . .                             | 102       |

|           |                                                                                                                  |            |
|-----------|------------------------------------------------------------------------------------------------------------------|------------|
| 8.5.1     | Participant Selection and Demographics . . . . .                                                                 | 102        |
| 8.5.2     | Evaluation Procedure . . . . .                                                                                   | 103        |
| 8.5.3     | Measurement Instrument . . . . .                                                                                 | 104        |
| 8.5.4     | Data Analysis . . . . .                                                                                          | 104        |
| 8.6       | Quantitative Results . . . . .                                                                                   | 104        |
| 8.6.1     | Overall Performance . . . . .                                                                                    | 104        |
| 8.6.2     | Usability Dimensions Analysis . . . . .                                                                          | 106        |
| 8.6.3     | Analysis of Dimensions of Player Experience . . . . .                                                            | 107        |
| 8.7       | IRT Quality Classification . . . . .                                                                             | 110        |
| 8.8       | Reliability Analysis . . . . .                                                                                   | 111        |
| 8.9       | Patterns of Engagement and Completion Measures . . . . .                                                         | 113        |
| 8.10      | Key Findings Summary . . . . .                                                                                   | 113        |
| <b>9</b>  | <b>Discussion</b>                                                                                                | <b>116</b> |
| 9.1       | Research Objectives Fulfillment . . . . .                                                                        | 116        |
| 9.1.1     | RO1: Researching the Most Frequent Web Vulnerabilities . . .                                                     | 117        |
| 9.1.2     | RO2: Determining Factors behind lack of security awareness<br>and insecure coding by Literature Review . . . . . | 118        |
| 9.1.3     | RO3: Findings on Factorials of Insecure Coding by Interview-<br>ing Developers . . . . .                         | 118        |
| 9.1.4     | RO4: Design and Development of a Game-Based Learning<br>Platform . . . . .                                       | 119        |
| 9.1.5     | RO5: Findings the Effective Game Evaluation Methods . . . .                                                      | 119        |
| 9.1.6     | RO6: Measuring the Proposed Game Effectiveness . . . . .                                                         | 120        |
| 9.2       | Comparison with Existing Works . . . . .                                                                         | 120        |
| <b>10</b> | <b>Conclusion and Future Work</b>                                                                                | <b>127</b> |
| 10.1      | Limitations . . . . .                                                                                            | 128        |
| 10.2      | Future Research Directions . . . . .                                                                             | 129        |
| 10.3      | Conclusion . . . . .                                                                                             | 129        |
|           | <b>References</b>                                                                                                | <b>138</b> |
|           | <b>Appendix</b>                                                                                                  | <b>138</b> |
| <b>A</b>  | <b>Gamified System Request</b>                                                                                   | <b>139</b> |
| A.1       | System Request . . . . .                                                                                         | 139        |
| A.1.1     | Business need . . . . .                                                                                          | 139        |
| A.1.2     | Business requirements . . . . .                                                                                  | 139        |
| A.1.3     | Business value . . . . .                                                                                         | 140        |
| A.1.4     | Special issues or constraints . . . . .                                                                          | 140        |

|       |                                                         |     |
|-------|---------------------------------------------------------|-----|
| A.2   | Functional Requirements . . . . .                       | 140 |
| A.2.1 | Module 1: User Management and Social Platform . . . . . | 140 |
| A.2.2 | Module 2: Learning Challenges and Feedback . . . . .    | 141 |
| A.2.3 | Module 3: Social Features and Leaderboard . . . . .     | 141 |
| A.2.4 | Module 4: Advanced Learning and Assessment . . . . .    | 141 |
| A.3   | Technology . . . . .                                    | 142 |
| A.3.1 | Framework and Languages . . . . .                       | 142 |

# List of Figures

|      |                                                         |    |
|------|---------------------------------------------------------|----|
| 1.1  | Gamification - Educational Tool . . . . .               | 4  |
| 2.1  | Gamification - Features . . . . .                       | 18 |
| 2.2  | MDA (Mechanics-Dynamics-Aesthetics) Framework . . . . . | 19 |
| 2.3  | MDE (Mechanics-Dynamics-Emotions) Framework . . . . .   | 19 |
| 2.4  | Octalysis Framework . . . . .                           | 20 |
| 3.1  | Detailed Methodology . . . . .                          | 32 |
| 6.1  | Octalysis Framework Implementation . . . . .            | 73 |
| 6.2  | Development Workflow . . . . .                          | 79 |
| 7.1  | Game Dashboard . . . . .                                | 82 |
| 7.2  | Game Path . . . . .                                     | 83 |
| 7.3  | The Social Wall . . . . .                               | 83 |
| 7.4  | Admin Dashboard . . . . .                               | 84 |
| 7.5  | Game Leaderboard . . . . .                              | 85 |
| 7.6  | Community Leaderboard . . . . .                         | 85 |
| 7.7  | SQLi Scenario . . . . .                                 | 87 |
| 7.8  | IDOR Scenario . . . . .                                 | 88 |
| 7.9  | XSS Scenario . . . . .                                  | 89 |
| 7.10 | NoSQLi Scenario . . . . .                               | 90 |

|                                               |     |
|-----------------------------------------------|-----|
| 7.11 Clickjacking Scenario . . . . .          | 91  |
| 7.12 Dom Clobbering Scenario . . . . .        | 92  |
| 7.13 Broken Authentication Scenario . . . . . | 93  |
| 7.14 Level 3 Scenario . . . . .               | 94  |
| 8.1 MEEGA+ Application Workflow . . . . .     | 98  |
| 9.1 Research Objective : Chart . . . . .      | 117 |



# List of Tables

|     |                                                                 |     |
|-----|-----------------------------------------------------------------|-----|
| 4.1 | Demographic Information of Novice Developers . . . . .          | 37  |
| 4.2 | Demographic Information of Mid-level Developers . . . . .       | 38  |
| 4.3 | Demographic Information of Senior Developers . . . . .          | 38  |
| 5.1 | Thematic Analysis Results on Novice Developers . . . . .        | 45  |
| 5.2 | Thematic Analysis Results on Mid-level Developers . . . . .     | 50  |
| 5.3 | Thematic Analysis Results on Senior Developers . . . . .        | 59  |
| 8.1 | Gender Distribution of Participants . . . . .                   | 102 |
| 8.2 | Badge Earning Statistics . . . . .                              | 103 |
| 8.3 | MEEGA+ Evaluation Results Summary . . . . .                     | 105 |
| 8.4 | IRT Quality Classification Results . . . . .                    | 110 |
| 8.5 | Individual Quality Level Distribution . . . . .                 | 110 |
| 8.6 | Reliability Analysis Results . . . . .                          | 112 |
| 9.1 | Papers Included in Literature Comparison . . . . .              | 121 |
| 9.2 | Comparison of Our Proposed System with Existing Works . . . . . | 122 |

# Chapter 1

## Introduction

### 1.1 Background

In today's digital age, the use of web applications in the fields of communications, commerce, education, and governance has become a fundamental phenomenon, making them the most targeted systems for malicious attacks. With the rapid growth in full-stack development using modern technologies like several frameworks, novice developers gain significant benefits, but system functionality is often prioritized over web security. Such negligent behavior gives hackers a way to break the confidentiality and integrity of system functions at the same time.

Introductory courses on web development are designed to create useful, efficient applications, but do not pay much attention to important security aspects, such as checking data, managing access, and ensuring safe session management. Without guided learning, they inadvertently create insecure applications, or they often reuse lines of code from open sources that are not verified, or apply AI-based tools that may promote insecure practices. The evolution of the sophisticated nature of Internet threats necessitates a shift in how developers (especially novice developers) are trained to reason in terms of security.

The research addresses the potential of gamification, an effective educational tool and training system to teach students about web security through several game-like scenarios. The game's primary goal is to aware novice developers about their mistakes that can cause several security issues. This will give students educational challenges to assess online threats and vulnerabilities, and teach them how to defend their systems. This thesis also develops some interactive coding elements that promote secure coding practices.

### **1.1.1 Importance of Secure Web Applications and the Threat Landscape**

The evolution of web technologies has paved the path from HTML pages to apps using React.js, Node.js, MongoDB, and SQL, enabling websites to do a lot more. Because of this complexity, cyber risks can target more parts of the network. Problems such as Insecure Direct Object References (IDOR), SQL Injection (SQLi), Cross-Site Scripting (XSS), and Broken Authentication lead to system invasions, reveal user information, decrease system performance, and have negative effects on an institution's reputation.

Threats that are not tackled properly can have catastrophic effects in real-world scenarios. For example, SQL injection bugs were used to have access to the administration section in the FlyCASS airline security portal, which allowed entries in the TSA clearance list to be susceptible to modification, causing potential risks to the airport security path. This kind of violation was connected with the unsafe implementation of third-party services and input processing [38]. Similarly, an exploit of XSS on eBay directed people to credential-phishing pages it showed that even highly scaled networks do not totally clean their front-end, despite their size [26]. These events give us an understanding of how the complexity of the web and insecure coding practices create areas of risk to both individuals and institutions.

Dalai et al. [14] suggest that most data breaches are the result of misconfigured APIs, negligent use of session tokens, or errors in login protection. As attackers improve, simple security mistakes can raise major concerns. Therefore, it is very crucial for developers to keep security in mind from the beginning of the development process to prevent these threats and make the system more secure.

### **1.1.2 Insecure Development Practices by Developers and their Knowledge Gap**

It is often the case that basic web development tutorials frequently overlook information regarding web security—input validation, authentication security, and session management—leaving it entirely up to the students. A reason novice programmers do not understand the significance of security risks in web development is that they often lack foundational knowledge of such risks. Consequently, even though most web applications function properly, they are still vulnerable to various exploits. Bai et al. [38] and Perry et al. [56] have found that programmers tend to use code or resources found on Stack Overflow or use the GitHub Copilot service, usually not being aware of the potential risks associated with these tools. When there is

not much guidance and a lot of pressure, these tasks can be particularly risky for developers. According to Gasiba et al. [47], many developer training programs do not cover secure coding due to time constraints, lack of resources, and appropriate platforms, and they do not prioritize security until a vulnerability is discovered. Additionally, not following the right protocols and using APIs dangerously indicates a lack of knowledge about security [14].

In contrast, Shar et al. [52] state that focusing on well-known issues in the OWASP Top 10 [11] helps students increase their secure coding skills. All in all, these outcomes suggest that students learn secure coding more effectively when they practice hands-on work rather than passive lectures.

### **1.1.3 Gamification as a Tool for Educational Purposes**

Gamification can be defined as a transformation of traditional learning environments as unlike traditional learning methods, it does not only rely on passive learning. It involves the use of design components from games in non-gaming contexts to enhance user engagement and motivation by creating a dynamic learning experience. According to Yu-kai Chou [63], “Gamification is the craft of deriving all the fun and addictive elements found in games and applying them to real-world productive games.” Conventional education often faces common challenges, such as low motivation, limited participation, and difficulty in maintaining concentration. This approach boosts motivation, supports better learning, and helps students focus, participate actively, and enjoy learning by turning lessons into fun. Using games as part of education has quickly become a popular approach for effective learning. By using levels, challenges, instant feedback, and rewards found in games, gamified systems motivate people, help them grasp concepts well, and assist them to stay actively involved in learning. Both Mechanics-Dynamics-Aesthetics (MDA) and Mechanics-Dynamics-Emotions (MDE) reveal that when features like puzzles and feature loops in gameplay affect a person’s emotions, it assists memory and the process of learning new skills [32][35]. Secure programming platforms are discovered to strengthen a person’s coding abilities and keep those abilities for a longer period. For instance, Figure 1.1 points out why a gamification system can be a great tool for educational purposes.

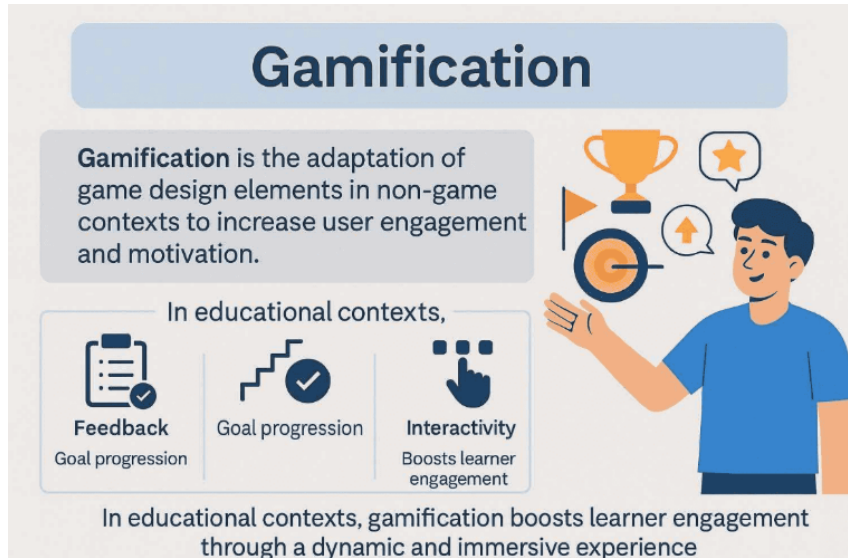


Figure 1.1: Gamification - Educational Tool

Attack-defense topics can be managed through the use of gamification in learning as developers learn how vulnerable code can create problems such as SQLi, XSS, or IDOR and how to mitigate them. Therefore, such hands-on experience inspires people to focus on security earlier rather than later. Likewise, SGI (Sustainable Gamification Impact) recommends that fun tasks and balanced rewards influence learners to stay interested in security training for an extended period [63].

#### 1.1.4 Gaps in Traditional Security Training and Real-World Consequences

Using traditional training methods such as textbooks and set lectures does not guarantee that web applications will be secure. Many developers are not concerned about security until they encounter a security incident themselves as shown by Bai et al. [38].

Both Gasiba et al. [47] and Shar et al. [52] encourage teaching developers about web security from the beginning. They believe that securing user data and sessions should be a basic development skill, not regarded as something unique. Most traditional learning methods usually exclude simulation scenarios and tools that provide feedback.

It is fair to say that developers cannot be prepared against threats with conventional training. The company Medibank failed to use multi-factor authentication, which resulted in hackers being able to log on to the network using stolen user credentials and VPN. After two full weeks, it was discovered that private client information

had been compromised. Fundamentally, the situation is not just a technical failure, but also the cost of ignorance, as it showcases a lack of proactive response against threats in the programming field. Even well-known vulnerabilities are exposed to businesses when secure coding practices are not taught in development courses.

The consequences of these educational inequalities are evident. Moniruzzaman et al. showed that 64% of government and business websites in Bangladesh had serious vulnerabilities such as SQLi and problems with TLS encryption [43]. It is less likely that personal shortcomings cause such errors and more that the training and monitoring processes are not effective. The Bangladesh Bank cyberheist in 2016 [61], which caused the loss of \$81 million, is a clear example of the risks that occur when security is not considered from the outset.

Such reasons highlight the need for training platforms that use games and practical exercises to encourage developers to address today's web security challenges.

## 1.2 Rational of the Study or Motivation

Web applications being the crucial backbone of the internet, compromising one's credibility can hamper both individual and institutional operations. These may arise for several reasons, including malicious attacks or improper testing, but are largely caused by insecure coding practices by novice developers who lack enough security awareness. Sensitive data like personal information can be lost, altered, or exposed as these vulnerabilities give a chance to the attackers to compromise data credibility. Our research [43], shows that Bangladeshi websites have been targeted by several cyberattacks, which emphasizes the necessity of raising urgent awareness. The increasing number of security breaches and illegal access highlights existing weaknesses in the country's digital infrastructure. To validate that statement, this study has analyzed different kinds of attacks, including SQL injection, XSS, Broken authentication, and IDOR.

According to the study of Moniruzzaman et al. [43], 36% of the web pages are secure, while the remaining 64% have various vulnerabilities, especially 53% of the websites are exposed to XSS, 56% of them are exposed to SQL injection, and 15% have been discovered at risk of CSRF. Similarly, a study on Bangladeshi websites that are government and non-government found that out of 359 tested websites, almost 86% of them were vulnerable to SQL injection, with 19% of them at high risk, 18% at medium risk, and 63% at low risk. Moreover, attacks like XSS, Clickjacking, and CSRF were identified as major threats against website security [67]. In 2016,

almost \$81 million was stolen from Bangladesh Bank using illegal Swift transfers and transacted in the Philippines as a result of weak banking infrastructure [61]. According to a report by Kundu et al. [34], almost 52% of banks are still at a high risk of cyberattacks despite the increased IT facilities and security measurements.

As these incidents reveal several infrastructure flaws, several research studies have found a fundamental issue, which is the lack of knowledge and poor coding habits, like the reuse of online code snippets by novice developers. As per Alghamdi et al. [53], several novice developers who are still students often copy code snippets without proper validation from resources like Stack Overflow. As a result, it often leads to poor quality and insecure code usage that is at risk for attacks like SQL injection Cross site scripting. Tahaei and Vaniea [44] further mention that security is often considered a secondary priority in terms of development. Beginner-level programmers rarely receive practical knowledge about safe programming, and numerous renowned online resources provide inaccurate examples. In the deployed applications, novice developers often cause avoidable security errors as they do not realize their mistakes because of their lack of proper knowledge. Additionally, while testing can be a great help to identify the potential attack while building any software, it becomes inefficient due to too many repeated procedures that arise from unsafe coding practices. Since all the vulnerabilities cannot always be mitigated, attackers often take a big advantage of that.

To address these issues, we have decided to work on a gamified system that will help the novice developers to know more about these attacks. Compared to the conventional approaches, gamified approaches increase learning secure coding practices by developing habits through engaging and scenario-based learning. For students and novice developers, the gamified platform aims to raise the standard of web security in Bangladesh.

### 1.3 Problem Statement

With the increasing dynamism of web applications, also increases the scope for intruders to attack and to exploit certain web vulnerabilities. Such vulnerabilities occur mainly due to incorrect implementation and development along with the integration of web technologies [57]. As a result of having a wide attack surface, web-based softwares and applications are targeted [64]. People with malicious intent can steal confidential data, cookies and sessions which can cause irreparable damage to users and service providers, legal and financial loss being one of the many. As a result, security flaws due to insecure coding by web developers need to be addressed more than ever. Testing can be done to mitigate the problem, but if the matter is

not handled from the root, it could result in a greater number of testing iterations and, therefore, higher cost and inefficiency. Relying solely on testing is also an unwise decision because some vulnerabilities might remain unaddressed, which could lead to serious consequences. Previous papers have addressed vulnerabilities caused by insecure coding practices and how to resolve them, but have yet to get to the root cause of the problem, which is developers prioritizing functionality over security despite existing tools and security guidelines. Moreover, the traditional way of education and training is not deemed effective [51], as they fail to motivate developers to develop security awareness and instill its importance in real-world applications. As a result, the gamified approach to education can play a huge role.

## 1.4 Research Objectives

The research aims achieving the following six goals.

- RO1 Investigating the most common web vulnerabilities:** The research identifies common web vulnerabilities, including SQL Injection, XSS, and IDOR, that are common to novice developers, and brings awareness about their impacts.
- RO2 Identifying factors behind lack of security awareness and insecure coding through Literature Review:** The study examines previous literature, articles, and documents to identify the primary factors contributing to novice developers' lack of security awareness and writing vulnerable code despite the availability of security policies and tools.
- RO3 Identifying factors behind insecure coding through Developer Interviews:** The study investigates the thoughts of novice, intermediate, and experienced developers to elicit real-world practices, misconceptions, and industry expectations that contribute to insecure coding practices.
- RO4 Designing and Developing a Game-based Learning Platform:** This research aims to introduce a gamified learning environment to simulate real-world scenarios and give developers hands-on experience on the common mistakes made by developers and how to mitigate common vulnerabilities.
- RO5 Identifying effective Game Evaluation Methods:** This research aims to identify the evaluation methods for the gamified platform and find the most effective one to ensure a proper evaluation of the platform.



**RO6 Evaluating the Effectiveness of the proposed Game:** Evaluation of the proposed game is done by a post-test approach, and feedback is taken from novice developers on the usability and effectiveness of the game.

## 1.5 Scopes and Challenges

Here we are describing the possible scopes of this research and the challenges we might face to implement those scopes.

### 1.5.1 Scopes

#### **Security Education through Contextual Gamification**

Our research creates a gamified platform that will be engaging, and it will teach developers how to design secure websites through real-world scenarios by resembling real-world developer practices. Moreover, it focuses on training the novice developers to recognize and correct their coding mistakes, which is essential for building secure web development behavior.

#### **Focusing on Novice Developer Mistakes**

Our research focuses on novice developers, where secure coding practices are often overlooked. They are frequently exposed to vulnerabilities as a result of improper input validation, authentication, and insecure session handling. These could result in several attacks like XSS, SQL injection, IDOR, broken authentication, and NoSQL injection, which are the focus of this research from real-life examples.

#### **Filling in the Knowledge Gap**

Based on the OWASP top 10, our research deals with the shortcomings related to privacy and safety so that at an early stage, developers can build safe coding practices. It aims to address insecure coding habits like using unreliable tools or copying code snippets from unsafe platforms and AI tools.

## **Short Testing for Evaluation and Learnability using Frameworks**

The gamification is followed by the Octalysis framework for better effectiveness in the educational purpose. The novice developers can take part in short testing sessions so that the usability and effectiveness can be evaluated.

## **Quantitative Assessment of Learning Impact**

Unlike previous studies, this study focuses more on the quantitative evaluation. It can be helpful in terms of measuring their level of understanding, and retention of secure coding practices can be evaluated as well.

### **1.5.2 Challenges**

#### **Insufficient Coverage of Attack Scope**

Although the study aims to teach novice developers about their coding habits that may cause web vulnerabilities, it is not feasible to evaluate all the web-based attacks within a single gamified platform. Attackers can always detect new types of vulnerabilities or techniques to exploit personal and other sensitive information. It can be quite challenging to find out all of them through this study.

#### **Advanced Vulnerability Target Modeling or Neglected Vulnerability**

The majority of gamified systems avoid technically complex and non-conventional attacks since they are either difficult or lack any systematic theory. Including such attacks like DOM Clobbering, CSP Bypass, CSS Injection, and CDN Tampering mechanisms into the gamified platform not only requires a high level of technical knowledge but also must be easily accessible to the learners. It might be a challenging task to prevent it from being overwhelming to the novice developers who will be attempting to use the scenario-based challenges for learning purposes.

#### **Balancing Educational Depth and Engagement through Gamification**

To make sure beginner developers are encouraged, the gamification system must also incorporate a point leaderboard, quizzes, interactive play, and feedback mechanisms that are both encouraging and delightful. It is a challenging task to find a point

of balance when leveraging a system that is both instructive and stimulating for learning web development.

## **Measuring Long-Term Impact**

Measuring long-term testing can be challenging. While short-term results are identified through several interviews and quizzes, it might be more difficult to determine whether a gamified system will result in long-lasting, secure coding practices. Over time, it might not show real-world application. Long-term evaluations require continuous observation and follow-up, which can be quite challenging because of participant availability and the time constraints of this study.

## **Designs of Real-Life Performance Evaluation**

Even though the gamification system can possibly prove to be effective in a controlled environment, it might be quite challenging to estimate the impact on life on the field. The participating developers should be checked to see to what degree they are able to recall their mistakes. It might also be challenging to determine their safe coding practices in the actual framework-based developmental practice, which actively involves the speed of work.

# **1.6 Our Contribution**

This study had the following six contributions:

- **Investigation of the most common web vulnerabilities**

We have identified and validated the importance of ten common vulnerabilities (SQL Injection, XSS, DOM Clobbering, CSP Bypass, CDN Tampering, IDOR, NoSQL Injection, and Clickjacking, CSS Injection, and Broken Authentication), which are common among novice developers, but which are not included in many traditional courses.

- **Identifying factors behind lack of security awareness and insecure coding through Literature Review**

We have identified the main factors that lead to insecure coding practices despite the security policies and tools that are offered, providing us with a theoretical basis for the research.

- **Identified the factors behind insecure coding through Developer Interviews**

We interviewed 32 mid-level, novice, and senior developers in Bangladesh (both working in large and small companies) in semi-structured interviews, which disclosed the following critical themes: functionality-security tradeoffs, the significance of code reviews, and the necessity of inspiring security education.

- **Designing and Development of a Game-based Learning Platform: Play to Secure**

We have created and implemented Play to Secure, a gamified learning platform based on the Octalysis framework, which uses interactive, scenario-based challenges to teach secure coding methods by engaging core game drives instead of traditional methods of learning.

- **Identification of the effective Game Evaluation Methods**

We have selected and used the MEEGA+ evaluation model as the most appropriate tool in evaluating our gamified platform, player experience, perceived learning, and short-term learning outcomes in thirteen dimensions.

- **Evaluation and Validation of the Effectiveness of the proposed Game**

We have tested the platform with the help of 30 novice developers through the MEEGA+ model and found that the platform scored above the 4.0 mark on all dimensions, which indicates that our context-specific solution is practical for them.

## 1.7 Thesis Structure

The rest of this thesis will be structured as follows. Chapter 2 includes a literature review on web security vulnerabilities, insecure coding practices, and gamification for security education. Chapter 3 goes on to talk about the requirements, impact, and limitations of developing the gamified learning platform. Chapter 4 describes the interview process that was used to conduct semi-structured interviews with Bangladeshi developers. Chapter 5 provides the thematic analysis and results of the interview that indicated that the main themes were security education needs and functionality-security tradeoffs. Chapter 6 tells about the design and development of the Play to Secure platform based on the framework of Octalysis. Chapter 7 gives the discussion on design implications and the core game functions. Chapter 8 describes the game evaluation strategy through the MEEGA + model. Chapter 9 is an interpretation of the results of the evaluation. It also discusses the effectiveness

of the gamified approach. Lastly, Chapter 10 provides an overall summary of major findings, contributions, limitations, and future work recommendations.

# Chapter 2

## Literature Review

This chapter presents a number of preliminaries related to our research such as secure coding concepts and its importance, mitigation techniques, few common vulnerabilities, gamification and its use in education and so on. Furthermore, existing research related to our study have been reviewed.

### 2.1 Preliminaries

This section highlights the value of secure coding techniques and typical vulnerabilities as well as the significance of gamification systems and associated frameworks.

#### 2.1.1 Secure Coding Concepts and Importance

Secure coding, or vulnerability-free software development, involves strong code that withstands frequent attacks and adopts best practices in the processing of inputs, authentication, and access control.

Secure code for online applications is required for input fields, session tokens, or API endpoints as they are common targets for exploitation. Input sanitization and validation, avoiding hard-coded credentials, and access control are some of the practices that are encouraged to avoid these issues.

The OWASP Top 10 is a globally recognized standard to identify and address web vulnerabilities, such as injection attacks, authentication, and design flaws [11].

## **Secure Coding Practices and Mitigation Techniques**

### **Access Control**

Access control is the mechanism that ensures users can access only the resources they are authorized to access [56].

### **Session Management**

It refers to using identifiers such as cookies or tokens to securely manage user authentication sessions [56].

### **Authentication**

Authentication is the process of verifying trusted users before giving them access to resources they are authorised to access [1].

### **Input Validation**

Input Validation is the process of ensuring that user input contains no malicious data and conforms to the expected patterns and values [58].

### **Parameterized Queries**

Parameterized queries ensure that user input is treated as data rather than executable code [58].

### **RBAC (Role-Based Access Control)**

RBAC is an access control model that grants access to resources based on the user's role [13].

### **Indirect Access to Objects**

Indirect access to objects refers to when web applications use indirect reference like using hashed identifiers to access internal objects like files, database and URLs [56].

### **Penetration Testing**

Penetration testing is the process of simulating real world scenarios to test for attacks such as IDOR, XSS, and SQLi [56].

### **X frames Options**

X Frames options is a security header that prevents a website from being embedded in a malicious iframe [15].

### **Input Sanitization**

The process of sanitizing input data to prevent malicious input from being executed [58].

### **UI Event Verification**

UI event verification refers to ensuring that user input is being accepted only when the UI elements are visible and intended [15].

## **2.1.2 Common Web Vulnerabilities**

Here we have pointed out frequent vulnerabilities from the points of developers, like SQL injection, Cross site scripting, Broken authentication, IDOR, Clickjacking and Framejacking, NoSQLi, DOM clobbering, CSP bypass, CSS injection and CDN tampering.

### **SQL Injection (SQLi)**

SQL Injection is a security vulnerability that occurs when user supplied data is directly embedded into a SQL query without any sanitization or parameterization. Without any authentication procedures, attackers can gain access to database content which they can then modify or delete. Attackers can also exploit this security flaw to download confidential information [40].

### **Cross-Site Scripting (XSS)**

Cross-Site Scripting (XSS) vulnerabilities occur due to adding user-generated content to the web page without verifying its quality. It allows the attacker to execute malicious JavaScript code in the web browser of other users, hence, obtaining session cookies, redirecting the user to a phishing page or implanting malware into a client machine [20].

### **Broken Authentication**

Broken authentication is a vulnerability that involves a user identity checking process. It is caused due to weak password rules, improper session management or insecure storage of credentials. This vulnerability enables the attackers to act as legitimate users and gain access to user accounts. They can then carry out malicious actions within the system [1].

### **Insecure Direct Object References (IDOR)**

IDOR is a vulnerability that is caused due to weak access control and authentication which occurs mainly when the software application provides direct access to objects based on user input and users can change the URL or parameter. Attackers can gain access to internal resources just by manipulating the identifier in the URL which can result in data leakage, deletion and modification [60].



## **Clickjacking and Framejacking**

Clickjacking mainly occurs when attackers are able to embed the legitimate page of a website in an iframe which contains malicious UI elements. Using clickjacking techniques attackers are able to violate the visual and temporal context integrity of users [15].

## **NoSQLi**

NoSQL injection is often caused by injecting harmful input, which compromises data security and avoids authentication. It allows attackers to make changes to the database because of widespread adoption of databases like MongoDB. Though one might perceive that the NoSQL databases are immune to any injection attack, they are not immune to security by obscurity. Attackers can exploit input by injecting an expression or an operator in a query to fetch the data, raise privileges, or access unauthorized data [28].

## **DOM Clobbering**

Due to client-side vulnerabilities, attackers can inject HTML code with special elements to assume precedence over JavaScript variables or functions in the DOM. Attackers can override client-side input validation. Moreover, they can alter the logic of the application by intercepting the application event handler, disrupting its functionality [71].

## **CSP Bypass**

The Content Security Policy (CSP) is developed to guard against attacks like XSS by restraining the whereabouts of the scripts. However, attackers may bypass CSP as a result of configuring rules incorrectly due to browser behaviour, usage of inline scripts, and whitelisted assets. Thus, it is possible to break the protection tool and inject scripts [65].

## **CSS Injection**

CSS Injection occurs when Cascading Styles Sheets (CSS) fail to filter user input allowing attackers to manipulate page styles, hide part of the information, alter layouts of pages, or even to launch side-channel attacks such as the leakage of data

by spillage, caused by visual cues [12].

## **CDN Tampering**

CDN Tampering involves the distribution of bad or infected files via trusted Content Delivery Networks. Simultaneously, until certain checks, such as Subresource Integrity (SRI) are executed, attackers can distribute malicious scripts to millions of users making it a serious challenge in Third-party resource loading [25].

### **2.1.3 Gamification and its use in Education**

Gamification can be simply defined as adaptation of game design in non-game situations to increase user engagement and motivation. It can be described as the capability to extract all the entertaining and engaging aspects of games and implement them towards real life scenarios or to the things that are productive, as mentioned by Chou [39].

This process of gamification can be applied to various domains to increase attention and motivation, particularly by incorporating elements like feedback, goal progression, and interactivity. In educational contexts, it can significantly boost learner engagement by creating a dynamic and immersive experience. For instance, in secure software training, where practical participation is crucial to understanding abstract security principles, gamification can be particularly effective. It helps learners better comprehend complex concepts by transforming theoretical lessons into interactive experiences [51]. Additionally, gamification fosters a sense of accomplishment and competition, further motivating users to engage with and master the content. By embedding these game-like elements into educational tools, instructors can enhance the learning experience and improve knowledge retention. For more understanding, Figure 2.1 shows several fetures related to the gamification system.

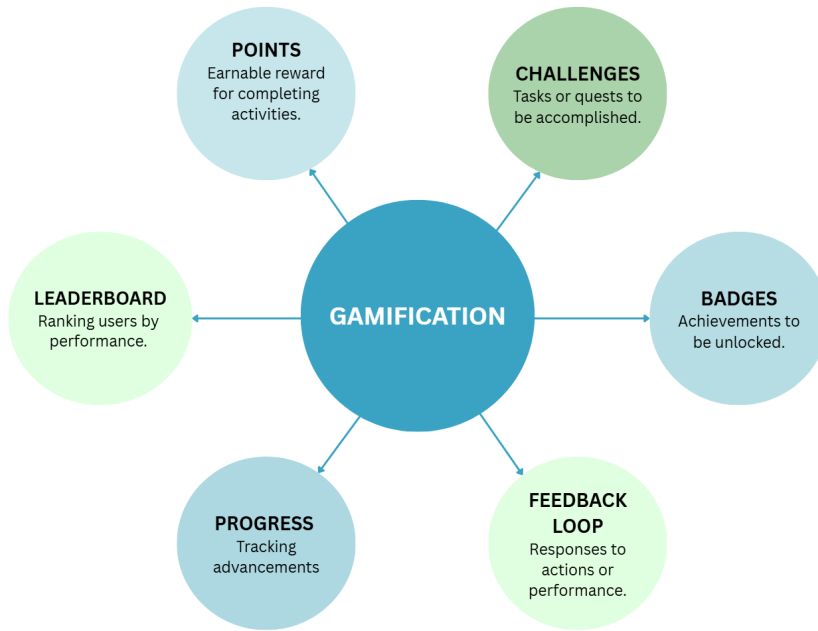


Figure 2.1: Gamification - Features

## 2.1.4 Gamification Frameworks

Structures creation endeavor requires theoretical exposure to the existing structures of gamification. Ivanova et al. [41] summarizes the results of gamification models that are used in the process of developing secure coding training environments.

The models are:

### MDA (Mechanics-Dynamics-Aesthetics) Framework:

MDA framework [32] was first offered to be used in the games' design in a manner that gamified systems would be divided into three components:

1. **Mechanics:** The rule and system parts are points, badges, and levels.
2. **Dynamics:** Dynamics of user behavior: cooperation, competition.
3. **Aesthetics:** Entertainment, fun, or tension caused by the mechanics .

MDA helps the designers to simplify the gameplay framework by becoming acquainted with the motivational responses that some features will tend to trigger. Here, Figure 2.2 shows the MDA framework in details.

### MDA (Mechanics-Dynamics-Aesthetics) Framework

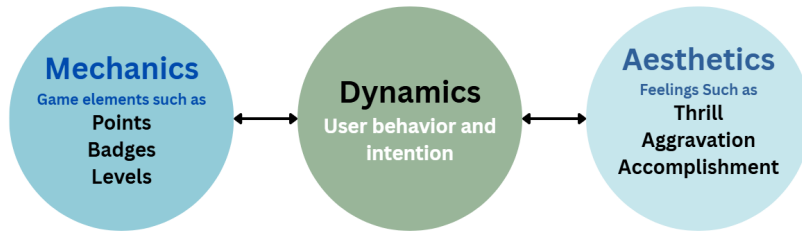


Figure 2.2: MDA (Mechanics-Dynamics-Aesthetics) Framework

### MDE (Mechanics-Dynamics-Emotions) Framework:

The MDE model [35] expands MDA and gives somewhat more emphasis to the affective experiences of the learners. In accordance with it, the mechanics form the user behavior (dynamics), and due to this fact, emotions are evoked. These emotional responses, like thrill, aggravation, or a sense of accomplishment, play a very crucial role in memory retention and strengthening of motivational foundation in learning situations. For better understanding, Figure 2.3 shows how each and every segments are interconnected with each other.

### MDE (Mechanics-Dynamics-Emotions) Framework

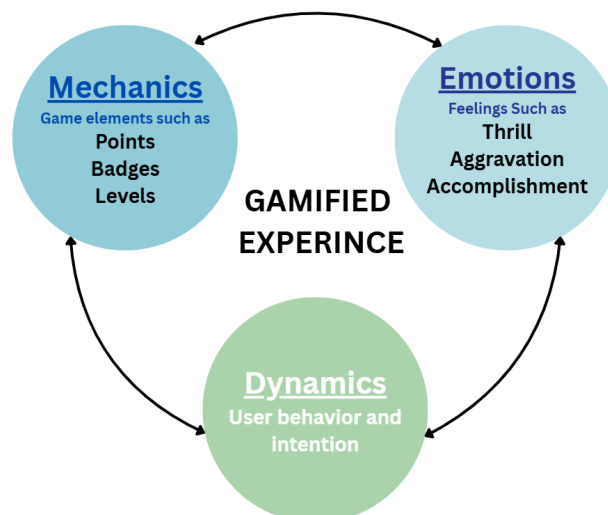


Figure 2.3: MDE (Mechanics-Dynamics-Emotions) Framework

## Octalysis Framework:

The Octalysis model, designed by Chou [35], describes eight core motivational drives, namely epic meaning and calling, loss and avoidance, and others, and divides them into

1. **White Hat, Good, Long term:** mastery, meaning, and independence.
2. **Black Hat, Emergency, Short term:** loss, lack, uncertainty.

This is the Roth model, which was applied to arrange motivational levels in the design of training to ensure that there is participation in the short-term behavior and behavior modification in the long-term behavior. Figure 2.4 represents the core drives of the Octalysis framework.

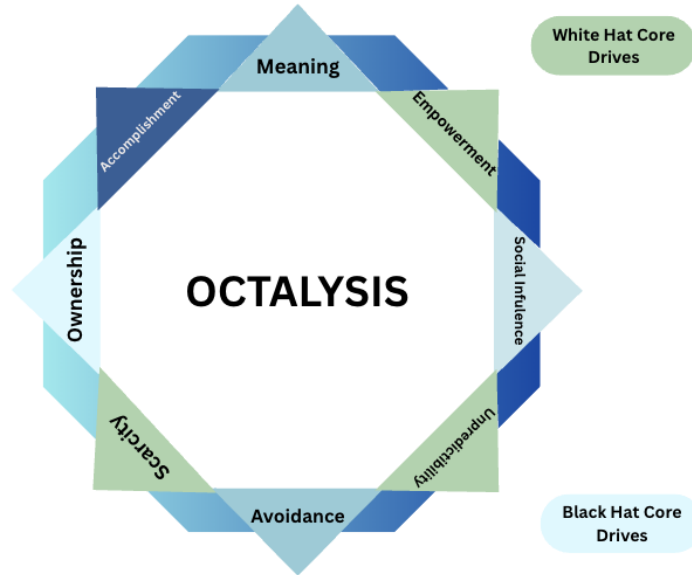


Figure 2.4: Octalysis Framework

## 2.2 Review of Existing Research

The comprehensive literature review was organized into three distinct sections. The first section examines the reasons why novice developers tend to adopt insecure coding practices, which include lack of awareness, knowledge gap and use of insecure tools. Mitigation techniques such as input validation, parameterized queries, and access control are also discussed. Consequently, such insecure practices give rise to vulnerabilities, which are explored in the following section. This includes common

web vulnerabilities such as SQL injection, Cross-Site-Scripting (XSS), DOM clobbering. The last section discussed our rationale behind implementing a gamification system for raising security awareness as opposed to relying on traditional methods. Existing frameworks such as MDA, MDE, and Octalysis are discussed to ensure user engagement and sustainable learning.

### **2.2.1 Security Awareness of Developers and the Impact of Tools**

A study by Gasiba et al. [47] assesses web security awareness and compliance amongst developers. They initially stated that 90% of vulnerable codes are caused by design flaws, according to the Department of Homeland Security. A comprehensive survey was carried out, based on well known ideas like Security-Related Stress Theory and Neutralization Theory. These theories examine how stress causes developers to ignore security policies and why developers are prone to non-compliance. The researchers then found that despite having various security tools and guidelines, developers fail to comply due to reasons such as lack of awareness and knowledge gap, time constraints and deadlines, and the lack of training programs to teach how to mitigate such vulnerabilities. Finally, they provided suggestions such as integrating secure coding practice early on in the development lifecycle to get developers habituated, providing appropriate security training, and fostering an environment that encourages developers instead of causing stress. Moreover, a study by Shar et al. [52] showed that integrating secure coding practices in web development courses significantly decreases vulnerable codes. The authors focus on contextualized learning where new material is introduced to students within the context of its application resulting in faster learning. They incentivised students to use security tools in their coding assignments and taught the OWASP top 10 Vulnerabilities early on. This study concluded in enhanced security knowledge and implementation which would later help students in industry level. Bai et al. [38] has researched why web developers use insecure code snippets from online forums such as Stack Overflow. They surveyed developers whose open source Github repositories contained exact snippets from these kinds of forums. Moss was used for code clone detection and an anonymous survey was taken to find out that developers knowingly use such snippets due to overconfidence in their security skills and prioritizing functionality over security. Moreover, many did not have enough knowledge to validate such code snippets and believed it was not their responsibility to do so. This study concludes that not addressing these problems from the beginning of the development process resulted in far more insecure codes. Perry et al. [56] came to a similar conclusion. They talk about AI assistants and bots like Github Copilot which generate insecure code

snippets which are then used by developers who do not have the necessary security knowledge. Despite providing functionally correct codes, AI tools overlook the security threats that come with such codes. Two groups of participants were studied, only one being allowed to use AI assistants and it was discovered that the group using AI wrote more insecure codes which they were confident was secure .

## 2.2.2 Attacks in Focus

This study segment explores the literature based on the vulnerabilities commonly discovered as a result of novice developers' lack of insight.

### SQL Injection

Over the past few years, SQL injection (SQLi) vulnerabilities have seen a significant rise, with reported cases increasing from 100 to over 7,000 according to the National Vulnerability Database [40]. According to Kamau et al. [69], SQL injection flaws are a major contributor to cyberattacks, leading to serious consequences such as data leaks, system disruption, and data manipulation or destruction. In fact, they account for approximately 60% of all reported vulnerabilities and are classified as critical under the CVSS v3 scoring system. The authors warn that these attacks can result in data breaches, denial-of-service conditions, and even unauthorized system-level control. Furthermore, according to Gupta et al. [40], SQL injection attacks can take different forms: In-band, Inferential (Blind), and Out-of-band techniques; these can then be categorized into methods such as error-based, union-based, tautology-based, and time-based attacks. Research on SQL Injection is typically divided into two main areas: detection and prevention. According to Kamau et al. [69], detection focuses on identifying vulnerabilities within web applications with the help of a variety of tools like BSQL Hacker, SQLmap, SQLninja, and Safe3SQLInjection; these tools have been developed for detecting vulnerabilities. On the other hand, Gupta et al. [40] emphasize prevention strategies focused on safe and secure development practices, particularly input validation and filtering, to reduce the risk of injection-based exploits . According to Halfond et al. [9], "SQL injection vulnerabilities have been described as one of the most serious threats for Web applications". They categorize SQL injection into tautology-based attacks, piggybacked queries, second-order injections, and blind SQLi. Tautology-based injections exploit conditional logic which allows unauthorized access by transforming queries to always return true. This paper further claims, developers were making an effort to detect and prevent SQLIAs, but failed to do so adequately and in every needed location. They claimed that even though developers were trying to detect and prevent SQL

injection attacks but were unable to do so. This human factor: overconfidence, ignorance, or time pressure, is the rationale for this thesis' gamified training model. According to Elshazly et al. [18], "SQLIA are easy to learn and exploitable, so this method of attack is easily used by attackers". They show that firewalls and other intrusion detection systems (IDS) and even antivirus cannot identify SQL injections attacks as these attacks happen at the application layer. This paper also shows the growing sophistication of SQL injection attacks and also highlights the necessity of better mitigation techniques.

## **Cross Site Scripting (XSS)**

XSS is considered to be a very critical client-side vulnerability. It occurs within the React.js front-end framework. Chowdhury [1] highlights several entry points for potential XSS attacks due to its dynamic rendering of user generated content. If user inputs are not properly sanitized, an attacker can easily inject scripts through input options or other interactive components of the webpage. This is especially dangerous when developers misuse React's dangerouslySetInnerHTML property which allows raw HTML to be rendered as per Kaur et al. [20]. Chowdhury [1] further references several cases in which improperly handled user input in React leads to XSS, resulting in risks such as session hijacking, token theft, or malicious redirects. Some of the mitigation strategies include: Sanitizing Input and Encoding Output, Content Security Policy (CSP), Secure Cookies and Developer Education. The interviews conducted with developers at Bangladeshi software firms highlight that XSS is a common vulnerability found in web applications. Several firms had claimed that it had started using automated tools along with manually reviewing codes to detect and prevent XSS during development. One also pointed out that there has been a decrease in XSS attacks after the introduction of development workshops that focused on stronger security while development. In conclusion, XSS remains a critical threat in MERN stack applications which require rigorous input validation, secure coding practices, and consistent awareness training. Lui et al. [42] highlight a broader problem in developer's cybersecurity knowledge with the help of the persistent cross-site scripting attacks and vulnerabilities in modern, security-critical web applications. They cite more than 3.7 million form hijacking attempts, through exploiting XSS vulnerabilities, intercepted in 2018 alone. They recommend multiple development disciplines and automated tools. They also categorize detection methods: static, dynamic, and hybrid analysis. Tools like eDacota, XSS-Guard, and JSPChecker were recommended for analyzing XSS vulnerabilities.



## Broken Authentication

Authentication and authorization are key pillars of web application security. Chowdhury [1] highlights that improperly implemented authentication and authorization mechanisms can have various vulnerabilities, which could include unauthorized access, privilege escalation, and data breaches. In MERN applications, authentication mechanisms such as JSON Web Tokens, OAuth, or session-based approaches are commonly used. When these systems are misconfigured or when authorization checks are insufficient, they may expose the application to risks such as token theft, session hijacking, and unauthorized privilege escalation. MERN applications usually use Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) models. Blundo et al. [45] highlight how RBAC gives permissions based on user roles but may lack the flexibility needed for dynamic or large-scale applications. On the other hand, Siderova et al. [66] highlight how ABAC has more adaptability by taking into account attributes like the user or the type of device used by the user. Chowdhury [1] also notes that challenges still remain consistent in large systems with multiple overlapping permissions where one incorrect role assignment could lead to significant security vulnerabilities.

## IDOR

Amissah and Bentil [60] highlight the significance of web security by exposing IDOR (Insecure Direct Object Reference) vulnerabilities in an academic website and how they allowed unauthorized access to sensitive data. As legitimate users, the authors were able to access sensitive documents, including author certificates, acceptance letters, and other users' payment records, by merely altering the document ID in the request parameter. This highlights critical privacy vulnerabilities in the system. By referencing OWASP top 10 cheat sheet, the authors were able to conclude that this attack falls under the IDOR category and gave suggestions on how to mitigate it, like using RBAC and indirect access to objects. Furthermore, using a bug bounty-style methodology, Shivam Singh and Monika Dandotiya [58] investigate IDOR by first identifying the vulnerable web application and parameters that use identifiers similar to the one in this example: `example.com/orders?id=54321`. They then check if the parameters can be manipulated and if they can gain unauthorized access to files, sensitive user information and access databases. They discuss mitigation strategies such as implementing access control, using parameterized queries, validating user input and conducting thorough penetration testing and regular security testing. This method is repeated on multiple applications and parameters to discover other insecure endpoints.

## Clickjacking

Huang et al. [15] researched different clickjacking methods used by attackers, like layering invisible UI interfaces, leveraging CSS, deceiving users using timing based cursor jacking. Existing mitigation techniques such as X frames options, Frame busting Scripts and built in protections in browsers are discussed and improved defense mechanisms such as CSP, UI event verification is proposed in this paper. It is suggested that developers ensure browser events like clicks and hovers are not being manipulated and user actions are prevented if a UI element is not visible. Detection of malicious overlays and prevention by blocking third-party framing is suggested. A research by Rydstedt et al. [13] gives us an insight on the top 500 websites according to Alexa and their use of frame busting defenses to mitigate clickjacking problems and the limitations to such defenses. They came to the conclusion that only 14% of the top 500 websites used any form of frame busting, most of which follow a similar pattern of having a conditional statement and a counteraction action that helps navigate the legitimate page to the top. They also mentioned how attackers use circumvention techniques like double framing or exploiting XSS filters and using `onBeforeUnload` handlers to bypass simple frame busting strategies. The authors then discuss defense recommendations such as X frame header and content security policy (CSP).

## NoSQL Injection

According to Ron et al. [28], MongoDB vulnerabilities usually arise due to a lack of user input verification or proper sanitization before data is inserted in the database that allows attackers unauthorized access to the database, allowing them to mess with data, delete tables, or even steal sensitive information. For example, by inserting `$ne` (not equal) operators or by using `BOOLEAN` conditions where the result always returned “True”, attackers were easily able to alter the reviews of a product in a vulnerable web application. Similarly, Hou et al. [27] added that URL-based injections take advantage of how MongoDB works, as it accepts operators like *ne* and *or*, which can give sensitive data to the attackers. Additionally, Ron et al. [28] have also mentioned several attack types, such as JavaScript injection, union queries, and Cross-Site Request Forgery (CSRF), that frequently allow attackers to evade authentication systems, as these injections take advantage of the dynamic query structure of MongoDB. Because of JSON-like format, novice developers often assume that NoSQL databases are relatively more secure than traditional SQL systems, whereas their lack of awareness and safe coding habits make them extremely vulnerable to these attacks.

## DOM Clobbering

Liu et al. [71] introduced Hulk, an automatic framework based on the Symbolic DOM that finds and exploits DOM Clobbering bugs. It states that DOM Clobbering is a harmful attack that relies on code reuse and leads to issues on websites by confusing uniform identifiers in HTML and JavaScript, often ending in very serious problems such as XSS or CSRF. Instead of using a single defence system and evaluating static code, Hulk can address attacks that might happen while code is running. Static methods won't find any clobbering gadgets in JavaScript code that features dynamic behaviour, aliasing or many nested if statements, claim the authors. Hence, Hulk builds a Taint Dependency Graph with the help of taint tracking and concolic execution and settles it with constraint modelling by using Symbolic DOM. Those testing on the Tranco Top 5,000 uncovered 497 security issues, among them zero-day flaws, in Google Client API, Webpack, MathJax and Closure, all popular JavaScript libraries. As a result, it was discovered that a feature in the Google API library made it possible for attackers to insert `<iframe name = "scripts">` items into the affected web applications and control them with new `Function()` calls. He also mentioned that sanitisers and CSPs do not work well if developers are unfamiliar with the handling of variables and conflicts by browsers. Ultimately, the study reveals that these weaknesses develop when programmers do not realise HTML and JavaScript might be mixed. According to Khodayari and Pellegrino [54], the difference between what users assume is secure and the true safety of their data was demonstrated when documenting over 70 gadgets that used CSP and sanitisation yet were still found to be vulnerable. Looking at each tool individually and applying symbolic logic, they make sure to stress that under recognised clobbering is still a severe threat. These interviews prove that DOM Clobbering is happening more since it affects many browsers due to old ways and people's habits.

## CSP Bypass

Nowadays hackers can easily bypass their initial constraints by using broad script whitelists and unsafe-inline directives, because poorly designed CSPs might make the countermeasures useless (Weichselbaum et al.) [29]. Additionally, as Pylp [65] has mentioned, hackers obtain sensitive information by inserting malicious JavaScripts that secretly redirect to servers used by the hacker. For example, a hacker avoids CSP enforcement techniques and exploits using the `window.stop()` method to obtain session-related data without notifying the user. In addition to that, Weichselbaum et al. [29] have discovered almost 94.72% of CSP policies were useless against XSS by evaluating almost 26,011 real-world CSP regulations because they used APIs like JSONP or AngularJS. Because of a lack of experience and proper knowledge about

these ways of attacks, novice developers trust third-party websites that contain dangerous resources, and even developers with enough experience accidentally increase the risk. They can better protect their applications from potential attacks like XSS by understanding the vulnerabilities that occur because of these APIs. That's why creating awareness among novice developers against these attacks has become crucial, which requires stricter CSP implementations and more comprehensive training for them on secure coding practices.

## **CSS Injection**

Huang et al. [12] argued that CSS should be considered vulnerable, disputing the common belief that CSS is secure and unnoticeable. It was revealed that by organising specific CSS on web pages, attackers might steal auto-filled usernames and emails and access them during cross-domain resource load. Because scripting is limited, this technique helps users evade JavaScript blockers and specific security policies known as CSP. They proposed Same-Origin CSS (SOCSS), a rule that blocks access to stylesheets from different websites, and this rule is used by modern browsers now. Moreover, Jin et al. [19] have looked into attacks on HTML5 applications for mobile devices built using PhoneGap and similar tools. Based on analysing 15,000 APK files, researchers discovered that 478 Android apps are at risk from code injection, even when the user interacts with very simple things like LEGO blocks or sends an SMS. They observe that these issues arise due to developers' misapprehension about the execution of CSS. Even with SOCSS and improved input checks, many CSS issues are caused by misunderstandings; such points are rarely covered in official training. Above literature reviews suggested interactive solution that shows the risks to ensure that new coders learn to respond. To ensure developers do not make mistakes, they also suggested to include CSS injection real-life scenarios in early training so they treat design and implementation together.

## **CDN Tampering**

Chen et al. [25] discussed what CDN forwarding-loop attacks are and described how even minor mistakes in redirection and caching errors can lead to significant Denial of service (DoS) incidents. In the paper titled CDN Loops, the researchers demonstrated how attackers can design and take advantage of URLs to have two or more CDNs continuously sending requests back and forth. It is believed that this mistake is a result of how developers handle fallback origins and redirect settings, as the novice developers lack a clear understanding of CDN connections. Likewise, Triukose et al. [10] discovered that developers lacked appropriate handling of caching

and checking how valid their request paths were, resulting in frequent exploitation of both Coral and Akamai CDNs. Random strings and modified headers enable the attacker to make each request evade the cache, forcing the CDN to request a new response from the origin server, leading to heavy malicious traffic. Since edge servers are considered credible infrastructure, the origin server ends up receiving lots of simulated requests that look similar to genuine user actions. Furthermore, Gillman et al. [22] went on to say that, even though CDNs help reduce the control of common volumetric attacks, not fully understanding DNS, caching and TTL policies can make them vulnerable. In another study, Liang et al. [21] analysed HTTPS and CDNs together and found that mismanagement or oversights with certificates and keys may lead to possible cases of spoofing, downgrading the security and failing authentication. These studies highlight that most of the time, developers put CDNs in place without looking into the possible dangers of how they are configured. Each of these papers stresses that the critical risks in CDNs are often caused by the developer not fully knowing how it works and cannot be blamed solely on malicious actors. From these concepts, author portrayed how lack of awareness and wrong assumptions by developers can lead to problems in system forwarding-loop issues and how we can tackle these by teaching system-level thinking in gamified tutorials.

### 2.2.3 Gamification System and Frameworks

Gamification enhances motivation, facilitates improved learning, and encourages students to focus, engage actively, and enjoy the learning process by transforming lessons into enjoyable experiences. Learning software development—which involves a multitude of coding languages, concepts, and design principles—can be particularly challenging for novice developers. However, gamification can significantly aid these developers in learning more effectively. Priyadi et al. [50], after reviewing 23 recent studies on learning software development through gamification, from IEEE Xplore, ProQuest and Scopus found that common gamification elements such as points, levels, badges, leaderboards and social engagement are widely applied to encourage productive behaviour. Nardo et al. [63] conducted a systematic review of 68 studies published between 2015 to 2023, adhering to PRISMA guidelines, on the application of software engineering education and training. They also identified the utilization of comparable mechanisms explored how gamification integrates game elements like quests, badges, points, and challenges into non-game contexts. Furthermore, Arif et al. [37] explore the design and implementation of an interactive web-based coding application to address the challenges faced by students in learning web programming languages such as HTML, CSS, and JavaScript, which are often difficult to grasp through traditional, non-interactive teaching methods using a combination of ADDIE instructional design model and prototyping a software

development the platform which offers real-time coding, rewards points and provide immediate feedback. For validating the application, they have done qualitative testing where both experts and students achieved an average score over 87% which demonstrates that the application is motivating, engaging, and supports learning. However, the study focuses on short-term engagement and lacks an assessment of long-term retention and sustained interest.

True gamification is about identifying the motivations of individuals and developing experiences that connect those reasons to drive changes in behavior and learning rather than about points, badges, and leaderboards[39]. Frameworks help structure gamification by connecting game mechanics to underlying psychological motivators and learner engagement. Some popular frameworks known in gamification include MDA, MDE, Octalysis, and Sustainable Gamification Design (SGD) [41]. Priyadi et al. [50] found that none of the studies use a standardized framework, as all of them rely on their own individual approaches. In contrast, Nardo et al. [63] use popular frameworks like Octalysis, MDA, and motivational theories like self-determination theory. However, there isn't any single dominator standardized framework that is universally adopted in software engineering education. Hunicke et al. [5] formalize game design into three interconnected components: mechanics (rules and features such as points, badges, and missions), dynamics (player interactions with these mechanics, fostering behaviors like competition, exploration, and collaboration), and aesthetics (the emotional and motivational experiences such as challenge, discovery, and narrative immersion). He analyzed a babysitting game using the MDA (Mechanics, Dynamics, Aesthetics) framework. The game evolved in three passes: simple exploration for young children, more challenges and complexity for kids 7-12, and competitive, strategic gameplay for older players. This progression targeted intrinsic motivations like challenge and competition by adjusting the mechanics, dynamics, and aesthetics at each level. Robson et al. [24] adopted the mechanics-dynamics-emotions (MDE) framework from MDA(Mechanics-Dynamics-Aesthetics) and focused on emotions because they capture better effective outcomes. The emotional responses of players, such as excitement, frustration, and triumph, drive engagement and behavioral changes. The MDA places a broader focus on aesthetic experiences evoked by gameplay, whereas the MDE emphasizes emotional engagement as central to motivating behavior change. Chou [39] proposed eight core drives—Epic Meaning and Calling, Development and Accomplishment, Empowerment of Creativity and Feedback, Ownership and Possession, Social Influence and Relatedness, Scarcity and Impatience, Unpredictability and Curiosity, and Loss and Avoidance that help structure gamification in education by enhancing motivation, engagement, and learning outcomes. Additionally, the Octalysis framework distinguishes between White Hat drives (positive, empowering—such as Meaning, Accomplishment, and Creativity) and Black Hat drives (urgent, pressuring—such as Scarcity, Unpredictability, and

Loss), enabling a balanced design that promotes long-term motivation without creating negative stress. Later on, CoMa [33], a gamification-based e-learning system designed to teach Java Programming to improve student learning outcomes, engagement, and motivation. The game uses the same mechanisms (leaderboards, experience points, badges, challenges, and leveling) to create a competitive and interactive learning environment. To ensure that its features function properly and meet the desired objectives, the system uses white-box and black-box testing. The results were positive; CoMa successfully motivated students and enhanced their learning experience, making it more enjoyable and effective.

## 2.3 Summary of Key Findings

The literature clearly indicates the lack of secure coding practices by novice web application developers, especially because the element of web security is not given the utmost emphasis in the formative stages of training programs on web development. Such critical vulnerabilities as SQL injection, XSS, IDOR, CSRF, DOM clobbering, or CSP bypass [38][56][28] are typically linked to unsafe coding practices, the unverified reuse of code, or the absence of a secure system of authentication or input validation. Studies by Bai et al. [38] and Perry et al. [56] further claimed that because of being unaware of potential risks of using insecure lines of code from Stack Overflow or AI-powered coding assistants, such as GitHub Copilot, the vulnerability disclosure expands. These practices have also been verified through our literature review while investigating the common web vulnerabilities caused by novice developers due to a lack of knowledge, which is mentioned in research objective as RO1.

The literature also states that gamification based on the frameworks, including MDA, MDE, Octalysis, offers a transformational experience of learning to impose the habits of secure coding. With the combination of the affective interactions (through MDE), motivation stimuli (through Octalysis), gamification would create engagement and behavior change among the developers [39][24]. It further testifies to the fact that gamification can be of use in the field of enhancing learning, especially when done on real-life security issues.

More than 60% of the sites examined in Bangladesh, both government-owned and privately owned, are in a dire state, according to empirical studies by Moniruzzaman et al. [43] and Ahmed [67]. The primary cause of this is that the web developers did not initially consider secure design. By integrating gamification approaches in the training of early development, this research will eradicate the awareness-implementation gap.

Even though prior literature explored gamified learning and secure coding, majority of the presented sources do not include the existence of vulnerabilities that can be associated with a specific gamification framework and apply to the recent stacks, e.g., DOM clobbering or CSP bypass. It should also be mentioned that consistent and systematic reviews of gamified tools containing real-life coding tasks or cognitive assessments do not exist. Finally, the majority of gamified apps work with theory rather than simulators of practical simulations of real-world attack and defense scenarios. This study attempts to fill such gaps by taking a context-sensitive, technically inclined, behavioral approach towards the development of a secure web development course.

In conclusion, the results demonstrate that, according to the current models of teaching and based on the known vulnerabilities of security, gamification is not just a convenient but an essential approach to enhancing secure coding habits in novice programmers. It also fills the gap of knowledge that lies between the acquaintance of the theory of web security and practice and provides a certain level of scale that can be utilized to fight the increasing threats of web security. Table 9.2 in Chapter 9 provides a comparative analysis of our research against existing studies.



# Chapter 3

## Methodology

This chapter describes the methodology of our research. The entire procedure is broken down into three steps, each one of which develops out of the previous one to create a workflow. The first stage is the discovery of vulnerabilities and the discovery of a gamification framework to develop a foundation. The second step is dedicated to the design of the gamified platform, and an interview is carried out to learn security practices and focus on the real-life developer experience for additional insight. The third and last stage is the implementation and evaluation of the gamified platform to determine its effectiveness. To be more specific, Figure 3.1 shows the general flow of the methodology.

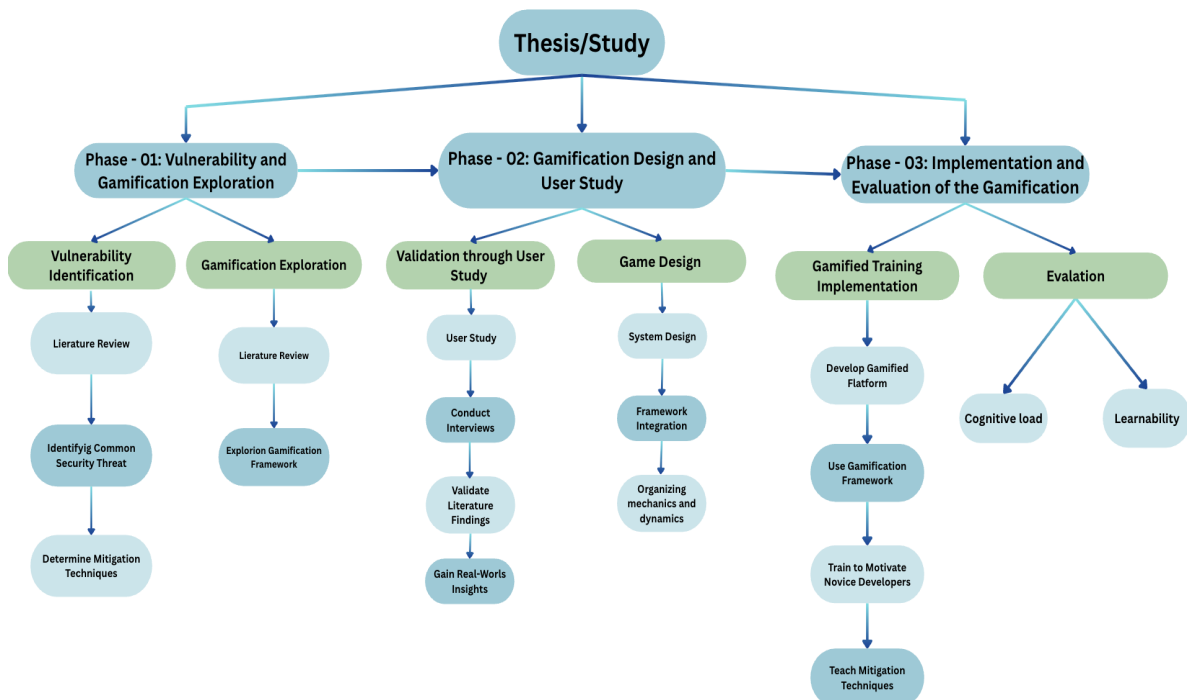


Figure 3.1: Detailed Methodology

## **3.1 Phase 1: Vulnerabilities Identification and Gamification Exploration**

During the initial stage of the study, the researcher determined the secure coding vulnerabilities that are typical among novice developers. Vulnerabilities that are brought about by poor coding practices are also discussed in terms of mitigation strategies. Furthermore, the use of gamification instead of conventional training approaches is discussed, as well as the frameworks. Identification of vulnerability and exploration of gamification divided into 2 segments:

### **3.1.1 Identification of Vulnerabilities**

#### **Literature Review**

An extensive literature review is done on typical web vulnerabilities, with several of them being emphasized in the OWASP Top 10. These are SQL Injection, Cross-site Scripting (XSS), Broken Authentication, Dom Clobbering, CSS Injection, CDN Tampering, NoSQLi, Clickjacking, and IDOR, among others. The review also found reasons behind the insecure development practices that included stress coding, time constraints, unawareness, use of AI-generated code, and insecure code snippets on the internet. The methods of mitigation that were examined included input validation, parameterized queries, secure session tokens and access control mechanisms.

### **3.1.2 Gamification Exploration**

#### **Literature Review**

Gamification has been researched as a replacement of conventional training techniques, with the help of a broad literature review. To determine how effective it is in enhancing the capacity of novice developers to acquire secure coding practices.

#### **Gamification and Framework Exploration**

To determine the elements that are best in the promotion of engagement, motivation and knowledge retention the MDE, MDA and Octalysis frameworks were investigated.

## **3.2 Phase 2: Gamification Design and User Study**

In the second step, the information gained during the literature review and user study is synthesized to inform the design of the gamified system.

### **3.2.1 Interview for Validation and Additional Insight**

An interview has been carried out with developers in order to understand not only what they know about secure coding and what vulnerabilities they have to deal with in real life, but also what difficulties they experience when implementing mitigation strategies. Their views regarding the adoption of secure coding practices have also been investigated in the interview. The findings have confirmed the findings of the literature review confirm the knowledge gaps and coding behaviors that will result in vulnerabilities.

### **3.2.2 Framework-Based Game Design**

According to the interview and study results, the gamified platform has been designed. This involves the choice and incorporation of adequate frameworks (MDE, MDA, Octalysis) to organize the mechanics, dynamics, and motivational influences of the game. The conceptual framework modules, challenges, and interactive components that are associated with certain vulnerabilities and mitigation practices. This phase makes sure that the gamified system is pedagogically viable and feasible.

## **3.3 Phase 3: Implementation and Evaluation of the Gamification**

The third and final phase focuses on the implementation and effects of the gamified training system using quantitative analysis.

### **3.3.1 Creation of the Gamified Platform**

A prototype of the game is created first to model secure coding practices. The modules tackle different vulnerabilities. It also includes progress tracking, points, badges, and rewards to improve engagement using frameworks and retention.

### **3.3.2 Evaluation Criteria**

#### **Cognitive Load**

The ability of the participants to process and retain secure-coding knowledge has been assessed. The MEEGA+ model is used to measure cognitive load in order to compare the gamified model with the traditional educational methods.

#### **Usability and Learnability**

To evaluate the model, usability, and learnability, comprehensive feedback have been taken from the participants. The results of the feedback will be analysed to determine the efficiency of the system.

The findings and recommendations extracted from the literature review and user study have been implemented through a gamification system along with a real-world evaluation process. The training system will ensure that knowledge gaps and insecure coding behaviour of novice developers are minimized by being both engaging and educational.

# Chapter 4

## Semi-Structure Interview

The chapter describes the approach taken to design, implement and analyze a set of semi-structured interviews used to research the security practices of software developers and their attitudes towards gamification-based software learning as a means of learning to write secure code. It also describes the development of the interview protocol, the method of recruiting and categorising participants, data collection and analysis. The pilot testing procedures, method of analysis and ethical considerations that were observed to proceed the research are also presented in the chapter.

### 4.1 Interview Approach and Participants' Engagement

#### 4.1.1 Overview of Participants and Study Approach

A total of 32 software developers from Bangladesh participated in the study, which covers a wide range of organisational contexts and professional roles. To represent knowledge, practice, and perspective variations among stages of professional development, participants have been grouped by experience level: 20 novices, 9 mid-level, and 3 senior developers. In Table 4.1, the demographics of novice developers have been shown. Moreover, in Table 4.2, the demographics of the mid-level and in Table 4.3, the demographics of senior developers have been shown. For clarity, JD is used to denote Junior Developer, while SE refers to Software Engineer.

The interviews were semi-structured, so there was consistency through sessions, but there was flexibility to explore emerging ideas in greater detail. This method encouraged participants to present real-life experiences based on development processes,

awareness of vulnerability, and the perspective on the effectiveness of gamified security learning. The interviews were carried out in person or via the Internet and were recorded with permission to use, and transcribed.

Table 4.1: Demographic Information of Novice Developers

| ID  | Age   | Gender | Experience<br>(Yrs) | Job<br>Role | Training | Projects | Code<br>Reviews | Work<br>Mode |
|-----|-------|--------|---------------------|-------------|----------|----------|-----------------|--------------|
| N1  | 18-24 | Male   | 1                   | JD          | No       | 3        | Sometimes       | Team         |
| N2  | 18-24 | Female | 1                   | JD          | No       | 3        | Sometimes       | Team         |
| N3  | 18-24 | Male   | 1                   | JD          | No       | 3        | Sometimes       | Team         |
| N4  | 18-24 | Male   | 1                   | JD          | Yes      | 6        | Not much        | Team         |
| N5  | 18-24 | Male   | 1                   | JD          | No       | 4        | Not much        | Both         |
| N6  | 18-24 | Female | 1                   | JD          | Yes      | 4        | Not much        | Team         |
| N7  | 25-30 | Male   | 1                   | JD          | Yes      | 3        | Not much        | Team         |
| N8  | 18-24 | Male   | 1                   | JD          | Yes      | 4-5      | None            | Both         |
| N9  | 18-24 | Male   | 1                   | JD          | No       | 3        | None            | Team         |
| N10 | 18-24 | Male   | 1                   | JD          | Yes      | 8        | Never           | Both         |
| N11 | 18-24 | Male   | 1                   | JD          | No       | 8        | Sometimes       | Team         |
| N12 | 25-30 | Male   | 2                   | JD          | Yes      | 4        | Regularly       | Individual   |
| N13 | 18-24 | Male   | 2                   | JD          | Yes      | 3        | Never           | Team         |
| N14 | 18-24 | Male   | 2                   | JD          | No       | 4        | None            | Team         |
| N15 | 18-24 | Female | 1                   | JD          | Yes      | 3        | None            | Team         |
| N16 | 18-24 | Male   | 1                   | JD          | Yes      | 7        | Sometimes       | Team         |
| N17 | 18-24 | Male   | 1                   | JD          | Yes      | 9        | Regularly       | Team         |
| N18 | 18-24 | Male   | 1                   | JD          | Yes      | 10       | Regularly       | Team         |
| N19 | 18-24 | Male   | 1                   | JD          | No       | 3        | None            | Team         |
| N20 | 18-24 | Male   | 1                   | JD          | No       | 4        | None            | Team         |

Table 4.2: Demographic Information of Mid-level Developers

| ID | Age   | Gender | Experience (Yrs) | Job Role | Training | Code Reviews | Work Mode |
|----|-------|--------|------------------|----------|----------|--------------|-----------|
| M1 | 25-30 | Female | 4                | SE       | No       | Regularly    | Team      |
| M2 | 25-30 | Male   | 3.5              | SE       | Yes      | Regularly    | Team      |
| M3 | 25-30 | Male   | 4                | SE       | Yes      | Regularly    | Team      |
| M4 | 25-30 | Male   | 4                | SE       | No       | Regularly    | Team      |
| M5 | 25-30 | Male   | 3                | SE       | Yes      | Regularly    | Team      |
| M6 | 25-30 | Male   | 3                | SE       | No       | Regularly    | Team      |
| M7 | 25-30 | Male   | 4                | SE       | No       | Regularly    | Team      |
| M8 | 25-30 | Male   | 4                | SE       | No       | Regularly    | Team      |
| M9 | 25-30 | Male   | 4                | SE       | No       | Regularly    | Team      |

Table 4.3: Demographic Information of Senior Developers

| ID | Age   | Gender | Experience (Yrs) | Job Role           | Training | Code Reviews | Work Mode |
|----|-------|--------|------------------|--------------------|----------|--------------|-----------|
| S1 | 25-30 | Male   | 6                | Senior Engineer    | No       | No Data      | Team      |
| S2 | 36-40 | Male   | 14               | CTO                | Yes      | Regularly    | Team      |
| S3 | 36-40 | Male   | 18               | Software Architect | Yes      | Regularly    | Team      |

The interview questions were formulated with great care based on three major inputs, which included a thorough literature review on the vulnerabilities chosen, a thorough insight into the secure coding practices and a succession of interviews with security experts. These professionals were essential in examining and certifying the sets of questions so that they could be pertinent as well as fit the various levels of developer experience. The consultations were conducted in several informal discussions where the research team could refine the questions in a manner that made it very close to the objectives of the study. The key aim of this process was to understand better the process of development, secure coding behaviours, and typical weaknesses that are manifest in various levels of professional experience.

The interviews were arranged in three successive circles, each of which was devoted to a specific group of participants:

### **Round 1: Novice Developers**

The first round was aimed at novice developers to scale their level and awareness of secure-coding expertise and awareness. Moreover, we investigated their preferred methods and formats for learning security concepts.

### **Round 2: Mid-level Developers**

The second round was designed with the integration of security concerns into everyday development, and with learning preferences and the source to their security knowledge (as possessed earlier as novices or later in industry practice, organisational conventions or formal courses).

### **Round 3: Senior Developers**

The last round discussed the views of senior developers on the pitfalls that may befall novices, how they mentor novices to develop a secure mindset, how they evaluate the status of novice developers, and what their recommendation is to teach security in teams.

The participants were initially grouped into novice, mid-level or senior developers through a short background and demographic block of questions that included years of experience, career stage and project history. This level-based classification made the comparisons between levels possible.

At this point interviews were conducted under two general themes across all cohorts: Insights on Security Vulnerabilities- discussing the most common vulnerabilities and how security practices are implemented. Secondly, Gamification Systems- exploring preferences to game-based strategies of enhancing good practice in secure-coding. As the question forms were similar to allow structured cross-level analysis, the content focus was differentiated as a novice would be established in baseline understanding and eliciting preferred learning modes, mid-level developers would explore how security is integrated into day-to-day practice and when/where that knowledge was acquired, and seniors would explore approaches to closing perceived gaps, mentorship practices, and advice to make effective security education.

## **4.1.2 Participant Recruitment**

The participants were recruited majorly by carrying out email invitations to the software developers, requesting them to participate in the study. Besides direct



recruitment, there was the snowball sampling strategy [31] whereby the current participants recommended other possible participants. This method facilitated the study team tapping a greater number of participants at various levels of experience. The interviews were either in-person or online sessions, as each participant preferred and was available. This versatile setup made it more accessible and convenient to everyone involved.

The eligibility criteria was decided primarily based on the years of experience in software development because this was an important variable in the research study. In the case of novice developers, besides the limited years of experience, the participants must have completed at least three development projects, which meant that they would have adequate practical experience to reflect on their own coding practices.

Overall, the sample size of the researchers was 32 developers, including 3 senior developers, 9 mid-level developers and 20 novice developers. The results given in this section are as a result of the data obtained on these 32 participants. Participation was voluntary, and no financial incentives or compensation was given.

### **4.1.3 Pilot Testing**

Pilot test was conducted to identify the clarity, relevancy and practicability of the interview protocol, prior to the actual data collection. The pilot tests were taken consisting of three individuals with different levels of experience in the field of development and one of them was a peer of the researchers (a novice developer), the other was the research assistant, working on the project (intermediate developer), and the third was the senior developer (the senior software solution architect).

The pilot tests' aim was to ensure that the interview questions were not too difficult to answer, the duration of the sessions was appropriate and that there were no procedural pitfalls that might affect quality of the data collection. The protocol was slightly modified depending on the feedback of the pilot participants. Specifically, two more questions were added to pursue additional issues of the research topic, and a number of the existing questions were reformulated to enhance their lucidity and brevity.

Pilot test results showed that there were insignificant changes that had to be made and that no significant differences were observed between the pre and post-changes in data. The pilot test datas were therefore retained and incorporated in the analysis as they did not influence the validity or the comparability of the responses.

#### 4.1.4 Interview Protocol and Initialization

Participants were given a brief overview of the procedures at the beginning of each of the sessions. It also explained the direction in which the interview would take. This was to ensure transparency and consistency between sessions. General house-keeping (when, format, consent) was also discussed in the brief, and a comfortable, predictable environment that encourages open responses was established. The consent process was conducted before the formal discussion began to ensure that the participants understood and wanted to participate in the discussion.

All participants were given a comprehensive consent form beforehand, which explained the aim of the study, the type of their participation, the mechanism of data handling, and their rights as the study participants. A copy of the consent form was signed and sent back to the researchers by each participant prior to the start of his/her interview session.

In order to introduce some contextual information about the participants, a short demographic interview was introduced at the beginning of each session. These included age, highest level of education, gender, job title and self-reported security expertise in addition to the nature of projects they were typically engaged in. Gathering such demographic information was necessary to put the responses of the participants into perspective and to detect trends at different degrees of expertise.

The interviews were open-ended, and thus the participants were able to give in-depth information depending on their own knowledge and experience in their line of work. The interviews were divided into two sections. The first section was devoted to the general experiences of the participants with the identified attack vectors and how familiar they were with vulnerabilities and mitigation approaches in real-world development environments. The second section was dedicated to the role and the possible contribution of gamification systems to secure development practices. Interviews were fully documented and audiotaped to achieve precision later on during analysis.

All the interviews were about half an hour to an hour long. The answers given were very insightful on the struggles novice developers go through and how the implementation of secure coding practices can be a decisive factor in mitigating such vulnerabilities.

## 4.2 Procedure Analysis

The process of analysis was initiated by the verbatim transcription of all the audio recordings of the interviews with the 32 participants. All the identifiable information, including professional titles or study levels, was eliminated to provide confidentiality. Each of the participants got a certain alias according to their order of interviews and experience: novice developers received the names N1 to N20, mid-level developers M1 to M9, and senior developers S1 to S3. This ensured the analysis focused on the content of the interviews rather than the participants' identities.

The interview data from novice, mid-level, and senior developers were analyzed using an inductive thematic analysis approach. Thematic analysis helps in identifying patterns and insights that emerge from participants' responses [8]. This methodology enabled the research team to uncover nuanced details about developers' experiences, knowledge, and attitudes toward security practices and gamification. During this process, the Taguette tool was used to assist in refining and shortlisting the initial codes for better understanding of the transcriptions. As a result, 42 preliminary codes were obtained from senior developers, 27 codes from mid-level developers, and 10 codes from novice developers. These codes were then carefully revised, polished, and grouped into overarching categories and major themes through several consultations within the research team to ensure analytical rigour and internal consistency.

First of all, for novice developers, four fundamental themes were selected for analysis and all of them were further divided into two categories. The first theme, Security Awareness was divided into, Knowledge of Vulnerabilities and Security Considerations. The second theme, Learning and Development was categorized into, Learning Sources and Framework Dependency. Thirdly, Security Practices and Experience into Implementation Practices and Code Reviews along with Real-world Security Incidents. Lastly, the theme Attitudes and Motivation was segmented into Security Responsibility and Learning Preferences.

In the case of mid-level developers, there were six primary themes. The former Secure Development Practices was further divided into three categories, namely, Workflow and Automation, Dependency and Tools, and Security Knowledge and Learning Gaps. The theme also emphasized the necessity of systematic work processes and application of security tools, and difficulty of filling the knowledge gap in secure development. The second theme, Responsibility Sharing, was further broken into two themes, Team Roles and Organizational Constraints. This theme explored developers perception of their roles considering organizational structures and teams. A third general theme, Vulnerability Awareness, included Framework Dependency vs. Explicit Knowledge, Common vs. Advanced Vulnerabilities, Infrastructure-Level

Security and Authentication Practices. This theme demonstrated how the developers were mindful of the various vulnerabilities levels they faced with the focus on safe practices in both common and advanced scenarios. The remaining codes were directly categorized under the three main categories. For senior level developers, it was found that there were three prominent themes and these included Security Practices and Mentorship, Security-Specific Topics, and Gamification and Motivation. The initial theme, Security Practices and Mentorship was further categorized into Effect of Deadlines on Security, Use of Security Tools, Security Responsibility, and Mentorship and training. This theme discussed the process of prioritization of security and deadlines by senior developers, their implementation of security tools, and their mentoring of junior developers. Security-Specific Topics was the second theme, and the categories that were added to it were SQL Injection, XSS, DOM Clobbering, CDN Tampering, CSS Injection, NoSQL Injection, Clickjacking, and Learning Sources for Vulnerabilities. This theme demonstrated what particular vulnerabilities senior developers target and how they learn about such vulnerabilities. Finally, the Gamification and Motivation theme was divided into Gamified Mentoring, Mystery, Gifts and Individual vs. Team Focus. This theme was dedicated to the possibility of gamification in improving the security training and encouraging developers, demonstrating the way in which the elements of the game may be incorporated into the mentoring and security training courses.

Altogether, the thematic analysis showed that there are significant differences between the way developers with different levels of experience perceive and approach security. The middle-level developers were far more preoccupied with practice in operations, workflows and direct responsibilities, but senior developers combined a strategic oversight with mentorship and broader security consciousness. These findings provided a fundamental context to the thesis, as it showed the specific gaps in secure coding behaviors of novice, mid-level and senior developers and how the gaps could be remedied by certain interventions, particularly through gamification-based learning approaches, to establish more robust security behaviors throughout the first few years of a developers career.

### **4.3 Ethical Considerations**

A semi-structured format was used in the interviews, and three groups of participants were identified, including novice, mid-level, and senior developers. This method gave an opportunity to have a guided discussion and the freedom to have unexpected insights that came up in the discussions.

The research protocol was properly structured to guarantee the confidentiality and

anonymity of all participants, so no personally identifiable data (names, email addresses, organisational affiliations.) were collected. Rather, general demographic data were collected, such as the age range of participants, years of experience in development, and the nature of the organisation that they were working in (e.g., startup, medium-sized company, or a large enterprise).

Data gathered have been anonymised before being analysed and reported to avoid identification of any individual participants. Transcripts and the recordings of the interviews were saved in encrypted and confidential online storage, which could be accessed by the research team only. The audio recordings will be destroyed after the submission and publication of the thesis in line with the data privacy rules.

Participants were well informed on the voluntary nature of their participation. They were informed that they may leave the interview at any time without any reason and no consequences would be applied. Informed consent was taken beforehand, and all the participants were informed about the purpose of the research, the data handling procedures and their rights as participants.

## 4.4 Study Limitations

This research paper recognises some of the limitations that can influence the generalisability, richness, and breadth of the research results. Though data saturation was achieved when the sample had 32 respondents, the sample size was purposely skewed towards novice developers, with 20 respondents having one to two years of development experience. Although this concentration is justified by the main point of the study of the novice developers' security practices, it also led to the under-representation of the middle-level and senior developers. The other weakness is related to the limited population of participants. Since the research was conducted among software developers, security specialists such as penetration testers, security architects, or members of special security teams were not included.

The interviews were limited to 45 minutes each, which provided insufficient time for in-depth discussions of complex vulnerabilities. While generic attacks like SQL Injection (SQLi) and Cross-site Scripting (XSS) were covered adequately, more advanced issues such as Insecure Direct Object References (IDOR), DOM Clobbering and CSS Injection, received limited attention. Consequently, the research has reduced depth on advanced attack vectors and mitigation strategies.

# Chapter 5

## Interview Findings

The findings of the interviews with novice, mid-level, and senior developers are discussed in this chapter. The objective was to investigate their attitudes toward gamification as a method of improving security awareness. Significant variations in the security practices and expertise of developers with various levels of experience have been shown through the analysis.

### 5.1 Analysis of Novice Developer Responses

A detailed analysis based on the results found from the thematic analysis shown in Table 5.1 is presented in the following section.

Table 5.1: Thematic Analysis Results on Novice Developers

| Theme              | Category                     | Findings                                                                                                                                                                                                                           |
|--------------------|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Security Awareness | Knowledge of Vulnerabilities | Majority unfamiliar with common vulnerabilities (e.g., SQLi, XSS). Nearly all have never heard of OWASP Top 10. Heavy reliance on AI tools (ChatGPT, BlackBox AI) for code generation without understanding security implications. |
|                    | Security considerations      | Most do not consider security during coding. Their focus is mainly on feature completion and functionality over security measures.                                                                                                 |

*Continued on next page*

Table 5.1 – Thematic Analysis Results on Novice Developers

| Theme                                    | Category                                  | Findings                                                                                                                                                                                                                                                                                                                                                      |
|------------------------------------------|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Learning and Development</b>          | Learning sources                          | Most of them are self-taught through online courses, YouTube, and AI tools. Got minimal formal security training from universities. Security topics treated lightly in academic settings.                                                                                                                                                                     |
|                                          | Framework dependency                      | Heavy reliance on frameworks (MERN, Laravel, Django) and authentication libraries (Firebase, JWT) without understanding underlying security mechanisms.                                                                                                                                                                                                       |
| <b>Security Practices and Experience</b> | Implementation Practices and Code Reviews | About half of the participants follow basic security practices (password hashing, token usage, input validation). Often copy-paste code from AI without any verification. Security handled rarely rather than proactively. Limited participation in security-focused code reviews. Primary focus on functionality completion rather than security validation. |
|                                          | Real-world security incidents             | The majority never faced security issues. Limited real-world exposure to security vulnerabilities and their consequences.                                                                                                                                                                                                                                     |
| <b>Attitudes and Motivation</b>          | Security responsibility                   | All agree developers should be involved in security planning. Recognize the importance of protecting user data (credit cards, personal credentials).                                                                                                                                                                                                          |
|                                          | Learning preferences                      | Nearly all prefer game-based learning over traditional courses. Motivated by competition, leaderboards, and engagement. Considered traditional university lectures as theoretical and forgettable.                                                                                                                                                            |

### 5.1.1 Security Awareness and Knowledge Gaps

The fundamental security understanding of novice developers was found to be severely lacking. Developers frequently expressed unfamiliarity when they were questioned about common vulnerabilities like SQL Injection and Cross-Site Scripting. *“I have no idea, I have only heard of these vulnerabilities”*. This knowledge gap extended to industry-standard security frameworks, as nearly all participants were unaware of the OWASP Top 10 guidelines. N2’s response, *“I have no idea about OWASP Top 10”*, represents the widespread absence of basic security concepts among novice developers.

Educational institutions appear to contribute to these knowledge shortcomings. Several developers reported that security received minimal attention in their formal training, with one noting *“I have learned nothing (security related) from university, just PHP and SQL.”* Another explained that while university courses mention security topics, they are *“treated pretty lightly among students. It did not really help a lot.”* These responses suggest that security awareness in academic settings remains superficial and fails to establish lasting comprehension.

The situation is further complicated by widespread reliance on AI-generated code without critical evaluation. Multiple developers acknowledged learning through *“AI,” “ChatGPT,”* or *“BlackBox AI,”* with one admitting they *“I just used the codes ChatGPT gave me”* without considering security aspects. This approach to development, copying code without understanding its security implications, accelerates feature delivery while simultaneously creating dangerous blind spots in developers’ security awareness.

### 5.1.2 Learning Sources and Framework Dependency

Novice developers acquire technical skills primarily through self-directed online learning, including courses, YouTube tutorials, and AI assistance, with minimal structured security training. This method of instruction results in developers who are capable of implementing features but lack the principles of security. University education, when mentioned, was characterized as providing only superficial coverage of security concepts that students find boring and struggle to focus on, quickly forgetting the material.

Their coding knowledge is heavily framework-dependent. Developers consistently rely on modern frameworks (MERN, Laravel, Django) and authentication libraries (Firebase) to handle security aspects automatically. While these tools provide important protections, developers demonstrate limited understanding of the underly-



ing security mechanisms. This creates a problematic pattern. When frameworks offer security features, for instance, developers can install them, but they lack the necessary knowledge to determine when framework protections are insufficient or how to separately implement security measures.

The lack of knowledge prevents novice developers from analyzing security threats critically, while the pressure to complete tasks and meet project milestones within the deadline encourages shortcuts and neglect of security. Use of overly simplified frameworks helps this cycle by allowing novice developers to deliver their project quickly without maintaining secure codes in their project. Together, these three factors produce developers who view security as an afterthought, perpetuating the gap between academic learning and real-world secure development.

### 5.1.3 Security Practices and Real-World Experience

#### Implementation Practices and Code Reviews

When developers implement security measures, their approach tends to be superficial and convention-driven. Authentication mechanisms like “*Firebase*,” “*JWT Token*”, “*2 Factor Authentication*” and “*Password Hashing*” are commonly used, but interviews revealed these are typically default choices rather than informed decisions. N2’s statement that “*The main target is to complete all functionalities, security is not prioritized at all*” encapsulates the broader pattern of treating security as secondary to functionality. Code review participation, a critical quality control mechanism, is missing among novice developers. Responses ranged from “*Never*” to “*Sometimes*” to “*Not often*” with N3 noting they only participated “*If I am confused about the code, then yes, I do code reviews.*” This inconsistent engagement in code review processes means security vulnerabilities in novice-written code may persist undetected through development stages.

#### Real-World Security Experience

Most novice developers reported never encountering security-related issues in their projects. Rather than indicating effective security practices, this reflects limited exposure to production environments and real-world threat scenarios. Several developers acknowledged their projects “*were not deployed out in the market*” or remained academic exercises, meaning they have never experienced the consequences of security vulnerabilities in operational systems. As the main objective of developing these projects was to complete the required functionality, novice developers never really targeted further secured deployment processes.

## 5.1.4 Attitudes Toward Security and Learning

### Recognition of Security Importance

Despite substantial knowledge deficiencies, all novice developers agreed that developers should participate in security planning. Their justifications demonstrated awareness of security's importance, citing concerns about *"data breaches,"* protecting *"credit card info, phone number,"* preventing *"unauthorized access,"* and ensuring *"users safer websites."* N1 referenced *"Bangladesh Bank security breaches,"* indicating awareness of local security incidents and their consequences.

However, a significant gap exists between theoretical acknowledgment and practical implementation. Developers identified several contributing factors: time pressures *"Due to time constraints, developers do not implement their theoretical security knowledge"*, prioritization of functionality over security, and absence of structured guidance for implementing security measures systematically.

### Preference for Gamified Learning

Responses to gamified learning approaches were overwhelmingly favorable, with nearly all participants expressing preference for game-based learning over traditional educational methods. N3 emphasized that games would be *"interesting"* and create engagement through competition, stating *"It will create a competitive environment so motivation to learn will increase."* N9 explained the retention problem with traditional education: *"During traditional methods such as university lectures, students learn theoretical concepts and forget everything after exams,"* highlighting how examination-focused learning fails to produce lasting knowledge.

N10 emphasized the ability of games to bridge the theory-practice gap, adding *"Theoretical knowledge can be gained from traditional courses, but it is difficult to implement in real life. So the game would be more effective."* This resolves the ongoing imbalance that several developers found between professional practice and classroom instruction.

Not all responses were positive in terms of the gamification process. One developer mentioned that gamification might only sustain engagement *"For a short period of time,"* raising concerns about long-term effectiveness. Another advocated for *"Combination of both options. Universities should include such courses"* alongside gamified approaches, suggesting complementary rather than competing educational methods.

The strong enthusiasm for gamification, combined with demonstrated knowledge

gaps and minimal engagement with traditional learning, provides strong evidence that game-based security education is an effective solution for the novice developers.

## 5.2 Analysis of Mid-Level Developer Responses

A detailed analysis based on the results from the thematic analysis of the Mid-level developers shown in Table 5.2 is presented in the following section.

Table 5.2: Thematic Analysis Results on Mid-level Developers

| Theme                        | Category                           | Findings                                                                                                                                                                                                                                                                                                                                           |
|------------------------------|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Project Experience           | Significant projects               | Developers worked on diverse projects: government databases (security-critical, large-scale), fintech systems (biometric verification, Mobile Financial Services), blockchain-based energy certificates, and rapid delivery projects with little QA. Many highlighted a theory–practice gap and the importance of security in real-world projects. |
|                              | Workflow & automation              | Some teams used automated pipelines (Jenkins, Spring boots) and dependency checks, while others relied heavily on manual testing or ignored security at setup. Security was often considered secondary to functionality.                                                                                                                           |
|                              | Dependency & tools                 | Mixed awareness: some prioritized secure dependencies and patching; others ignored deprecated packages. Security tools were inconsistently used.                                                                                                                                                                                                   |
| Secure Development Practices | Security knowledge & learning gaps | Many developers reported mentorship gaps, reliance on AI tools, or learning by doing. Security was often not prioritized in early development.                                                                                                                                                                                                     |

*Continued on next page*

Table 5.2 – Thematic Analysis Results on Mid-level Developers

| Theme                          | Category                                    | Findings                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------|---------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Responsibility Sharing</b>  | Team roles                                  | Security responsibility was often senior- or architect-led. Medium developers overlooked security but were corrected in code reviews. In some cases, developers outsourced security to testers or specialists.                                                                                                                                             |
|                                | Organizational constraints                  | Time pressure and client requirements often dictated security practices. Some companies avoided certain tools (e.g., GitHub banned for safety) to reduce risks.                                                                                                                                                                                            |
| <b>Credentials Management</b>  | Handling credentials                        | Practices included Vaults, .env files, runtime injection, Jenkins credentials, environment variables, and hashed passwords. Some developers emphasized avoiding GitHub leaks.                                                                                                                                                                              |
| <b>Vulnerability Awareness</b> | Framework dependency vs. explicit knowledge | Developers relied heavily on framework-provided protections for SQLi and XSS but lacked foundational understanding of underlying concepts (e.g., query parameterization). Many admitted knowing vulnerabilities only theoretically without practical implementation knowledge.                                                                             |
|                                | Common vs. advanced vulnerabilities         | Strong awareness of SQLi, XSS, and IDOR (emphasized UUID insufficiency and need for server-side access control/authorization checks, primarily using RBAC). Severe knowledge gaps for advanced threats: DOM Clobbering, CSS Injection, CDN Tampering, and NoSQL Injection. Most developers had limited to no understanding of these niche vulnerabilities. |

*Continued on next page*

Table 5.2 – Thematic Analysis Results on Mid-level Developers

| Theme                        | Category                      | Findings                                                                                                                                                                                                                                                |
|------------------------------|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | Infrastructure-level security | CDN security and SRI were largely considered DevOps responsibilities. Most developers showed minimal engagement or deferred to infrastructure teams, revealing a blind spot in end-to-end security awareness.                                           |
|                              | Authentication practices      | Implemented password hashing (mostly bcrypt) and access control but choices were often convention-driven rather than informed. Some developers couldn't explain why specific algorithms were chosen, indicating passive adoption of security practices. |
| <b>Learning and Training</b> | Gamification and awareness    | Participants agreed gamified learning quizzes, leaderboards, and competitions were more engaging than traditional workshops. However, concerns were raised about practical relevance and avoiding real-world misuse of game examples.                   |

### 5.2.1 Project Experience and Security Awareness

Mid-level developers often faced the challenge of transitioning from university-level projects to real-world applications, which revealed significant gaps in their understanding of security. Many of them worked on large-scale government, fintech, and blockchain-based projects, which required handling sensitive data and ensuring robust security measures. M2 shared, *“Security is a huge priority”* in their project, highlighting the growing importance of securing sensitive data in real-world applications. While university training provided foundational knowledge, it was only through hands-on experience that they truly grasped the significance of security practices. For example, M2 mentioned that his experience in government projects involved significant database management, where *“confidential client data has to be handled”* properly to avoid legal consequences. Moreover, M9 quoted: *“Finally when all our modules are done, our main coordinator says we will release an Alpha application for certain users. After good feedback we move to the production*

*deadline.*” further shows staged deployment with real users before full production, a security-conscious practice that allows vulnerability discovery in controlled settings. This contrasts with other participants who didn’t mention staged rollouts. M9 also mentions : *“When we are developing a project to aim for a certain market place, these security vulnerabilities need to be considered and good practices need to be enforced. That is when we need to study ourselves. When we enter a company, the company structure teaches us their security policies.”* This observation underscores the gap between theoretical knowledge and practical security needs in the industry. Developers realized that security is not a secondary concern but a fundamental aspect of development, especially when handling critical data. As M2 mentioned, *“Security concepts learned at university were not given as much importance,”* but in a job setting, security becomes non-negotiable.

### 5.2.2 Secure Development Practice

The use of security tools and integration into development pipelines was inconsistent across teams. While some developers adopted automated pipeline tools like Jenkins or Spring Boot and dependency checking to streamline their workflows, others were more reliant on manual processes, often neglecting security during setup. M1 added that secured codes were not the client’s main priority reflecting the tendency for security to be deprioritized, especially when deadlines are tight or functionality is the main focus. Moreover, M1 mentioned that developers are not security concerned while developing initially, when security is a concern, a separate security team is hired to test for security after development. M9 validates this by saying *“When we develop a feature, our first priority is to complete the feature. Only after that is the layer of abstractions and security policies added”*.

Some developers highlighted their approach to ensuring security during setup. For instance, M3 mentioned using reliable dependency versions to mitigate vulnerabilities, but he noted that not all projects implemented security tools in the same manner. He added, *“Not all projects contain deployment pipelines,”* which led to inconsistent security practices. This variation in practice, especially the reliance on manual testing or ignoring deprecated packages, shows that security is sometimes an afterthought or an added step rather than integrated from the start.

### 5.2.3 Responsibility Sharing

Security responsibility within teams was typically shared, but the extent of involvement varied based on team roles. Senior developers, devops engineers, and security

specialists played key roles in setting security standards, while novice developers were often more focused on code functionality than security practices. M2 noted, *“Security issues are not prioritized during development, mainly novice developers overlook security,”* but these issues were usually addressed in code reviews. *“If insecure code is overlooked during these reviews and is used in production, it is immediately fixed,”* emphasizing the importance of peer review and team collaboration in maintaining security standards. M9’s explanation further reveals role specialization that creates knowledge boundaries. This shows how organizational structure can limit security tool exposure even for mid-level developers.

Mentorship played a significant role in spreading security awareness, especially for novice developers. Developers noted that novice team members often lacked the practical knowledge to implement security effectively. As M1 pointed out, *“Due to time constraints, developers do not implement their theoretical security knowledge in coding,”* suggesting that ongoing mentoring and real-world practice are essential for bridging this gap. In some teams, security specialists took the lead, but in others, the lack of a dedicated security expert meant that developers were responsible for self-educating on security best practices.

#### 5.2.4 Credential Management and Security Practices

For credential management, developers employed various strategies, such as storing database credentials and API keys in secure vaults and using environment variables to prevent sensitive data from being exposed. *“We try to keep API keys, tokens, and database credentials in a secure place,”* said M2, highlighting the importance of proper data storage practices. Despite these practices, not all developers followed such security measures. For example, M6 admitted that *“I don’t know what kind of packages are being downloaded”* during project setup, indicating a lack of focus on securing dependencies. Some developers also emphasized encrypting credentials and using secret scanning tools during deployment. M4 further highlighted security concerns with code repository platforms, stating that they do not trust GitHub’s public cloud because of risks related to data leakage and conflicts. Instead, they prefer a self-hosted or private GitLab instance, which allows greater control over both the environment and security. M6 stressed that uploading environment variables should be avoided at all costs. In contrast, M3 prioritized secure dependency selection and regular updates, underscoring the importance of managing dependencies for security.

### 5.2.5 Vulnerability Awareness and Security Knowledge

Developers recognized the importance of preventing vulnerabilities such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and Insecure Direct Object References (IDOR), though there were varying levels of awareness and practices. The responses revealed a clear divide between framework-reliant security and explicit security knowledge, with many developers depending heavily on modern frameworks to handle security automatically.

#### Framework Dependency vs. Explicit Knowledge

For common vulnerabilities like SQLi and XSS, developers exhibited strong reliance on frameworks. M2 explained, *“Modern day frameworks can mitigate this vulnerability (SQLi) and internally parameterize the queries, thus the risk is mitigated. During coding, people don’t inherently keep this vulnerability in mind. They mostly know it in theory.”* M3 echoed this sentiment: *“SQL injection is one of the most known attacks, so current frameworks already mitigate it.”* Similarly, for XSS, M2 noted, *“Firebase contains built-in scripting, React has content verification which mitigates this issue (XSS). Developers don’t need to handle it as tools handle it.”*

However, this framework dependency revealed a concerning pattern. M5 candidly admitted, *“I know SQL injection but I am not familiar with the term Query Parameterization,”* highlighting a gap between theoretical awareness and practical understanding. M6 expressed similar uncertainty regarding XSS mitigation tools, simply stating *“No idea”* when asked about HTML sanitization and output encoding. This suggests that while frameworks provide protection, many developers lack the foundational knowledge to implement security measures independently or recognize when framework protections might be insufficient.

#### The Depth Divide: Common vs. Advanced Vulnerabilities

A striking knowledge gap emerged between well-known vulnerabilities and more advanced attack vectors. While developers were generally familiar with SQLi and XSS, understanding deteriorated sharply when discussing DOM Clobbering, CSS Injection, and CDN Tampering.

M6 acknowledged the existence of DOM Clobbering but dismissed it as *“Less risk for modern systems as client side and server side validation is being done.”* However, M3 and M5 both admitted uncertainty, as M5 stated that *“Not sure”* and only noting that *“DOM manipulation is done using JS.”* This partial awareness suggests



that many developers recognize the theoretical existence of such attacks without understanding their practical implications.

The knowledge gap was even more pronounced for CSS Injection. M2 deferred responsibility entirely: *“Frontend developers would know about this vulnerability. From the backend, we ensure validation occurs and bad data is filtered.”* M5 confessed, *“He has no idea about CSS Injection. He just knows that JS injection can send unauthorized access.”* Only M4 demonstrated deeper understanding, noting that *“CSS injection is harder”* because it is more difficult to detect compared to JavaScript Injection, and emphasized that *“CSP sometimes works, but if we use inline CSS, CSS injection can be avoided from that perspective.”*

NoSQL Injection revealed the most significant knowledge gap overall. Among those who had some awareness, opinions differed on which injection type posed the greatest risk. M2 identified *“Timing based injection”* as *“most commonly overlooked,”* while M3 focused on *“syntax injection,”* and M4 prioritized *“operator injection”* as the most critical concern. This scattered understanding and lack of consensus suggests that as NoSQL databases gain popularity, security education has not kept pace with adoption, leaving developers vulnerable to attacks they may not even recognize.

## Infrastructure-Level Security

When discussions moved to infrastructure-level concerns like CDN Tampering and Subresource Integrity (SRI), most developers showed limited engagement or deferred responsibility to senior roles. M2 candidly stated, *“In general, Junior and mid-level developers do not handle it, senior developers do. People who prioritize security and build the infrastructure keep this in mind for higher security projects.”* This perspective revealed a common pattern where infrastructure-level security is perceived as outside the scope of typical development work. Some developers expressed passive trust in external providers. M3 explained his reliance on CDN providers: *“If you can access something with CDN other people also can. If there are vulnerabilities in such CDN providers it will be fixed immediately. That is why I trust it.”* This assumption that security would be handled by third parties reflects a potential blind spot in understanding shared responsibility models. Others demonstrated confusion about different types of infrastructure threats. M7, stating *“HTTPS is not enough as DDoS (Distributed Denial-of-Service) can occur,”* mixing concerns about denial-of-service attacks with content integrity issues. This confusion highlights the difficulty many developers face in distinguishing between different infrastructure-level security concerns. In contrast, M5 demonstrated comprehensive understanding of SRI mechanisms, explaining that it *“compares it with the provided hash. And if it does not match, the browser basically blocks the resource.”* He further noted the limita-

tions of HTTPS alone, explaining that *“HTTPS actually creates the channel to pass the compromised content”* if the CDN itself is compromised, showing awareness that encrypted channels don’t guarantee content integrity.

## **Authentication and Access Control: Practical but Uninformed**

Interestingly, while developers implemented authentication measures, their choices were often driven by convention rather than informed decision-making. When asked about password hashing algorithms, M2, M6, and M7 all used Bcrypt, but their reasoning revealed passive adoption. M6 stated that *“it came as a default and everyone used it and was comfortable with it. I know how to use it, that’s why I use it.”* M1 admitted to using Bcrypt *“because someone else took that decision, I did not.”* Only M5 demonstrated informed selection, choosing Argon2 specifically because it *“can basically mitigate both GPU related attacks”* and is *“more time consuming, and actually it’s hard work, it’s more safe.”* For IDOR protection, developers showed better awareness of the limitations of UUID alone. M2 emphasized, *“UUID is not sufficient, access control is required. UUID can be acquired and an attacker can gain access to privileged information.”* M6 provided detailed insight into her approach: *“Some things are not passed in the URL parameter, these informations are taken from logged in information, or session. UUID/Hash is not used, backend check is done.”* However, she acknowledged that the use of UUID or hashed IDs depends on project requirements and architectural decisions. She emphasized the importance of comprehensive security checks, noting that *“Developers have to think about all cases and handle this vulnerability. Sometimes they can miss a case and then an IDOR attack occurs. I myself fixed someone else’s code.”* This practical experience highlighted that regardless of the identifier strategy chosen, proper authorization checks remain essential to prevent IDOR vulnerabilities.

### **5.2.6 Gamification and Security Training**

Developers generally prefer gamified security training over traditional methods, finding it more engaging and effective, especially for novice learners. M2 stated, *“Security issues are overlooked in case of training, it is only addressed when an issue has already occurred,”* suggesting that a proactive approach through gamification could prevent such oversight. *“For prevention games are a better option,”* he added, advocating for game-based learning to reinforce security knowledge before vulnerabilities arise. M6 emphasized the importance of depth in gamified learning: *“If games are designed in a way to ensure I am learning the concept to the core, I would appreciate playing those games. I would get entertainment and I would learn those concepts.”*

M9 provides support for gamification: *“I always love gaming experience from those types of workshops. Attack scenarios are given and then we have to solve them from our side, I think this will be a more efficient way to learn.”*

However, the effectiveness of gamification was not universally accepted. M3 raised concerns about gamified learning, noting, *“For novice developers, it won’t be beneficial in the long term if short questions or snippets are given on a topic. However, they might find the topic interesting and later want to do more research. So along with the quizzes, add resources on the topic.”* M1 was even more skeptical, stating bluntly, *“If someone has interest in the topic, they would want to learn. In this case, gamification would be a good option. If he has no interest, nothing would work,”* suggesting that gamification alone cannot overcome fundamental lack of motivation. Gamified elements like leaderboards, badges, and team challenges were seen as particularly motivating tools. M6 highlighted the competitive aspect: *“If 5 of us are given a challenge to write the best coding solution, we can then be assessed based on that. Coders have the ego that they know best so we can use that component in a positive way to make them learn security aspects.”* He further noted, *“If I can showcase my skills and badges (for example in Codeforce), I would be more encouraged to use it.”* M4 agreed, adding, *“Yes it will motivate players as it will create a competitive environment among many players. If these badges can be converted into something valuable, users will be motivated.”* However, M5 offered a more conditional perspective on gamification design, cautioning that effectiveness *“depends on the design of the game,”* and emphasized practical concerns: *“Make the game in such a way that attackers cannot take advantage and use it in real life for the wrong purpose. Not everyone will learn with good intentions therefore be careful. You can use pseudo code or a gaming scenario or controlled environment.”* This concern about potential misuse was critical, as he warned, *“Make sure no attacks can be done IRL by playing the game.”*

### 5.3 Analysis of Senior Developer Responses

A detailed analysis based on the results of the senior developers from the findings of the thematic analysis is shown in Table 5.3 and presented in the following section.

Table 5.3: Thematic Analysis Results on Senior Developers

| Theme                                    | Category                        | Findings                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------------------------------------|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Security Practices and Mentorship</b> | Impact of deadlines on security | Senior developers recognize security practices must be maintained despite tight deadlines. Some negotiate deadlines to ensure security (e.g., payment-related features like Robi scratch card). Non-critical apps like blogs may have relaxed security requirements. Novice developers may skip security aspects during urgent deadlines, relying on code reviews. Frameworks (Laravel, Django) mitigate basic security risks, but hands-on practice and guidance from seniors are needed for efficiency. |
|                                          | Security tool usage             | Security tools are part of the workflow and may slow progress slightly, but they are not frustrating. Cybersecurity teams provide vulnerability reports, recommend patches, and discourage unsafe server versions. Poorly optimized queries can affect security but not necessarily the tools.                                                                                                                                                                                                            |
|                                          | Security responsibility         | Security is shared across the team. Senior developers, leads, and security specialists guide practices, but all senior developers have responsibility. In small or mid-range companies, security may be less formal due to cost. In larger companies, DevOps, and security teams collaborate. Penetration testing is handled by dedicated teams in renowned companies.                                                                                                                                    |

*Continued on next page*

Table 5.3 – Thematic Analysis Results on Senior Developers

| Theme                    | Category              | Findings                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Security-Specific Topics | Mentorship & training | Mentorship is crucial for awareness of input validation, secure coding, and confidential data handling. Seniors conduct grooming sessions, provide feedback, encourage learning through courses and certifications, and foster a culture of knowledge sharing. Novices learn through guided practice, code reviews, and gamified platforms like HackTheBox to simulate vulnerabilities (SQLi, XSS). |
|                          | SQL Injection         | Most frameworks mitigate SQLi, but novice developers sometimes write vulnerable code. Industry-grade infrastructure (API gateway, framework validation) adds protection. Practice and repetition are essential for muscle memory.                                                                                                                                                                   |
|                          | XSS                   | Novice developers often lack practical knowledge. Many have only theoretical awareness. Black-box testing is mostly handled by QA teams; in development, it may complicate workflows.                                                                                                                                                                                                               |
|                          | DOM Clobbering        | Senior developers have limited awareness. Predictable global variables are risky. Patch and disclosure practices depend on experience.                                                                                                                                                                                                                                                              |
|                          | CDN Tampering         | Use of third-party CDNs (e.g., Cloudflare) is preferred over self-hosting due to security specialization and cost. Security relies on provider guidelines and reliability.                                                                                                                                                                                                                          |

*Continued on next page*

Table 5.3 – Thematic Analysis Results on Senior Developers

| Theme                              | Category                             | Findings                                                                                                                                                                                          |
|------------------------------------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                    | CSS Injection                        | Backend-focused senior developers have limited exposure. Best practices include proper tag closure, sanitization, avoiding third-party stylesheets, minimal inline CSS, and secure data handling. |
|                                    | NoSQL Injection                      | Novice developers rarely use mitigation libraries (e.g., <code>mongoSanitize()</code> ). Awareness is limited when using NoSQL databases.                                                         |
|                                    | Clickjacking                         | Server-side mitigations are emphasized; client-side methods are limited due to browser control. Headers like CORS are often sufficient.                                                           |
|                                    | Learning sources for vulnerabilities | Basic knowledge often comes from university, but in-depth understanding develops during industry experience. Continuous learning is needed as vulnerabilities evolve (e.g., OWASP Top 10).        |
|                                    | Gamified mentoring                   | Mixed opinions: some senior developers prefer traditional responsibility-based learning; others find gamified approaches (quizzes, leaderboards) engaging and helpful for retention.              |
| <b>Gamification and Motivation</b> | Mystery Gifts                        | Suggested rewards include skills or tools that provide market value (JWT, Bcrypt, API security, Docker, CI/CD pipelines). Flexibility is advised, as framework relevance varies.                  |

*Continued on next page*

Table 5.3 – Thematic Analysis Results on Senior Developers

| Theme | Category                  | Findings                                                                                                                                                                                                  |
|-------|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|       | Individual vs. team focus | Individual progress is preferred for initial learning; team collaboration can follow after feedback. Team-based security learning encourages holistic awareness but may be harder to implement initially. |

### 5.3.1 Security Practices and Mentorship

#### Impact of Deadlines on Security Practices

The senior developers interviewed highlighted a critical tension between tight deadlines and the maintenance of robust security practices. Senior developers emphasized that security must remain a priority regardless of deadlines. As S1 noted, *“Security practices are unspoken rules to follow even if deadlines are urgent. If the deadline is near, you have to negotiate that it is not possible in this many days as security cannot be disregarded.”* This indicates that, while deadlines may push senior developers to consider shortcuts, security is a fundamental requirement that cannot be compromised.

However, in some scenarios, developers working under tight deadlines may skip security checks, relying on other teams (e.g., QA or code reviews) to mitigate risks. S2 shared that novice developers, in particular, may skip security measures when pressed for time, emphasizing the need for continuous learning and awareness, particularly through frameworks like Laravel or Django that mitigate basic security risks. The interviewees also observed that frameworks automatically address certain security vulnerabilities (e.g., SQL Injection), but novice developers may lack the foundational knowledge to avoid mistakes manually. In essence, frameworks help alleviate risks but hands-on knowledge and guidance are necessary for efficiency.

#### Security Tool Usage and Workflow Impact

Most senior developers acknowledged that security tools are integral to the development workflow, although they can slow down progress. S1 remarked, *“Security is not an obstacle, it is a part of the development pipeline. It may slow you down but it does not frustrate me as it is part of work.”* The use of specialized security tools, provided by dedicated cybersecurity teams, allows senior developers to identify and

fix vulnerabilities. These tools help enforce security measures and prevent security breaches, even if they slightly delay the development process. Notably, senior developers also discussed how security tools, such as automated testing and vulnerability detection, are essential in maintaining security without adding undue complexity to the workflow.

## Distributed Security Responsibility

In many companies, senior developers handle most of the project’s security and deployment tasks before handing it over to junior developers. S3 explained, *“Practices depend on the company, but in most cases, senior developers do not let junior developers handle initial project setups. All aspects of project setup related to security — such as writing deployment scripts, creating virtual machines (VMs), configuring AWS security, and integrating security libraries — are handled by senior developers. Only after these foundational security measures are in place is the project handed over to juniors. For day-to-day tasks, most teams rely on authentication libraries provided by frameworks to handle authentication, CSRF tokens, password resets, and similar security concerns, as these libraries are well-established and raw coding for such tasks is discouraged.”* While the ideal of a collective security culture, where all team members share responsibility, exists in theory, practical constraints often prevent full realization. Many small to mid-sized companies in Bangladesh cannot afford dedicated security teams, forcing senior developers to carry the bulk of security responsibilities. This creates a two-tier system: larger organizations can implement comprehensive security measures, including QA validation, dedicated security teams, and DevOps optimization, whereas smaller organizations rely on senior developers who may lack specialized training. As a result, novice developers often get stuck handling project functionality, unable to contribute to critical components. Restricting junior developers from implementing authentication systems reflects pragmatic risk management, helping ensure short-term security outcomes. However, it also limits learning opportunities for juniors and slows the development of broader security expertise within teams. Moreover, after handling this heavy workload, senior developers are also responsible for code reviews and fixing security or functionality mistakes made by novice developers, as noted by S1. By centralizing initial security responsibilities among senior developers, organizations reduce immediate risks but risk constraining the long-term growth of distributed security capabilities.



## Mentorship and Training in Security Awareness

Mentorship emerged as a critical factor in spreading security awareness, particularly for novice developers. Senior developers emphasized the importance of providing structured guidance through hands-on training, grooming sessions, and real-world examples. Developer S2 shared, *“Mentorship is necessary for all aspects to improve the learning curve, they emphasize security and encourage novice developers to consider security measures. They emphasize security and encourage novice developers to consider security measures.”*

A common theme across responses was that novice developers often lack awareness of security vulnerabilities and need to be introduced to these concepts through active mentoring and real-world practice. Novice developers are encouraged to use platforms like HackTheBox to simulate vulnerabilities such as SQL Injection (SQLi) and Cross-Site Scripting (XSS), as a way of engaging them in security practices in a gamified environment.

### 5.3.2 Security-Specific Topics

#### Vulnerabilities and Their Impact on Development

##### SQL Injection (SQLi)

Senior senior developers acknowledged SQLi as one of the main vulnerabilities in industry-grade applications. Even though most frameworks today mitigate SQLi vulnerabilities through validation and ORM features, many senior developers still end up writing vulnerable code. S1 emphasized that infrastructure-level protections such as API gateways and framework-level validation help prevent exploitation even when insecure code is introduced. Another developer (S2) highlighted that avoiding SQLi requires repeated practice until secure coding becomes *“muscle memory.”* This shows that frameworks act as a safety net, but lack of consistent hands-on awareness among senior developers keeps the vulnerability relevant.

##### Cross-Site Scripting (XSS)

XSS was identified as another common issue, particularly among novice developers. Seniors observed that many novice developers do not fully understand how XSS works. Some skip security steps in the frontend, implementing features incorrectly and creating risks. Another pointed out that many novices have only theoretical knowledge but fail to apply it in practice. While black-box testing is often used, seniors noted it is usually handled by QA teams rather than senior developers. They

also recognized tools such as Postman’s Newman library that can assist in testing API functionalities. This highlights that developer-side awareness and direct testing are often limited, leaving responsibility to quality assurance.

### **DOM Clobbering**

Awareness of DOM Clobbering was reported as very low among senior developers. One senior (S3) explained that only a small number of projects had actually considered such risks. Another (S2) stressed that *“issues like predictable global variables could easily be exploited by attackers, pointing to gaps in secure coding guidelines and developer training.”*

### **CDN Tampering**

On the issue of CDN usage, senior senior developers consistently recommended reliance on established third-party CDN providers, such as Cloudflare, over self-hosting. Their reasoning was that third-party CDNs are more specialized in security and maintaining their systems. By contrast, self-hosting would require dedicated teams and high maintenance costs, which are often not feasible. The clear preference was to depend on reputable external providers while following their guidelines.

### **CSS Injection**

Insights here were mixed. One senior (S1) stated, *“I have not personally encountered CSS injection issues in backend work.”* Another (S2) provided specific best practices, including closing all tags properly, sanitizing data, avoiding too much inline CSS, and not importing stylesheets from untrusted third parties. These responses suggest that awareness varies, but standard coding practices were seen as effective defenses.

### **NoSQL Injection (NoSQLi)**

When NoSQLi was asked about , senior developers mentioned that many novice developers do not typically use libraries like mongoSanitize. While some lacked direct experience with NoSQL databases, they stressed that issues arise when senior developers fail to keep databases updated with the latest patches, as outdated versions are often publicly exploitable. Given that managed database services take care of a large portion of the security automatically, they were cited as safer alternatives. Overall, Seniors noticed that developers often apply inconsistent or outdated methods to prevent NoSQL Injection (NoSQLi).

### **Clickjacking**

On clickjacking, senior developers agreed that mitigation should primarily be handled server-side through browser security headers and policies. They noted that client-side measures, such as frame-busting code, could easily be bypassed. How-

ever, they emphasized the importance of teaching novices about these client-side mitigations for awareness, even if headers are generally sufficient to prevent this attack.

### Learning about Vulnerabilities

When asked about how they first came across vulnerabilities such as SQLi and XSS, seniors gave varied responses. Some claimed they learned about these during university courses and research projects, while others only gained practical knowledge after entering the industry. They emphasized that because vulnerabilities evolve frequently, knowledge from university often needs to be updated through continuous learning in professional environments.

### 5.3.3 Gamification and Motivation in Security Training

When it comes to the use of gamification for security training, there were mixed opinions. Some senior developers felt that traditional responsibility-based learning was more effective. For example, one senior developer (S1) argued, *“Personally I would not like this approach. At the end of the day, security is part of the developer’s responsibility. Senior developers are recruited based on their skills, so it is not like we always have to pressurize them to train.”* In contrast, other senior developers (S2 S3 ) saw value in gamified approaches, such as using leaderboards, badges, and level-based tasks, to enhance engagement and retention, stating that *“through activities, people would learn and retain the information more.”*

There was consensus, however, on the importance of flexibility in the gamified approach. Senior senior developers suggested using *“mystery gifts”* or rewards that could enhance the marketability of the senior developers’ skills, such as tools for API security, Docker, and CI/CD pipelines. These rewards would help make learning more appealing and relevant to the senior developers’ career progression. One developer (S2) recommended, *“Make them familiarized with terms such as Authorization so that they can further research,”* acknowledging the importance of tailoring rewards to the specific framework or toolset being used.

## 5.4 Discussion: The Developer Security Lifecycle

### 5.4.1 From Novice to Senior: The Professional Journey

The shift between a novice and a senior developer is a crucial change in the perception of security as well as its implementation in professional practice. New developers come to the industry with a high level of knowledge deficiencies and show little to no experience in understanding the common security vulnerabilities or the security frameworks that the industry operates on. They are mainly focused on meeting the functional requirements and security is considered as an after thought but not a prerequisite. This trend can be explained by the fact that in education, issues of security are often briefly covered, forgotten after exams, and excessive exposure to AI-generated code is unconverted regarding its security consequences. Universities usually do not make security a part of instructing fundamental concepts of development, but instead have students take additional security courses, which may or may not be mandatory, giving the impression that security is an option, but not a requirement of software development.

Mid level developers are in an intermediate position between theory and practice, where theory starts to face practice. Their efforts in government databases, fintech solutions, and blockchain are vulnerable to a condition that security lapses have a tangible cost. The stage demonstrates the gap between theory and practice that university education is unable to bridge. Although mid-level developers apply security practices including password hashing and access control, they do not always consider their application and are usually influenced by conventions. They highly depend on structures to manage security automatically with little understanding of how mechanisms work. The functional requirement to meet deadlines often leaves security concerns to be addressed later in the development cycle, and individual security groups or code inspections often become the main quality control tool instead of active security integration taking place during the new functionality development.

Senior developers have a vastly different attitude to security and regard it as an integral part of professional responsibility and not a nice-to-have addition. When security requirements are not achievable, they bargain deadlines, and they understand that technical debt and liability are possible outcomes of cut corners. They do it based on the experience of dealing with security incidents in real-life situations and the knowledge of the repercussions of the vulnerabilities in production settings. Experienced senior developers perform a mentoring role, initiating grooming sessions and delivering formal advice to novice and mid-level developers on the way to become security conscious. They make important security-related decisions when configuring the initial projects, such as deployment scripts and configuration of vir-

tual machines as well as integration of security libraries and then hand over projects to juniors on the team. This security-concentrated design among senior-level developers brings about a state of organizational stability but also exposes structural difficulties in creation of distributed security skills among development teams.

The transition of a new employee to an experienced employee usually takes a number of years of practical work in the field of production systems where security weaknesses have direct implications. University education equips the basics, however, without any permanence of the understanding that is required in safe development practices. The actual learning is achieved by being exposed to security incidents, reviewing codes which reveal patterns which are vulnerable. The mentoring of the experienced developers and through practice and exposure to security concerns, secure coding is not about deliberate action but automatic action. This long shelf life causes a vulnerability gap in the novice and middle levels two stages of developer development in which developers have enough technical capability to develop working systems but lack the technical security awareness to detect and avoid more vulnerabilities.

## **5.4.2 Security Awareness Across Experience Levels**

The depth and nature of security knowledge differs substantially across experience levels, revealing patterns that extend beyond simple accumulation of information. Novice developers demonstrate awareness of security terminology without understanding implementation details. They did not recognize terms like SQL Injection and Cross-Site Scripting when mentioned. Their reliance on frameworks creates a false sense of security, assuming automated protections eliminate the need for security knowledge rather than recognizing frameworks as tools that require informed usage.

Mid-level developers exhibit a framework-dependent security model where protection relies almost entirely on the automatic safeguards provided by modern development frameworks. They implement security features as prescribed by framework documentation but struggle to explain why specific approaches work or when framework protections might prove insufficient. This creates significant blind spots for advanced vulnerabilities. knowledge remains minimal, with most mid-level developers either entirely unfamiliar with these attack vectors or possessing only superficial theoretical awareness without practical understanding. Infrastructure-level security concerns such as CDN Tampering and Subresource Integrity frequently fall outside their perceived scope of responsibility, deferred to DevOps or senior developers rather than understood as shared security concerns.

Senior developers display comprehensive security understanding that encompasses both implementation details and strategic decision-making. They recognize that frameworks mitigate certain vulnerabilities but maintain awareness of edge cases where framework protections fail or prove insufficient. Their security knowledge extends beyond common vulnerabilities to include advanced attack vectors and infrastructure-level concerns. However, even senior developers acknowledge limitations in their security knowledge, particularly for emerging vulnerabilities and attack techniques that evolve faster than educational systems or professional development programs can address. The continuous evolution of security threats requires ongoing learning throughout careers rather than static knowledge acquired during training.

The seriousness with which developers approach security correlates directly with their exposure to security incidents and understanding of consequences. Senior developers who have experienced security breaches or worked on projects where vulnerabilities created legal liability or financial loss treat security as non-negotiable. They integrate security considerations into design decisions, negotiate timelines when security requirements conflict with delivery pressures, and allocate resources for security tools and processes even when these create workflow friction. This contrasts sharply with novice and many mid-level developers who view security primarily through a compliance lens, implementing security measures because they are required rather than because they understand the risks of not doing so.

### **5.4.3 Gamification as a Cross-Level Learning Approach**

A gamified method to teach novice developers about security coding was well supported by the developers of all experience levels. Grasping of traditional lectures taught in the university was not effective to novice developers, whose theoretical knowledge was soon forgotten once examinations were passed. They showed interest in the game based learning in terms of competition, leaderboard and interactive challenges which brought some engagement not found in normal instruction.

Mid-level developers already realized gamification as useful in overcoming the gap between theory and practice and recognized the importance of going deep rather than superficial. They valued such aspects of competition as leaderboards and the skill badges as displayed professionally, and expressed concerns over keeping the game examples relevant to real-life and over being abused maliciously.

The senior developers had subtle perceptions as some were inclined toward the conventional responsibility-based learning and others saw the value of gamification in mentoring and knowledge retention. There were strong proponents who stressed

the value of the tool in modeling vulnerability events without production risks. To align gamified learning with career development goals, they proposed that mindful efforts to integrate mystery gifts with market value (i.e. skills in API security, Docker and CI/CD pipelines) should be incorporated. The support at the intersection of experience levels shows that conventional education on security lacks the ability to establish a durable understanding, and gamification has a potential to fill the knowledge gaps in the lifecycle of the developer.

# Chapter 6

## Play to Secure Design and Development Process

### 6.1 Introduction

The results of the interviews carried out in the earlier chapters have revealed a lack of web security awareness amongst the novice developers. This includes a lack of knowledge of secure coding practices, common vulnerabilities as well as excessive dependence on security tools that are built-in with the modern frameworks. To address these challenges, conventional learning resources might not be engaging enough. A more innovative solution and a more interactive approach would be required to attract the attention of the learners and impart the best practices in an interactive manner.

To address this, the chapter describes the design and development of an educational gamified platform that focused on introducing web security awareness among novice developers. This game aims to use the principles of the game-based learning approach to integrate knowledge reinforcement with interactivity so that the game participants can assess their grasp of the concepts of web security in a low-risk interactive setting. The game will be designed to encourage the learners, make them feel motivated by providing them with scenario-based questions in the form of quizzes, immediate feedback, and an organized development of the progress system to ensure the retention of secure development practices.

This chapter describes the entire conceptualization, design and development procedure of the game. It starts with the description of the learning objectives and then proceeds to the description of the concept of the game and mechanics, framework, tools and technologies used in the development.



## 6.2 Design Objectives and Learning Goals

The interview revealed developers' preference for the design of the educational game as it was found that novice developers had serious knowledge gaps. It was revealed that most of the participants were not aware of basic web vulnerabilities such as: SQL Injection, Cross-site Scripting and that most were not aware of the OWASP top 10, it was also clear that most of them did not consider these vulnerabilities and security while coding. Even those who stated that they implement security measures, even in most cases, their measures were limited to two-factor authentication, basic input validation, or Firebase and JWT tokens, and everyone involved was in unanimous agreement that a gamified approach to learning would encourage them to learn and enhance their knowledge on web security. According to these results, the learning objectives of the game were to give an idea about the basic web vulnerabilities, educate on the impact of the insecure coding practices, and encourage secure coding. Moreover, it can help to promote active risk management of safety and make learning about web security more motivating by means of gamification.

## 6.3 Game Conceptualization

The game is developed as a security learning platform with some gamification to encourage learners and enhance their security knowledge. The quiz material, according to the interview results, is directly related to the gaps that were found among novice developers. The user motivation through eight core drives through the Octalysis Framework, developed by Yu-kai Chou, forms the conceptual framework of the game design. The mapping of the game components to these core drives, as portrayed in Figure 6.1, makes sure that the design does not just provide knowledge but also makes it engaging. Application of each of the respective core drives is as follows:



Figure 6.1: Octalysis Framework Implementation

### 6.3.1 Our Gamification System through Octalysis Process

The design of “Play to Secure” follows Octalysis by intentionally balancing Left Brain motivators (extrinsic, structured, goal-driven) with Right Brain motivators (intrinsic, exploratory, creative). The following design implications directly shaped how our features were implemented across the seven-attack learning path.

#### Epic Meaning and Calling (White Hat, Right Side)

This means believing you are part of something bigger, inspired by purpose. Developers are guided on a mission to build secure systems and protect organizations from real-world attacks. To prevent novice developers from treating security as an optional burden, the game frames the player as a “Security Guardian” with a meaningful mission. Instead of “fixing bugs,” the learner is positioned as someone protecting local digital infrastructure and the safety of users on real-world platforms. This sense of greater purpose (“development of greatness”) turns secure coding into a responsibility with social impact, helping learners feel that completing levels contributes to something beyond personal grades or tasks. This makes players feel their learning is directly connected to real-world opportunities.

### **Development and Accomplishment (White Hat, Left Side)**

The drive to make progress, achieve mastery, and overcome challenges. The system is based on Development and Accomplishment that is executed with the help of the Game Path. The play increases in a graded series of seven attacks with levels, points, progress tracking, and quizzes. The level gives instant feedback- players can know what they have accomplished, what they still need to accomplish, and how much they have left to complete the entire curriculum. Badges and points are awarded for correct answers. The visible growth design will make sure that the learners feel the continuous growth in the form of short milestones, which will keep the motivation to progress in the whole game. Coding prevention exercises (Level 3) give mastery recognition as users succeed in building secure systems.

### **Empowerment of Creativity and Feedback (White Hat, Left Side)**

The motivation to explore, experiment, and receive feedback. The platform is not constructed for passive reading contents. Rather, our level-based gameplay and interactive simulation promote meaningful play, in which learners themselves make decisions, attempt solutions, and immediately get results. Players have an option on how they approach tasks, experimenting with inputs, discovering consequences, and learning by trial and error. Level 3 coding prevention challenges let players experiment with various prevention strategies and see instant feedback on whether they succeeded. Each attempt gives personalized feedback to guide players toward mastery. This makes security learning more of an experiment where users learn through cause and effect and not by definition.

### **Ownership and Possession (White/Black Hat Mix, Right Side)**

The drive of feeling ownership over progress and rewards. As players progress, they unlock new features like the knowledge wall, chat, and groups. Players receive Vulnerability Badges every time they master a module to create a sense of ownership that will last long. The badges can be collected as a personal and satisfying goal of completion. This is reinforced by the profile identity of the player, where the progress is displayed, and by the leaderboard that the user is in and that they are at a lower or higher position, which makes one more attached to accomplishment (this is my progress, my rank, my badge set). It is thus not only a symbolic ownership (badges) but also a social and measurable one (position and recognition), reinforcing a sense of ownership.

### **Social Influence and Relatedness (White/Black Hat Mix, Right Side)**

The drive of social interaction, recognition, and belonging. Given the high peer learning culture and community impact (particularly in Bangladesh), we introduced

a special social platform within the framework. The Social Wall gives the learners the opportunity to share their progress, insights, and knowledge. Through this, they can communicate and stay connected on a personal level. Chat and group creation features encourage collaboration and team-based learning. We also added two leaderboards to maintain the meaning of foster collaboration, peer learning, a sense of belonging, and healthy competition among players. Players receive validation through likes, comments, and peer recognition on their posts and contributions.

### **Scarcity and Impatience (Black Hat, Right Side)**

The drive of wanting something rare or limited. The game does not give the player everything right at the beginning; higher levels and some community interactions are deliberately disabled until the player reaches milestones. It consists of waiting to access features like chat, group discussions, and posting privileges (or full-time community participation) and unlocking subsequent attacks only after the previous ones have been finished. The system will create an urge to advance to new levels and get to upper levels, as well as provide access to what is not accessible by restricting it in the early stages. Exclusive badges are available only to players who complete certain challenges early or under special conditions.

### **Unpredictability and Curiosity (Black Hat, Left Side)**

The drive of curiosity and surprise, wanting to see what happens next. Mystery Gift rewards, which present unexpected content at levels of progression and make players want to continue. Randomized quiz questions in each attack category to keep learning fresh. Also, different representations of each attack, in which all vulnerabilities have a specific theme, user interface, and narrative. The difference makes the learner not feel that one is repeating the same screen pattern seven times; however, every attack is a new experience at the same time, supporting security goals.

### **Loss and Avoidance (Black Hat, Left Side)**

The drive to avoid negative outcomes and penalties. Players lose points for incorrect quiz answers. If they fail to complete coding prevention tasks, they miss out on rewards or slow their progress. The system displays a warning when insecure decisions are made. Moreover, the learners suffer direct loss in the form of the loss of points as a result of negligence. In SQLi, as an example, when a user uses an incorrect method or provides an incorrect query or solution throughout the progression, they will lose marks, a fact that supports the idea that insecure or haphazard coding does not go to waste. Time-sensitive challenges where failing to complete on time means losing rewards, or even access priority to certain features. Through this integration of Octalysis principles, the game aims to create an engaging and

motivational learning environment where novice developers can repeatedly practice and reinforce essential cybersecurity knowledge. This core design philosophy is that learning should be both educational and enjoyable, ensuring that the lessons have a long-term impact.

## **6.4 Game Design and Development Process**

### **6.4.1 Game Design Objectives and Learning Principles**

Here, we present the objectives behind the game design process and the learning principles that informed the creation of the objectives. Our objective in developing the Play-to-Secure platform was to teach users four things:

- Foster awareness that development and security demand equal rigor.
- Promote the adoption of secure coding practices through gamified competition and enjoyment.
- Illustrate the consequences of insecure coding practices and the risks of neglecting them.
- Establish a community-driven platform to raise security awareness among novice developers.

According to the requirements analysis, interview findings, and the design process that were presented in the previous chapters, this section relates the pedagogical objectives to the functionality implemented by the system. Specifically, it is dedicated to how seven core web vulnerabilities can be converted into interactive learning units and how the entire game-based learning process facilitates the development of safe coding habits in novice web developers. The game design is based on literature review, developer interviews, and gaps in traditional security education. It has been shown by previous research and the findings of interviews that novice developers are generally more concerned with functionality than with security, and they usually view security as a secondary feature of development, instead of a part of development. To solve this problem, Play-to-Secure inserts security decisions into development-like problems and supports the idea that secure coding cannot be isolated and introduced as a separate development problem. Learning science research has proposed that successful training ought to be contextual, goal-oriented, challenging, and interactive [6]. In line with this, the platform focuses on experiential learning as opposed to passive learning. The findings of the previous research and

our interview further that passive methods of learning cannot be effective in long-term knowledge retention. Consequently, the majority of the vulnerabilities in the game are learned by interactive simulations where the players initially observe an insecure behavior, then apply it to a controlled environment where they learn to exploit the behavior, and finally apply or study the correct mitigation factors. All the vulnerability modules are further broken down into levels to fit novice learners. Lower levels provide simplified models of attacks, and higher levels include more realistic models like authorization logic, database query syntax, and client behaviour. This progressive development of complexity can be correlated with cognitive load theory [17], by which learners are able to develop knowledge bit by bit without feeling overwhelmed. During gameplay, the system also offers instant feedback in the form of visual effects, hints, explanations, and quizzes, which allow fast learning loops.

The gamification design is further guided by the concepts of the Octalysis framework, namely feedback focus, empowerment, and meaningful engagement. The game features, like points, leaderboards, and quizzes, as well as mystery rewards, are carefully structured to support the learning results instead of distracting them. Combining motivation and instructional depth, the platform is expected to facilitate short-term and long-term behavioral change due to competitive and exploratory learning processes [6] [7].

First of all, we used the reflection principle that states the need to give learners a chance to pause and reflect on what they have learned. Research proves that learning is enhanced when educational games provide self-reflection of newly learned material [6]. This is followed in Play-to-Secure by using end-of-level score sheets that examine the vulnerabilities experienced, whether the player was able to recognize security issues correctly, and provide explanatory information on what constitutes secure coding practices. Secondly, we used the story-based agent environment principle, which includes the incorporation of agents in the story-based content, promoting learning. Stories make people learn because they arrange events in a significant context and provoke thinking processes [3] [30]. Our platform will put the learner in the role of a Security Guardian, having a valuable mission of securing digital infrastructure, making secure coding not an optional load but a socially significant responsibility. Moreover, in each of the seven attacks, the player has a different agent environment. For example, in the NOSQL injection level, he is a white knight saving his king in chess with 4 turns, and in the XSS level, he will take the role of a developer and see the consequences of his negligence in input sanitization. Thirdly, we applied the principle of conceptual-procedural, which states that conceptual knowledge and procedural knowledge affect each other in mutually supportive directions and develop iteratively [4] [17]. Every vulnerability module introduces conceptual knowledge on

the mechanisms of attacks, and then it introduces procedural knowledge on certain mitigation measures. As an example, the SQL Injection module has its first section explaining how query integrity (conceptual) is undermined by unsanitized input, then proceeds to teach the use of parameterized queries (procedural). Also, in the Broken Authentication and Authorization level, users will know about the importance of hashing passwords, and later, they will be given the experience of how it works. According to the learning sciences theory, effective training should be goal-oriented, challenging, contextual, and interactive. It thus provides simulated real-life development conditions where the learner can identify weaknesses, test attack vectors, and implement mitigation policies as the difficulty increases, as opposed to being taught to do so [6] [7].

Leaderboards are also some of the most popular gamification features in education, and their design has an immensely powerful impact on learner engagement and motivation [62]. The empirical studies on the topic are systematically reviewed to substantiate the idea that properly designed leaderboards can increase learning motivation, engagement, and performance, and one should exercise careful design decisions. Comparative analyses also indicate that relative leaderboards (placing learners in competition with others) lead to increased engagement and healthy competition, but absolute leaderboards may stress out underachievers [48]. On this basis, our Play-to-Secure platform uses two leader boards to account for the two types of competition, a macro speed-based leader board that rewards fastest completion (prone to domination by the best competitors) and a micro post-game participation leader board that rewards sustained engagement, reducing domination whilst promoting inclusive competition- as evidenced by these findings.

### **6.4.2 Hybrid Approach for Developing the Gamified System**

Our educational game will be developed with the help of a Hybrid Agile-RAD (Rapid Application Development) process. This approach to properly manage the development process in our research context. We plan on merging the iterative nature of Agile and the rapid prototyping of RAD. This type of hybrid method will allow us to find a compromise between creative flexibility and systematic development.

#### **Rationale for Hybrid Agile-RAD Selection**

We plan on using a Hybrid Agile-RAD approach due to the unique requirements of our game. As a result of our research team consisting of five members, we adopted a framework that encourages collaboration, accountability, and adaptability. Moreover, the experimental nature of the game production has also been considered in

the decision-making process.

- The Agile component of the project will enable us to track our iterative progress and help us allocate tasks clearly to each member. Moreover it will include structured sprints which would make the project more organized and ensure that each team member had defined responsibilities [16]
- The RAD component will allow us to quickly prototype and test the game's features such as the quiz engine and leaderboard. As a result we will be able to validate the design ideas before full scale integration. [2]

The traditional waterfall SDLC models would not fit our project since they are based on massive upfront planning, and no testing is done in this methodology. Likewise, pure Agile would also be inappropriate because it may be process-intensive for a small team like ours. In the case of an educational platform that requires constant verification and progressive refinement, the Hybrid Agile-RAD methodology presents the most optimal combination of structure, flexibility, and rapid prototyping.

### 6.4.3 Development Workflow

We used a systematic six-step iterative workflow in our work process, as shown in Figure 6.2:

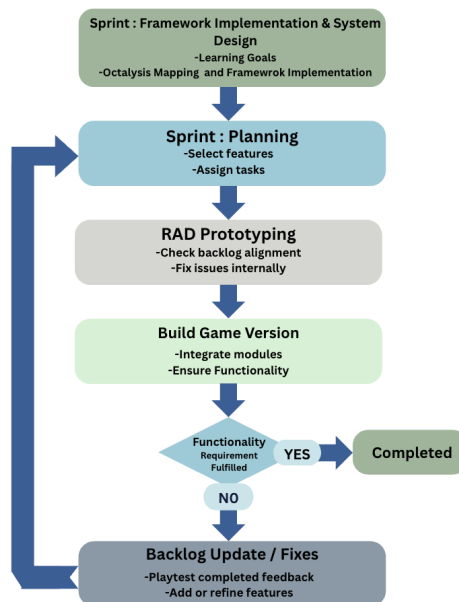


Figure 6.2: Development Workflow

#### 1. Sprint (Framework Implementation and System Design)



This planning step involves defining learning objectives and mapping them to Octalysis Core Drives. As an example, the quiz feature will be in accordance with the Core Drive 2 (Development and Accomplishment) and the leaderboard will be related to the Core Drive 5 (Social Influence and Relatedness). Certain features will be given priority based on their motivational value and practicality and these will be recorded in a product backlog.

## **2. Sprint Planning**

Two or three features were selected for each sprint and the team members were assigned tasks according to their area of expertise (backend logic, UI/UX design, database integration). We then made sure that everything had been grounded on the principles of Octalysis core.

## **3. RAD Prototyping**

This phase will involve rapid prototypes that will focus primarily on the basic mechanics rather than refinements. With RAD, we plan on building prototypes, test functionalities, and make necessary refinements iteratively. For instance, the original prototype would include a leaderboard displaying the top five ranked participants and a quiz system with multiple-choice questions designed for quick scoring. To enhance user engagement, we also plan on adding badges based on participant progress (Core 4 of the Octalysis framework). RAD would allow us to implement these additions efficiently through its iterative development approach.

## **4. Final Version and Evaluation**

This phase involves plan on assembling the prototypes that will be integrated into a single comprehensive system, combining the leaderboard, scoring, and quiz functionalities into a seamless user experience. All functionalities will be merged together, and the system designer will evaluate whether everything is working properly and aligned with the project objectives.

## **5. Refinement**

Developers will test the system and provide feedback regarding user engagement and learning objectives. Based on the findings, the backlog will be updated to incorporate improvements or plan additional iterations as needed.

This Hybrid approach would offer the flexibility to test with the mechanics of gamification and keep the continuous flow by organized sprints, making it a logical solution to the development of an educational game.

# Chapter 7

## Design Implications and Game Functionality

This chapter describes the design implications based on the research results and how the proposed gamified platform works in practice. Based on the requirements analysis, the interview results, and the design process mentioned in previous chapters, this chapter relates the pedagogical objectives to the system's actual functionality. It is concerned with how the seven fundamental web vulnerabilities can be converted into interactive learning units and how the entire game-based learning process can be used to promote the development of secure coding habits among novice web developers.

### 7.1 Design Implications

The game design is based on some of the major implications found during the literature review, interviews with the developers, and gaps found in the traditional security education. Security should be an aspect of development that is instilled, but not a mission after development. The results of the interviews indicated that novice developers tend to prioritise functionality over security. Consequently, the game integrates security decisions into development-like tasks, which strengthens the notion that secure coding cannot be separated in any way from daily programming.

Learning should be practical and interactive. As the results of the previous studies and the interview study revealed, passive learning techniques could not be applied to long-term retention. Thus, the majority of vulnerabilities will be taught using interactive simulations, during which the players will witness an insecure behaviour, abuse it in a controlled setting, and then implement mitigation measures or learn

how these work. Each of the vulnerability modules is broken down into several levels to accommodate novice learners. Simplified concepts of attacks are presented at early levels, whereas realistic constraints, like authorization logic, database query structures, or client-side behaviour, are presented at later levels. This progressive increase is in line with cognitive load theory.

The game offers immediate feedback with the help of visuals, hints, explanations, and quizzes, enhancing learning loops. This is consistent with the focus of the Octalysis framework, where the emphasis lies on empowering creativity and feedback so that learners can learn within a short time why a certain action is unsafe and how it can be rectified.

The motivation should be moderate with the depth of education. The aspects of gamification, including points, leaderboards, quizzes, and mystery rewards, are specifically created to reinforce the learning outcomes as opposed to distracting them. Short-term involvement and long-term behaviour change are promoted through competitive and exploratory mechanisms.

## 7.2 How the Game Works

The gaming experience starts with a central dashboard, which acts as the main user entry point, as shown in Figure 7.1. Using this dashboard, players can directly access the main Game Path, community interactions, or review their attack details.

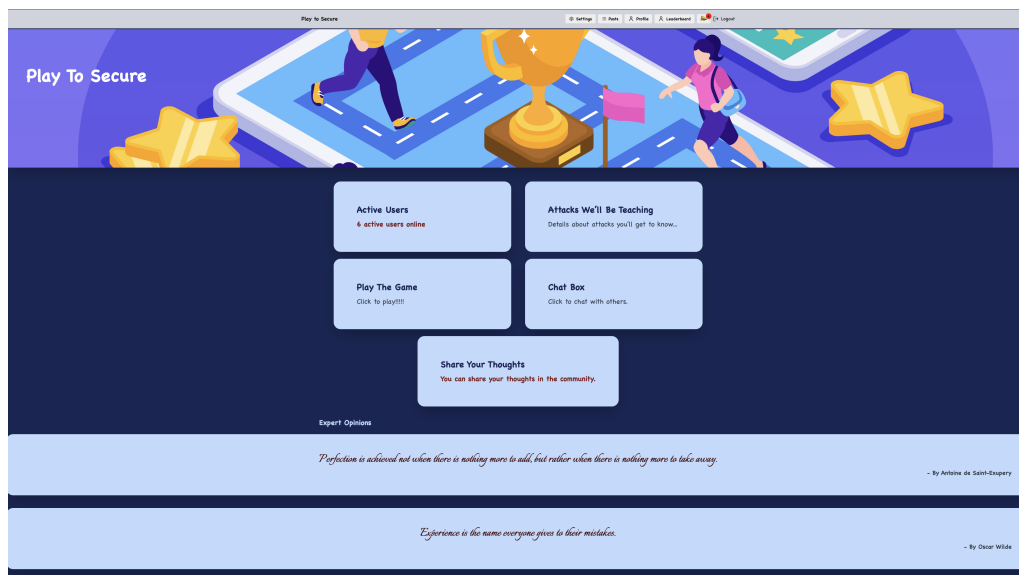


Figure 7.1: Game Dashboard

The platform avoids viewing security training as a one-time task and instead po-

sitions it as a continuous practice by enabling users to move freely between the learning and social experiences. The flow of the training material is structured in a Game Path in the form of a map presented in Figure 7.2. Every security vulnerability, such as SQL Injection and XSS, is represented as a separate node on the route, and a player has to pass the levels in order to progress.



Figure 7.2: Game Path

Such a graphic depiction of progress diminishes the confusion that may occur in the mind of a novice developer since the sequence and extent of tasks are clearly shown. Instead of confusing users with unlimited options, the journey-based format explicitly shows the next step, the amount of the content, and the personal challenges as they are part of the overall learning goal. In addition to the actual gameplay, the system also has a social wall, chatbox, and group interaction features. The social wall part is shown in Figure 7.3.

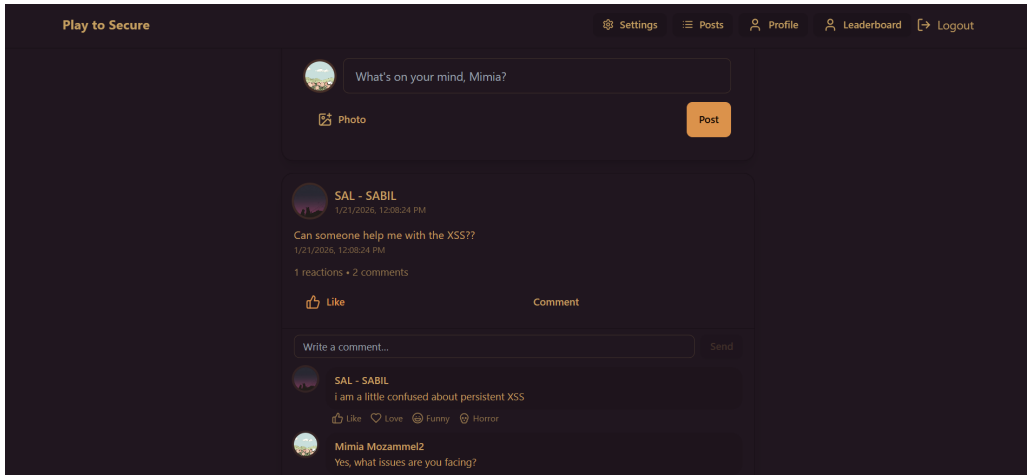


Figure 7.3: The Social Wall

These elements enable users to update, comment on others' experiences, and share

difficulties encountered while playing the game. These features have been added because peer discussion and collaborative problem-solving can be highly helpful when creating secure code. Simultaneously, the admin handles the quiz questions used in the game module, mystery gifts that will be provided and notifications for each purpose. These are shown in Figure 7.4. Along with that, governance is required. It is on this basis that community rules and admin control are incorporated into the system, so that collaborative learning can be constructive, safe, and educational.

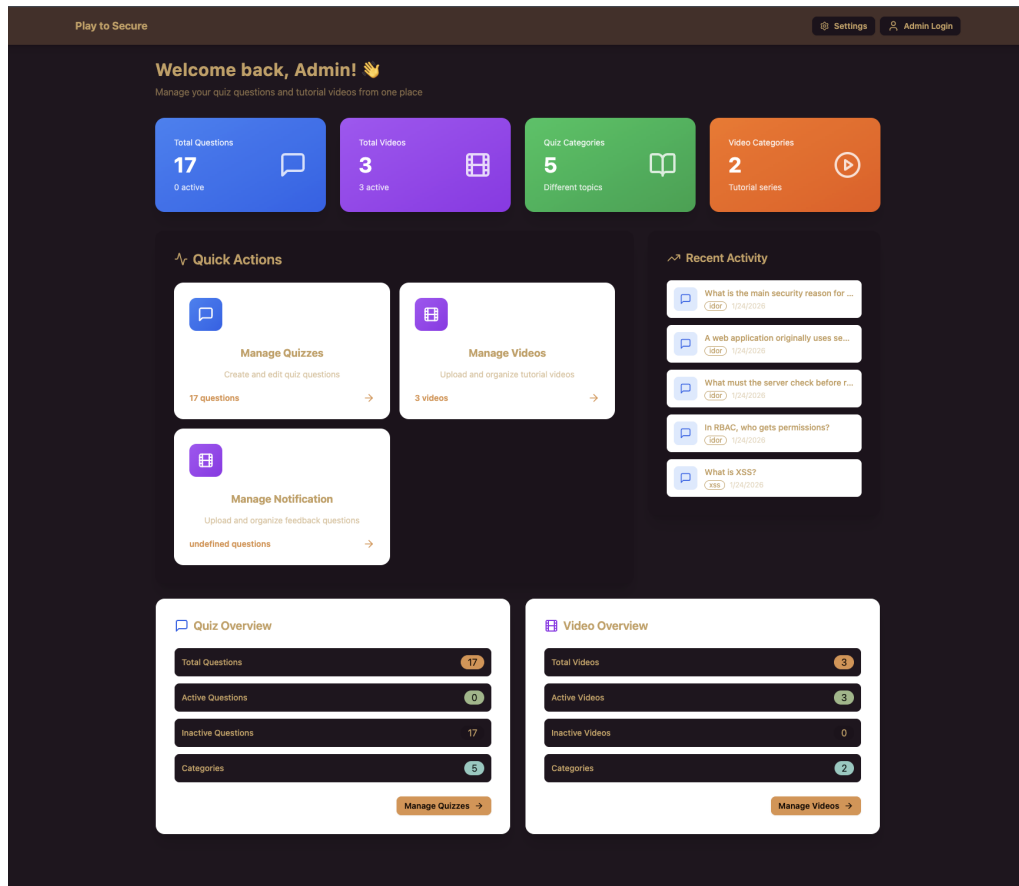


Figure 7.4: Admin Dashboard

Another design implication is manifested in the decision to use two different leaderboards. It is impossible to have one leaderboard that can be used to reflect the various types of achievement without watering down the concept. The Game Leaderboard is portrayed in Figure 7.5. It promotes competition through the assistance of performance indicators (vulnerability completion, quiz success, and level progression), which makes sure that players are working on their competitiveness.

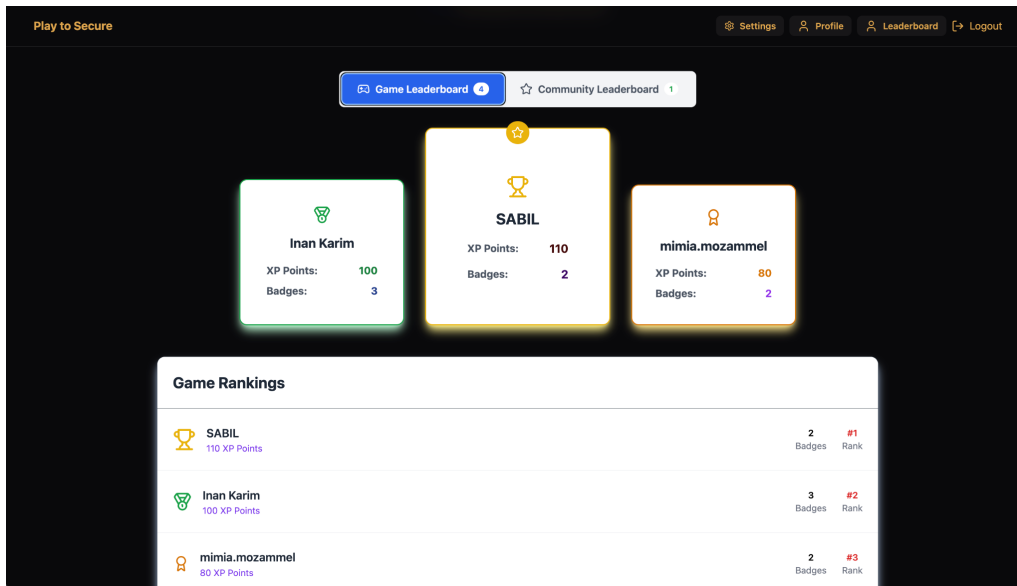


Figure 7.5: Game Leaderboard

After accomplishing the task, the user is more experienced than the players who are still playing, and unlocking the Community Leaderboard, as shown in Figure 7.6, at that stage will allow the experienced players to share knowledge, guide others, and be given the credit of making significant contributions (posts, comments, reactions).

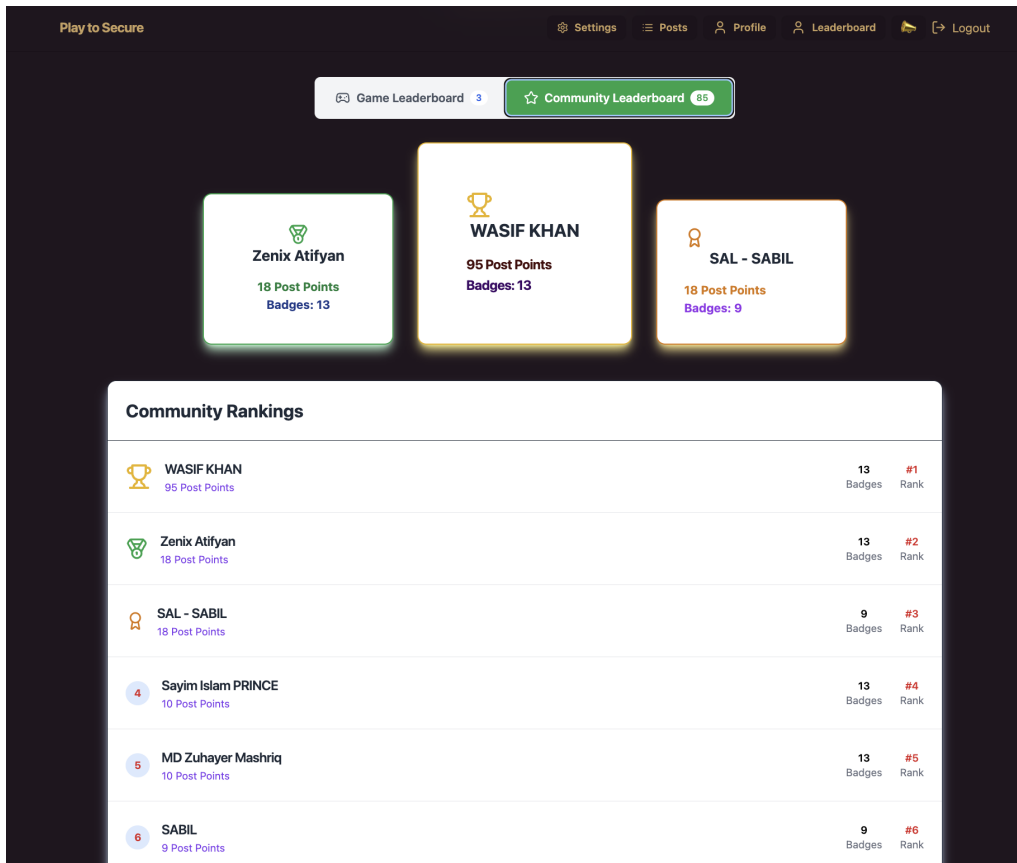


Figure 7.6: Community Leaderboard

This makes the leaderboard not obsolete after the game is finished; it does not lose its value after the game is over, and it also does not allow new players to grind easy social points rather than learning, as the community scoring is a reward that can be earned only after the training is finished.

### **7.2.1 Admin Panel Design Implications**

The Admin Panel is very important in maintaining the quality, safety, and durability of the platform. Content-wise, it enables the administrators to edit, add, or delete quiz questions related to a particular level, which keeps learning materials relevant and accurate. The panel also assists in managing a video library used to supplement the gamification system with explanatory content and reinforcement content. The Admin Panel allows governance and behaviour-based interaction using a notification system, in addition to managing content. The use of this system is two-fold. From a moderation perspective, administrators are able to give warnings or advice when users post unacceptable or disturbing posts, hence ensuring a good and respectful learning environment. Motivationally, users who are performing well can be provided with a notification to keep them going, or those who seem to be struggling can be provided with specific assistance, using performance and behavioural indications. Such a combination of control and support has a direct impact on a safer, more accommodating, and sustainable learning community, especially with the existence of social features.

### **7.2.2 Mystery Gifts and Reward Design**

Another feature of the platform is the Mystery Gift, whereby reward videos are unlocked at certain levels, which can be controlled by the Admin Panel. These rewards are made to be unpredictable on purpose, and this brings anticipation to the learning process. Design-wise, these unlockable rewards act as a surprise. This mechanism comes in especially at later stages of the game, where learners can get tired as the challenges get complicated. The system minimizes the rate of drop-off by occasionally providing unexpected rewards to keep the learners engaged without diminishing the severity of the learning goals.

### **7.2.3 Design Details per Vulnerability**

One of the design implications of the current project is that traditional learning systems may easily turn out to be monotonous, particularly among new learners. In

response, interaction style and visual themes like comic, chess, and jungle, in each vulnerability module are different, and the same underlying learning loop is used. In this strategy, learners maintain familiarity in the way they engage with challenges and have novelty in the different presentation styles.

## SQL Injection (SQLi)

Players are exposed to unsafe server-side query handling environments in a cave theme, which is introduced in Figure 7.7.

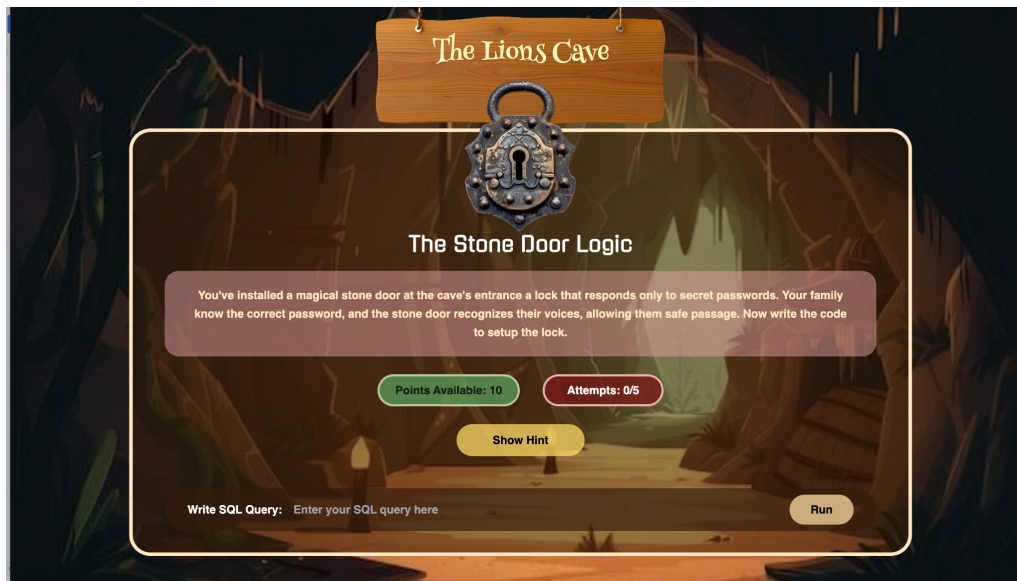


Figure 7.7: SQLi Scenario

The bottom levels depict the alteration of SQL queries due to unsanitized input, whereas the upper levels incorporate parameterized queries and safe backend code[9][20]. The SQL Injection module involves a breach-report narrative and a simplified flow description. The players are made to first feel the blow of the poor input handling security, which can either be a data breach or unauthorized access, before realizing what the manipulated inputs have done to the backend queries [9]. The training will be done with the help of a quiz checkpoint that will determine the mastery of concepts, which will ensure that the learner will be able to correlate.

## Insecure Direct Object Reference (IDOR)

The IDOR module is the adventure-based module, presented in Figure 7.8. This is mainly a jungle-themed module, which highlights the idea of gaining access to resources without the necessary authorization.



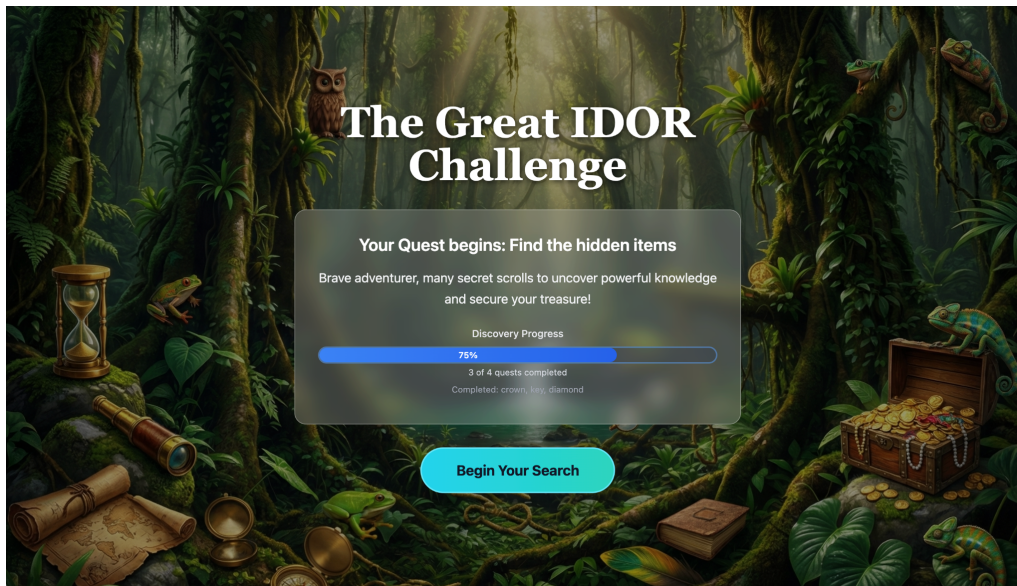


Figure 7.8: IDOR Scenario

The IDOR problems are broken access control problems where an attacker exploits predictable identifiers of objects to gain access to unauthorized resources. Players are presented with situations where they have to find four elements in the game, and for each element, they will be presented with four different scenarios for each case. These include URL parameter manipulation (e.g., replacing `userId=123` with `userId=124` in API requests), which is used to expose unauthorized data because authorization checks are not done [60][68][70]. Later, they apply several defense layers: they use Universally Unique Identifiers (UUIDs) instead of sequential identifiers[68][70], they use user context, not URL parameters, with authentication tokens [49][60], they use ownership checks, performing server-side checks that authenticated user IDs are the same as `resource.ownerId` [60], and they use RBAC and ABAC to control access to resources based on roles and attributes [49][60].

### Cross-Site-Scripting (XSS)

The Cross-Site Scripting module illustrates attacker-to-victim flows through the comic-style storytelling, which is demonstrated in Figure 7.9.

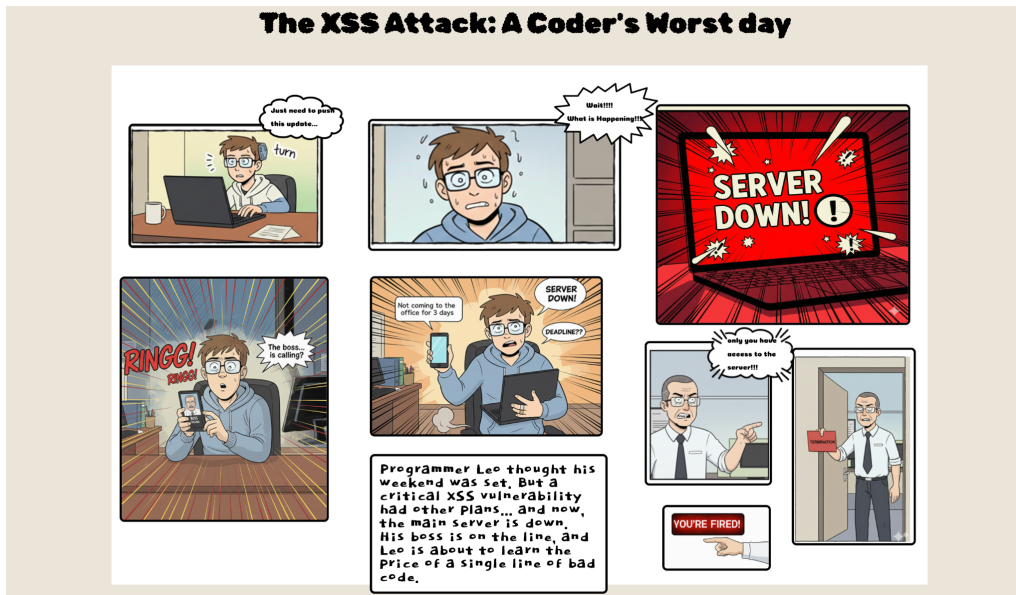


Figure 7.9: XSS Scenario

This module is concerned with the mishandling of user-generated content in the frontend. Players get to know how reflected and stored XSS attacks are and how input sanitization, output encoding, and Content Security Policy (CSP) can help in mitigating these threats [23][12]. The various types of XSS are displayed as narrative panels, which show the way malicious scripts spread and influence users. Encoding and sanitization, Dom purify [23] are introduced later into the story, which strengthens the relationship between the visualized attacks and the actual defence measures.

## NoSQL Injection (NoSQLi)

This module shows some of the types of NoSQL injection that are possible using the operators of MongoDB. This includes *\$ne* or *\$gt*, *\$where* to exploit queries. Players get to know that NoSQL databases are not resistant to injection attacks and how to use secure libraries. In NoSQL Injection, the design has a theme of a strategy board that looks like a chess game, which is shown in Figure 7.10.

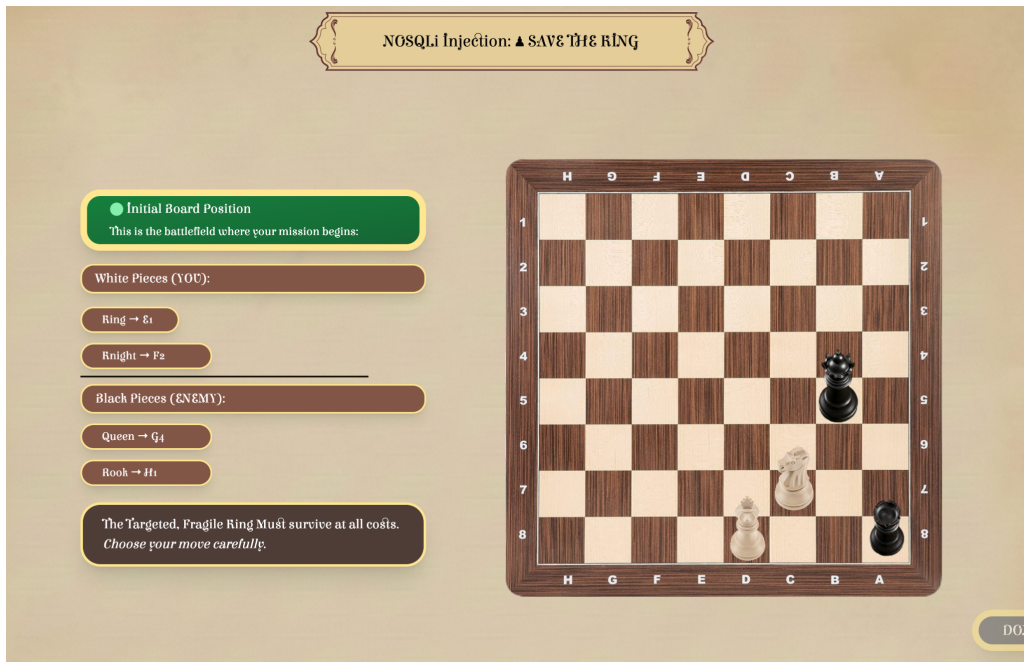


Figure 7.10: NoSQLi Scenario

Such a metaphor signifies the sequential character of injection logic and splitting query choices, like using the query parser, `mongoSanitize()` [59][46]. Viewing interactions as a considered action, the module promotes critical thinking as opposed to trial-and-error, which is especially important since NoSQL injection is relatively unknown to novice developers. In the meantime, the user can get to know the significance of utilizing the Express Mongo Sanitize.

## Clickjacking

UI-based attack simulations allow the players to experience the way users are tricked by malicious iframes and overlays using a magical theme to make it more engaging, as portrayed in Figure 7.11.

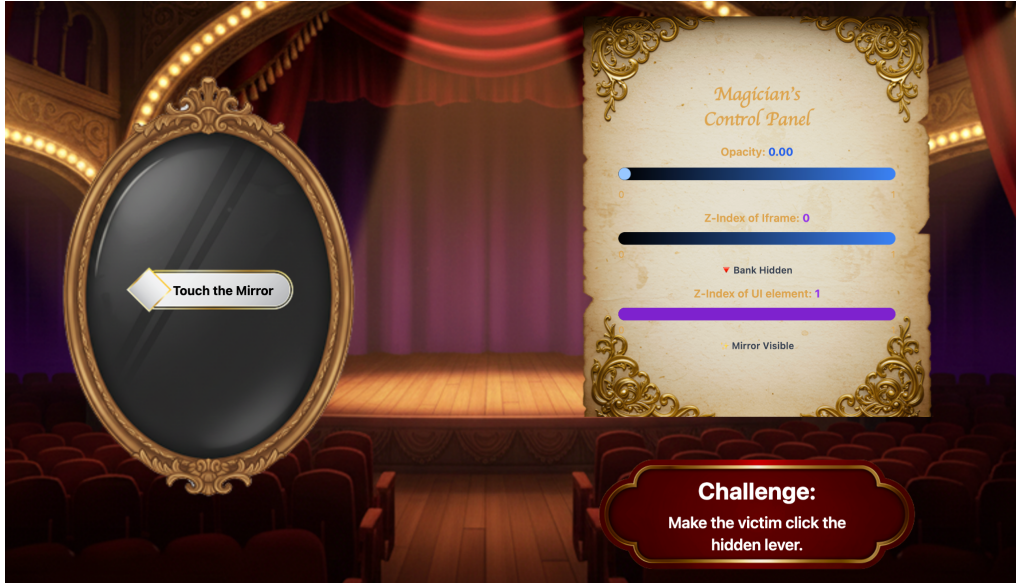


Figure 7.11: Clickjacking Scenario

X-Frame-Options and CSP frame-ancestors are defensive measures brought up. It entails exercises on implementing mitigation measures that take the learners through frame-busting JavaScript (e.g., if (top.location != location) top.location = self.location) [13] and server-side protection (e.g., HTTP response headers (X-Frame-Options: DENY, X-Frame-Options: SAMEORIGIN) and modern CSP commands (Content-Security-Policy: frame-ancestors "self ")) [15][13]. Comparative analyses at the end of the module will explain the greater strength of server-side headers over client-side JavaScript frame-busting because JavaScript can be bypassed [15][13].

## Dom Clobbering

The DOM Clobbering module introduces players to a bit of an unfamiliar but advanced client-side vulnerability, in which the attacker has the ability to overwrite JavaScript variables or functions using HTML elements due to name clashing on the DOM name[54].





Figure 7.12: Dom Clobbering Scenario

This module focuses on safe handling of the DOM and further exposes more insights into the behaviour of the browser that could be used. This abstract vulnerability is introduced using the DOM Clobbering module based on the story-driven discovery mechanism, and a magical theme showed in Figure 7.12. Players start with a narrative Story Time that frames the situation with exciting magical scenarios that allow the learners to grasp the overall idea, and only then move on to the technical aspect of implementation. The module then moves on to illustrated boards that reveal three major learning paths: Browser Weirdness, which is a description of the unexpected behaviour of a browser; How to Prevent DOM Clobbering, which outlines the steps to take as a developer to protect the security of the development; and What Developers Should Do, which contains advice on how to act to ensure the safety of development. Players consider actual code examples, in which the names of the elements of the form `<input name=x>` can accidentally create global variables and circumvent the JavaScript expectations. The interactive game challenge, the City of Names, puts the player in the role of a junior city engineer charged with the responsibility of running a bureaucratic system, two competing registries, the Registry Book (HTML IDs) and the Council Book (JavaScript), to support the argument about the real-life implications of naming disputes. This game scenario has shown that the mix of HTML names and JavaScript variables can lead to unexpected behaviour. The module still goes on to concrete code cases of vulnerabilities and secure versions, and pragmatic hints like setting on the use of strict mode to prevent accidental global variables [71]. With a gradual shift between story and structure, the module can lessen cognitive load without compromising on the technical rigour and enable learners to make sense of how apparently harmless HTML can be used to subvert the execution of JavaScript functionality and how to avoid such attacks

by inoculating oneself against such attacks through defensive programming [55].

## Broken Authentication

The Broken Authentication module is in the form of a medieval castle siege narrative with a forge-like transformation process, which is shown in Figure 7.13.

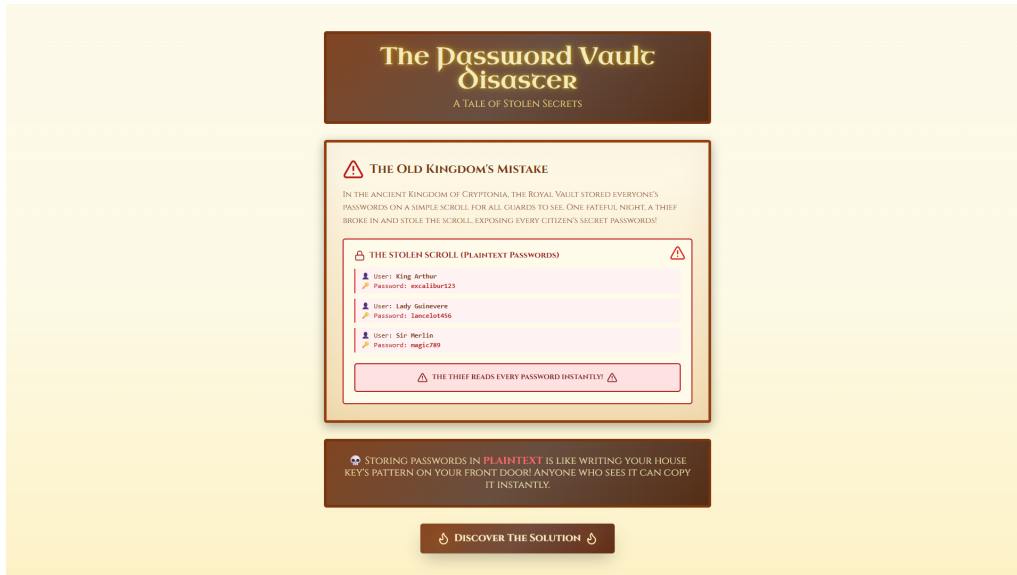


Figure 7.13: Broken Authentication Scenario

The experience uses a wizard-themed quiz checkpoint that authenticates conceptual knowledge by using interactive challenges to ensure that learners relate technical authentication mechanisms to real-life attack situations. The module builds on the fundamentals of password hashing, such as how the bcrypt algorithm can be used to hash plaintext passwords into non-reversible hash formats with the help of salt rounds, to the higher level of session management and token security. Castle gate metaphors of authentication barriers and forge animations to display the one-way hashing transformation are visual representations that support the idea that strong authentication needs more than just password complexity [1].

### 7.2.4 AI-Based Automated Secure Code Assessment

A code analysis and scoring system based on AI is assessed on user-submitted code to determine security vulnerabilities and compliance with best practices. Figure 7.14 shows that on receiving the code snippets based on the scenario presented, the system compiles them to discover both secure and insecure coding patterns (such as the use of parameterized queries instead of string concatenation in the context of SQL Injection), and a percentage-based score on the security of the code is assigned.

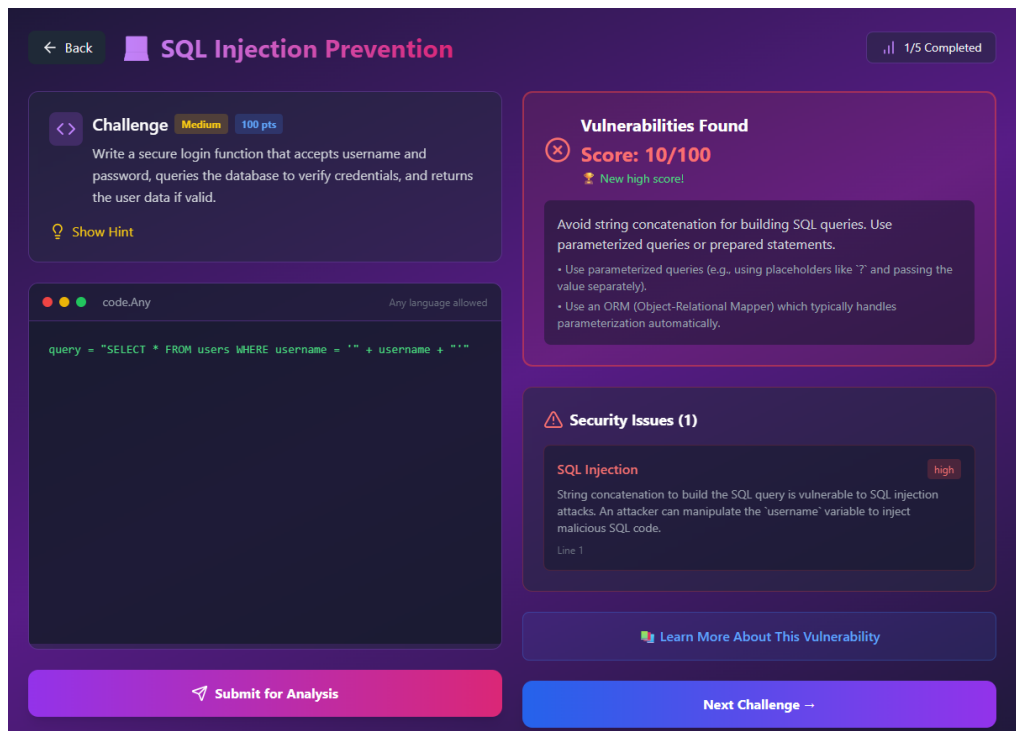


Figure 7.14: Level 3 Scenario

Such a score indicates the ability of the submitted code to reduce the targeted vulnerability as opposed to checking the functional correctness. The system then gives explainable feedback as to why the code is secure or partially secure, what defences are missing, and reinforces existing principles of secure coding. To facilitate a long-term interaction, every level is created as an ongoing, endless cycle of user interaction instead of a challenge that is one-dimensional. New challenges, newer vulnerability scenarios, and fine-tuned evaluation criteria are introduced every now and then, such that repeat users are always presented with new content. This interactive design fosters recurring engagement, long-term skill building, and makes the learning process consistent with the dynamic character of the security threats in the real world. Such systems have been traditionally applied in gamified cybersecurity education systems, secure programming training software, and developer education systems, where immediate, adaptive feedback is employed to instruct developers to internalize security conceptualizations during the development process, rather than after development.

Overall, this chapter explained the game's basic functionality and design, which incorporates social interaction, progression systems, administrative control, and reward mechanisms. It was also able to show how different thematic designs were used on each vulnerability to ensure that the learning remained engaging without altering the learning structures. These design considerations all favour an interactive and communal method of teaching secure coding that extends past the detached

gameplay and into the long-term learning practice. In this chapter, the design implications that inform the gamiest platform were presented, as well as how seven fundamental web vulnerabilities are programmed into interactive learning modules. It also outlined the key features of the game and the general process of its work. All these design choices make it clear that, when based on empirical research and pedagogical study, gamification can be a powerful way of teaching novice web developers safe coding skills.



# Chapter 8

## Game Evaluation Methodology

After designing and developing the gamified platform described in chapter 6, a systematic evaluation platform is essential to assess its quality, educational effectiveness, and impact on novice developers' learning. The evaluation must go beyond superficial metrics to determine whether the game successfully addresses the critical knowledge gaps and insecure coding practices identified in our interview study previously.

The interviews revealed that novice developers lack fundamental awareness of common vulnerabilities, depend heavily on frameworks without understanding underlying security mechanisms, and find traditional educational methods ineffective for long-term retention. These findings necessitate an evaluation approach that can measure not only whether the game is functional and engaging, but whether it actually improves security knowledge and motivates behavioral change in development practices.

This chapter presents the evaluation methodology for the educational game. It describes the MEEGA+ framework selected for this evaluation, explains the rationale for choosing this specific model over alternatives, details how the evaluation will be conducted, and outlines the expected outcomes that will determine whether the game achieves its educational objectives.

### 8.1 The MEEGA+ Evaluation Framework

#### 8.1.1 MEEGA+

MEEGA+ (Model for the Evaluation of Educational Games) is a validated framework specifically designed for evaluating educational games for learning purposes.

Developed by Petri, Gresse von Wangenheim, and Borgatto (2018), it evolved from the original MEEGA model through studies involving over 700 students across 40 case studies, addressing validity and reliability issues in the initial version [36].

The framework evaluates games across two major quality parameters. Usability measures functional quality through four dimensions: aesthetics (visual design and interface appeal), learnability (ease of understanding mechanics), operability (clarity of controls), and accessibility (readability and visual clarity). Experience of the player considers educational and motivational factors in a nine dimensional set that includes: focused attention, fun, challenge, social interaction, confidence, relevance, satisfaction, perceived learning, and user error protection [36].

MEEGA+ uses a standardized 35-item questionnaire with a 5-point rating scale and includes an Item Response Theory (IRT) based quality scale that uses the statistical analysis in classifying the games into three categories, low ( $\theta < 42.5$ ), good ( $42.5 \leq \theta < 65$ ), or excellent ( $\theta \geq 65$ ) based on statistical analysis and not subjective judgment alone.

### 8.1.2 MEEGA+ Evaluation Approach

Our interview findings revealed that novice developers need both knowledge improvement and motivational engagement, and MEEGA+ evaluates both simultaneously. It will address our core research question about whether gamification can effectively improve security awareness. The framework's rigorous psychometric validation, with Cronbach's alpha values of 0.856 for player experience and 0.930 for usability, ensures credible and generalizable results [36]. MEEGA+ also provides ready-to-use tools including standardized questionnaires, automated spreadsheets, and R scripts. It will reduce implementational complexity while enabling objective quality classification against established thresholds. Unlike frameworks that focus narrowly on usability or learning separately, MEEGA+ recognizes that educational game success depends on the intersection of both factors. A game might be highly usable but fail to engage learners, or engaging but ineffective for learning. MEEGA+ is perfect for evaluating our game fundamentally since it captures this complexity through its extensive dimension structure.

### 8.1.3 MEEGA+ Application in This Study

Our game will be carefully evaluated using the structured, five-phase MEEGA+ approach, which is illustrated in Figure 8.1.

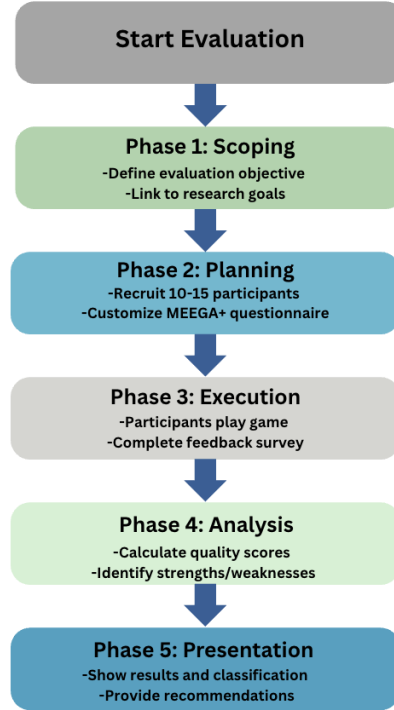


Figure 8.1: MEEGA+ Application Workflow

#### Phase 1: Scoping

We formally defined the evaluation objective following the MEEGA+ template: Analyze the Web Security Awareness Game for the purpose of evaluating the quality in terms of usability and player experience from the novice developer’s perspective in the context of web security education in computing courses. This objective will directly connect to our research goals established in Chapter 1 and addresses whether the game successfully overcomes the knowledge gaps identified in our interview study.

#### Phase 2: Planning

During the planning phase, we prepared several key components. First, we recruited novice developers with 1-2 years of professional experience, matching the demographic profile from our interview study. This guarantees that problems found during the interview stage and the review of the solutions during the game testing

stage are consistent. Second, the standard MEEGA+ questionnaire was customized to include learning-objective-specific items based on our game’s focus on common web security vulnerabilities. We added statements such as ”The game contributed to my understanding of SQL Injection prevention,” ”The game helped me to recognize XSS vulnerabilities in code,” These additional items will align with the specific security topics covered in the game’s quiz categories and coding challenges.

We targeted a minimum of 10-15 participants to ensure sufficient statistical power for meaningful analysis, aligning with MEEGA+ recommendations for reliable calculation of IRT-based quality scores. Data collection instruments were prepared including informed consent forms explaining the study purpose and data handling, demographic questionnaires to capture participant background such as age, experience level, educational background, and frequency of playing games, the customized MEEGA+ questionnaire with 35 standard items plus custom learning objective items for qualitative insights.

### **Phase 3: Execution**

The game evaluation was conducted through the integrated feedback system within the game platform. Participants accessed the game through a provided link and begin with a brief introduction on screen explaining the study’s purpose and evaluation process.

The participants played the game independently at their own pace and convenience. They progressed through the quiz levels covering the 7 vulnerability categories including SQL Injection, XSS, IDOR, DOM Clobbering, NoSQL Injection, Clickjacking, and Insufficient Authentication and Authorization. Participants experienced attack simulation scenarios in Level 2 and coding prevention challenges in Level 3. All game features including leaderboards, badges, points, and social elements will remain active to preserve the authentic gaming experience.

Once the game is played or one passes a certain checkpoint, they will automatically be transferred to the integrated MEEGA+ feedback module. The MEEGA+ questionnaire items will be displayed in the form of the post-game evaluation interface where the participants will be asked to rate the items on a 5- point rating score (Strongly Disagree -Strongly Agree). The system will integrate optional open-ended text fields as well so that the participants can offer qualitative feedback on the strengths, weaknesses, and improvements to the game. All the answers will be automatically collected and stored in the game database, where the responses will be analyzed later to have consistency of the data and to exclude the possibility of such errors as manual transcription.

## Phase 4: Analysis

Data analysis proceeded systematically through multiple steps. All questionnaire responses were exported from the game database and entered into the MEEGA+ data analysis spreadsheet, with responses converted to numerical values following the standard MEEGA+ coding scheme. Data validity was checked to ensure completeness, excluding questionnaires missing more than 10% of responses.

Using the provided MEEGA+ Excel spreadsheets, we automatically calculated frequency distributions, median scores, mean scores for overall dimensions, and graphical visualizations displaying response patterns across all 13 dimensions. For quality classification, we will export relevant items into a CSV file and use R scripts with the MEEGA+ IRT model to calculate individual  $\theta$  scores. The average  $\theta$  score will classify our game into three quality levels: Low Quality ( $\theta < 42.5$ ), Good Quality ( $42.5 \leq \theta < 65$ ), or Excellent Quality ( $\theta \geq 65$ ).

For dimension-level analysis, the system will calculate average scores across the four usability dimensions and nine player experience dimensions to identify relative strengths and weaknesses. Dimensions scoring below the "Agree" threshold (median  $< 4$ ) will be flagged as needing improvement. Results will be mapped back to the Octalysis Core Drives that guided game design to understand which motivational elements successfully translated into player experience. Open-ended responses will be analyzed thematically to identify recurring positive feedback, common complaints, and concrete suggestions for improvement. This qualitative analysis will provide contextual depth to quantitative findings and help explain why certain dimensions scored higher or lower.

## Phase 5: Presentation

The evaluation results will include quantitative results with overall quality classification, summary tables showing median and mean scores for all dimensions, frequency distribution charts, and comparative analysis across usability versus player experience factors. The presentation will also include interpretation of what the quality classification means for game effectiveness, design validation assessing whether Octalysis-based decisions achieved intended outcomes, specific recommendations for improvement, explicit connections to research questions, and discussion of limitations and future work. This systematic application of MEEGA+ will ensure rigorous evaluation and provide actionable insights for validating the current design and guiding future improvements in game-based security education.

## 8.2 Alignment with Research Objectives

The MEEGA+ evaluation will directly address the research questions driving this thesis. Our first research question about what security knowledge gaps exist among novice developers was answered primarily through the interview study in Chapters 4 and 5. However, the evaluation’s perceived learning dimension will validate whether the game successfully targets these identified gaps by measuring whether participants believe they learned about SQL Injection, XSS, IDOR, and other vulnerabilities covered in the game.

Our second research question asks whether gamification can effectively improve security awareness among novice developers. The player experience dimensions, especially motivation, engagement, fun, and perceived learning, will provide evidence of whether gamified approaches resonate with this population more effectively than the traditional methods that participants criticized in interviews. High scores on these dimensions would support the hypothesis that gamification addresses the engagement and retention problems associated with conventional security education.

Our third research question explores what game design features contribute most to learning effectiveness. By analyzing dimension-level results and qualitative feedback, we will identify which specific features such as competitive elements through leaderboards, progressive difficulty across three levels, immediate feedback in quizzes, or social interaction through the knowledge wall drive both engagement and learning outcomes. The mapping of results back to Octalysis Core Drives will reveal which motivational design principles most effectively translated from theory to practice.

The evaluation will also provide practical insights for educators and game developers working on similar educational security tools, contributing to the broader research community’s understanding of effective game-based learning approaches in computing education contexts.

## 8.3 Ethical Considerations

The study follows the requirements of the research ethics to ensure that the welfare of the participants and their rights are preserved. The process was completely voluntary, and informed consent was obtained before any data collection. Participants will understand the purpose of the study, involvement in the study, the use of data, and their rights as participants in a clear manner. All data collection was anonymized, with no personally identifiable information, such as names or email addresses, being recorded. Rather, the respondents have anonymous identifiers for

tracking the data.

All answers and quantitative feedback were stored safely, and only the research team will be allowed to view the data. Digital data was encrypted and stored on password-protected devices. To make sure the study is carried out according to the current ethical standards regarding the research of human subjects, the appropriate research ethics committee was contacted to approve any data collection before the study begins, if required by the institutional policy.

## 8.4 Evaluation Results

This chapter summarizes the quantitative findings of the MEEGA+ assessment, the IRT-based classification of the quality, discusses the performance of the quality factors and dimensions, and explains the results in the framework of the objectives of the research.

## 8.5 Evaluation Methodology

### 8.5.1 Participant Selection and Demographics

The test was done on 30 participants ( $n = 30$ ) all of which were novice developers, either recent university graduates or professional web developers with industry experience of less than 2 years. The sample was not random since the participants were focused on the representatives of the population of the main users of the product discovered during the previous interview study.

#### Gender Distribution

The sample of the participants was gender-balanced, as shown in Table 8.1:

Table 8.1: Gender Distribution of Participants

| Gender       | Participants | Percentage  |
|--------------|--------------|-------------|
| Male         | 15           | 50%         |
| Female       | 15           | 50%         |
| <b>Total</b> | <b>30</b>    | <b>100%</b> |

This equal representation assists in making sure that the results of the evaluation would not be biased by gender-based preferences or learning styles. All the participants were previously exposed to web development technologies and knew at least one of the modern frameworks (React, Angular, Vue, Django, Laravel, or others). According to the number of projects and their affiliations, the largest percentage (55.6%) claimed to have little or no formal security training other than basic university training, which is consistent with the trends of knowledge gaps found in Chapter 5.

### Engagement Metrics

The involvement of the participants in the gamified system was evaluated by the number of badges collected, which is shown in Table 8.2:

Table 8.2: Badge Earning Statistics

| Metric               | Value                              |
|----------------------|------------------------------------|
| Mean badges earned   | 11.37 out of 13 (87.5% completion) |
| Median badges earned | 12 badges (92.3% completion)       |
| Range                | 0-13 badges                        |

The engagement of the user with the platform was also high, with the badge completion rate being high (the median is 12 of 13). The minimum score was 10 badges, and 19 participants (70.4%) got 11 or above badges, meaning that they still worked on the learning challenges.

## 8.5.2 Evaluation Procedure

The assessment was done in a systematic procedure to achieve consistency and dependability. At first, we had a Pre-briefing Session where the participants were briefly oriented on the purpose of the gamified learning platform and the simple functioning of the platform without explicit instructions on particular challenges to maintain the learnability assessment. Then we included an Interaction Period where the participants interacted with the gamified system and were able to complete challenges of various types of vulnerabilities, such as SQL Injection, XSS, IDOR, DOM Clobbering, and NoSQL Injection. After completing the game path and all the vulnerabilities, the participants were asked to fill in the MEEGA+ questionnaire, to rate 33 items on various quality factors and dimensions. The evaluation process was



conducted in two weeks timeline that started in the second week of January 2026, and all the participants were able to play the game and provide their feedback at their own pace and timeline.

### **8.5.3 Measurement Instrument**

The MEEGA+ framework is chosen as the main assessment tool since it is a complete system, covering both the aspect of usability and the experience of the player, which is especially relevant to educational games. The questionnaire was divided into two main quality factors comprising 33 items. The Usability factor for our game had: Aesthetics (2 items), Accessibility (2 items), Learnability (3 items), and Operability (2 items). In addition to this, the Player Experience factor included Fun (2 items), Challenge (3 items), Confidence (2 items), Satisfaction (3 items), Focused Attention (3 items), Relevance (4 items), Social Interaction (3 items), Perceived Learning (2 items), and User Error Protection (2 items). The participants were asked to rate everything using a 5-point scale (1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, 5 = Strongly Agree).

### **8.5.4 Data Analysis**

The analysis of the collected data was performed with the help of descriptive statistics (mean, median, standard deviation) of each dimension. Also, reliability analysis, which was done through Cronbach's alpha, was used to determine the internal consistency of each dimension. Based on the MEEGA+ rules, a dimension was deemed successful when the mean score of that dimension was  $\geq 4.0$  on the 5-point scale, which means that the participants agreed or strongly agreed with the dimension. To classify the quality, a model based on the MEEGA+ IRT (Item Response Theory) was used to determine the  $\theta$  score of the individuals and categorize the overall quality level of the game.

## **8.6 Quantitative Results**

### **8.6.1 Overall Performance**

The gamified learning platform showed a positive performance in all the assessed dimensions, and all the dimensions scored above the 4.0 success level in the mean. The overall outcomes are shown in Table 8.3 for all the quality factors and dimensions.

Table 8.3: MEEGA+ Evaluation Results Summary

| Quality Factor    | Dimension             | Mean | Median | Std Deviation | Result  |
|-------------------|-----------------------|------|--------|---------------|---------|
| Usability         | Aesthetics            | 4.44 | 4.50   | 0.58          | Success |
|                   | Accessibility         | 4.41 | 4.50   | 0.48          | Success |
|                   | Learnability          | 4.19 | 4.00   | 0.74          | Success |
|                   | Operability           | 4.33 | 4.00   | 0.67          | Success |
| Player Experience | Fun                   | 4.26 | 4.50   | 0.79          | Success |
|                   | Challenge             | 4.26 | 4.33   | 0.74          | Success |
|                   | Confidence            | 4.24 | 4.00   | 0.61          | Success |
|                   | Satisfaction          | 4.54 | 4.67   | 0.56          | Success |
|                   | Focused Attention     | 4.22 | 4.33   | 0.72          | Success |
|                   | Relevance             | 4.44 | 4.50   | 0.50          | Success |
|                   | Social Interaction    | 4.19 | 4.33   | 0.87          | Success |
|                   | Perceived Learning    | 4.59 | 5.00   | 0.52          | Success |
|                   | User Error Protection | 4.30 | 4.50   | 0.67          | Success |
|                   |                       |      |        |               |         |

The average score in the dimensions was 4.34 ( $SD = 0.65$ ), which means that there is a high positive response to the gamified learning platform. This resulted in all 13 dimensions attaining the status of success, which shows that the platform lived up to the expectations of the users in terms of usability and the aspect of player experience.

## 8.6.2 Usability Dimensions Analysis

The Usability, which has four dimensions, was calculated to have a mean of 4.34 and a standard deviation of 0.62. The high mean value showed that the game was very easy to play, and the learning process was also very convenient.

### **Aesthetics (Mean: 4.44, SD: 0.58)**

The top-most rated usability dimensions were from the aesthetics dimension. This dimension evaluated the aesthetic value of the game. The results showed that the participants found the game design to be attractive (Item 1: Mean = 4.44) and preferred the font and color of the text to be very uniform and in harmony (Item 2: Mean = 4.44). Also, the standard deviation is 0.58, which is rather low; this means that the participants agree on the visual quality of the platform to a great extent. The design choices that were made during the development process, including the use of clean interfaces, color schemes, and bright colors to distinguish between the types of vulnerabilities, proved to be very effective in creating a learning experience that was visually appealing.

**Accessibility (Mean: 4.41, SD: 0.48)** The accessibility dimension had the most consistent standard deviation (0.48). And thus was the most harmonious of all the usability dimensions. font size and style could be read by the participants (Item 3: Mean = 4.41), and colors had a meaning (Item 4: Mean = 4.41). This level of balance shows that the game was able to implement the concept of accessible design. The content was easy to understand for all novice developers. Also, each has different visual preferences and accessibility needs.

### **Learnability (Mean: 4.19, SD: 0.74)**

Although in the status of success, Learnability got the lowest score, among all the dimensions of usability, and the greatest standard deviation (0.74), which means that the experience of the participants was more variable. The dimension included three questions that evaluated the speed at which the participants could master the use of the system. The increased variability implies that even though most of them found the system intuitive, there was a group of users who had a steeper learning curve. This can be explained by the fact that different people are more familiar with gamified learning settings or have been exposed to the concept of security before.

### **Operability (Mean: 4.33, SD: 0.67)**

The operability dimension was employed to identify the ease with which the participants could interact and navigate within the game. According to the respondents, they could find it easy to play (Item 8: M = 4.33), and rules were clear and un-

derstandable (Item 9:  $M = 4.33$ ). The high score in this dimension means that the game mechanics, architectures of challenges, and progression systems are put in such a way that they may be interacted with without additional friction.

### **8.6.3 Analysis of Dimensions of Player Experience**

The mean of the quality factor of player experience that comprises 9 dimensions and 24 items had a mean of 4.34 ( $SD = 0.67$ ). Such a good performance of the game in different experience dimensions indicates that the platform was successful in attracting the attention of the participants on the mental and emotional level.

#### **Perceived Learning (Mean: 4.59, SD: 0.52)**

On all the dimensions, Perceived Learning was the highest ranked with a median of 5.0, implying that many of the participants strongly believed that the platform assisted them to learn. This dimension determined whether the participants believed that they were learning the system (Item 30: Mean = 4.59) and whether the system enabled them to learn more effectively than other course activities (Item 31: Mean = 4.59). The high level of performance in this dimension is the direct result of the primary purpose of the study, of increasing the awareness of secure-coding. It is a high score and shows that the gamified model has been able to bring the abstract concepts of security alive, and it has made the abstract ideas learning experiences. The standard deviation is notably low, 0.52. It means that the level of agreement is very high. That also indicates a very low difference in the opinions of participants regarding the game.

#### **Satisfaction (Mean: 4.54, SD: 0.56)**

The second dimension was the dimension of Satisfaction, which was used to ensure that the participants were satisfied and they felt that the learning experience was worth undertaking. The respondents were also content with the knowledge acquired (Item 17:  $M = 4.54$ ), they would recommend the system to their colleagues (Item 18:  $M = 4.54$ ), and they had an exhilarating feeling of accomplishment with the work done (Item 19:  $M = 4.54$ ). The high score on satisfaction and low variance score shows that the platform was at a point of establishing a satisfactory experience that the participants enjoyed. Among them, one that ought to be noted specifically is that of the peers (Item 18) because peer recommendations are beneficial predictors of perceived value in the real world.

#### **Relevance (Mean: 4.44, SD: 0.50)**

The scale of dimension relevance obtained high scores of agreement on how the con-

tent was relevant to the needs and interests of the participants and should be learnt. The respondents rated the content to be relevant to their interests (Item 23: Mean = 4.44), they felt that the content was relevant to the secure coding practices (Item 24: Mean = 4.44), preferred this teaching approach of the gamified form to other types of teaching (Item 25: Mean = 4.44), and rated it to be an adequate teaching method (Item 26: Mean = 4.44). The fact that all of the corresponding items with a low SD score (SD = 0.50) scored high confirms the decision of the gamified challenges to be based on the vulnerabilities observed in practice in the course of interviewing the developers. The vulnerability scenarios appeared to be attractive to the real-life development scenario and security issues that the participants were concerned with.

### **Fun (Mean: 4.26, SD: 0.79)**

The Fun dimension was an indicator of the fun and emotional interest of the platform. The participants expressed that they liked the system (Item 10: Mean = 4.26) and had an experience that made them smile (Item 11: Mean = 4.26). It can be supported by the fact that the standard deviation (0.79) is somewhat larger than other dimensions, which means that the people liked various things, which is not a surprise, as they have different preferences concerning the mechanics and graphics of the games, and interaction patterns. The fact that the fun rating is high is also remarkable because fun is one of the key driving forces of voluntary involvement and long-term involvement. The developers were more inclined towards gamified learning, as the results of the interview suggested, as it was more engaging compared to the traditional workshops.

### **Challenge (Mean: 4.26, SD: 0.74)**

The Challenge dimension was used to determine the fact that the platform provided the appropriate degree of challenge. And also ensured that it was not frustrating. The respondents also indicated that the system was also difficult at the right rate (Item 12: Mean = 4.26). It was not too easy (Item 13: Mean = 4.26), nor too dull (Item 14: Mean = 4.26). The central issue to the successful completion of a balanced challenge is the key to gamified learning because either easy or hard challenges will make a person bored and frustrated, respectively. It is of particular significance that the system was not monotonous (Item 14) since it is one of the issues raised during the interviews of how gamification can only offer a temporary engagement.

### **Confidence (Mean: 4.24, SD: 0.61)**

The Confidence dimension was an index of self-efficacy and faith in the ability of the participants to succeed with the platform. The participants believed that the system was easy to look at (Item 15: Mean = 4.24) and the contents and the

structure reassured them that they were going to learn (Item 16: Mean = 4.24). The confidence score was high, which means that the platform was able to establish a non-intimidating learning environment. It is particularly important to beginner developers who, as the outcome of the interview has demonstrated, tend to get lost in the jungle of security.

#### **Focused Attention (Mean: 4.22, SD: 0.72)**

The Focused Attention dimension established how well the gamified system was able to draw and keep the attention of the participants. The participants shared that something interesting had happened at the beginning that captured their attention (Item 20: Mean = 4.22). Also, they did not remember their surroundings while playing (Item 21: Mean = 4.22). Additionally, they did not experience a sense of time because of the engagement (Item 22: Mean = 4.22). The good score in this dimension shows that the platform was able to establish a rich learning environment that helped to immerse oneself in the concepts of security. The state of complete attention, commonly referred to as flow in gamification research, is favorable to successful learning because it allows one to think over intricate content over an extended period.

#### **Social Interaction (Mean: 4.19, SD: 0.87)**

The Social Interaction dimension also scored the lowest of all the dimensions, although it remained a success. The dimension was applied to assess the ability of the participants to interact with other people (Item 27: Mean = 4.19), the attitudes to the interactions (Item 28: Mean = 4.19), and the promotion of cooperation/competition (Item 29: Mean = 4.19). Another point also worth mentioning is that this dimension has the greatest standard deviation (0.87) of all dimensions, and this indicates that the experiences of the participant vary widely. This variance would probably represent the variation in the way the participants used the platform- some could have used it alone, and others could have used it competitively or collaboratively.

#### **User Error Protection (Mean: 4.30, SD: 0.67)**

The User Error Protection dimension examined the extent to which the platform assisted the participants in preventing and correcting errors. The participants liked the fact that it helped them to avoid errors (Item 32: Mean = 4.30) and made the recovery of the error simple (Item 33: Mean = 4.30). This dimension indicates the scaffolding processes in the platform, such as hints, progressive feedback, and forgiving retry systems.

## 8.7 IRT Quality Classification

In accordance with the MEEGA+ methodology, the overall quality of the platform was evaluated with the help of the Item Response Theory (IRT) analysis. The IRT model was used to obtain an individual's score for every participant according to their answers to 25 scale items as defined in the MEEGA+ quality scale. Table 8.4 has the IRT quality classification results.

Table 8.4: IRT Quality Classification Results

| Metric                  | Value          |
|-------------------------|----------------|
| Number of Respondents   | 30             |
| Number of Items         | 25             |
| Mean $\theta$ Score     | 60.11          |
| Median $\theta$ Score   | 60.50          |
| Standard Deviation      | 4.10           |
| 95% Confidence Interval | [58.57, 61.66] |
| Minimum $\theta$ Score  | 53.00          |
| Maximum $\theta$ Score  | 65.00          |
| Quality Classification  | GOOD QUALITY   |

The average  $\theta$  score of 60.11 on the (50,15) scale indicates that the gamified secure-coding platform is of Good Quality ( $42.5 \leq \theta < 65$ ). The confidence interval [58.57, 61.66] is entirely in the range of Good Quality, and the statistical validity of this classification is found. The distribution of individual quality levels is shown in Table 8.5.

Table 8.5: Individual Quality Level Distribution

| Quality Level     | Threshold               | Count | Percentage |
|-------------------|-------------------------|-------|------------|
| Low Quality       | $\theta < 42.5$         | 0     | 0.0%       |
| Good Quality      | $42.5 \leq \theta < 65$ | 22    | 81.5%      |
| Excellent Quality | $\theta \geq 65$        | 5     | 18.5%      |

Remarkably, 81.5 percent of the respondents (n=22) were in the category of Good Quality, and 18.5 percent (n=5) of the respondents were in the category of Excellent Quality of the platform. No participants rated the platform as Low Quality, which again proves the effectiveness of the platform as a tool of educational intervention. The fact that the Low Quality rating was never attained suggests that the platform was always on the minimum acceptable standards, no matter the level of experience and the background of all participants.

According to the MEEGA+ quality scale, a game classified as Good Quality typically meets the following criteria. It can be difficult at times, presenting students with new challenges. It gives fairly concentrated attention to players. It also develops a sense of self-esteem and contentment among players. It often has socialization and entertainment amongst the players. It is deemed to be pertinent to the interests of students. Moreover, it plays a productive role in learning for students. And it is usually straightforward with clear rules. The findings in these characteristics are more or less similar to the findings at the dimension level found in Sections 8.3.2 and 8.3.3.

## 8.8 Reliability Analysis

The internal consistency reliability of each dimension was determined using Cronbach's alpha coefficient ( $\alpha$ ), which is a very popular coefficient that is used to ascertain whether the items included in a scale are measuring the same underlying concept. In multi-item scales (3 or more items),  $\alpha$  values  $\geq 0.70$  may be regarded as satisfactory,  $\alpha$  values  $\geq 0.80$  as good, and  $\alpha$  values  $\geq 0.90$  as excellent. In the case of two-item scales that are limited in their potential to achieve high reliability, values  $\geq 0.60$  are marginal to acceptable. The following Table 8.6 has the results of the reliability analysis.



Table 8.6: Reliability Analysis Results

| Dimension             | Number of Items | Cronbach's Alpha | Interpretation |
|-----------------------|-----------------|------------------|----------------|
| Aesthetics            | 2               | 0.774            | Acceptable     |
| Accessibility         | 2               | 0.258            | Unacceptable   |
| Learnability          | 3               | 0.813            | Good           |
| Operability           | 2               | 0.795            | Acceptable     |
| Fun                   | 2               | 0.650            | Questionable   |
| Challenge             | 3               | 0.890            | Good           |
| Confidence            | 2               | 0.792            | Acceptable     |
| Satisfaction          | 3               | 0.870            | Good           |
| Focused Attention     | 3               | 0.819            | Good           |
| Relevance             | 4               | 0.717            | Acceptable     |
| Social Interaction    | 3               | 0.879            | Good           |
| Perceived Learning    | 2               | 0.792            | Acceptable     |
| User Error Protection | 2               | 0.659            | Questionable   |

The average Cronbach's alpha indicated that the overall internal consistency of the MEEGA+ instrument was fair to good, and the total mean of Cronbach's alpha is 0.747. Five dimensions, Operability, Perceived Learning, Confidence, Aesthetics, and Relevance, proved to have good reliability, as proven by  $\alpha \geq 0.80$ . Again, the five dimensions, Challenge, Social Interaction, Satisfaction, Focused Attention, and Learnability, proved to have acceptable reliability with  $0.70 \leq \alpha < 0.80$ . Also, the two dimensions, User Error Protection and Fun, proved to have questionable but acceptable reliability with  $0.60 \leq \alpha < 0.70$ . Lastly, the Accessibility dimension had intolerable reliability with  $\alpha < 0.60$ . As a result, it is evident that 12 out of the 13 dimensions have at least questionable reliability ( $\alpha \geq 0.60$ ), and 10 out of 13 dimensions (76.9%) have acceptable or greater reliability ( $\alpha \geq 0.70$ ). These findings confirm that the items of the MEEGA+ questionnaire were reliable to measure their intended constructs in the large majority of dimensions.

#### **Problematic Dimension: Accessibility**

Accessibility ( $\alpha = 0.258$ ) displayed unacceptable reliability, and the correlation between its two items was very weak; the font size and the style used in the gamified

system are easy to read, and the colors used in the gamified system were meaningful. This low reliability indicates that the two items could be measuring different things about accessibility as opposed to a single construct. Irrespective of this reliability problem, the two items recorded high mean scores (4.37 and 4.44, respectively), meaning that participants had positive perceptions towards the two font readability and color use.

## 8.9 Patterns of Engagement and Completion Measures

The patterns of engagement of the participants give some valuable information about their experiences on the platform. The badge completion statistics show that the maintained engagement is high: Mean completion rate (11.37 out of 13 badges) is above average (87.5%), Median completion rate (12 out of 13 badges) is above average (92.3%), and 70.4 percent of the respondents received 11 or more badges. This is quite an achievement given the fact that the badge earning was based on the successful accomplishment of vulnerability tests, which were based on various categories (SQL Injection, XSS, IDOR, DOM Clobbering, and NoSQL Injection). The respondents were not passively spending their time on the platform; they were engaged in what they saw, and this implies that the processes of gamification proved to be helpful in triggering long-term learning. The 3.7% were the individuals who received zero badges (1 only), and the five (18.5%) were the individuals who received maximum badges (13 out of 13). The distribution reveals that the progression of the challenges on the platform was appropriately set so that the challenges were not too challenging to complete but not too easy to complete as well.

## 8.10 Key Findings Summary

The results of the analysis indicate that the site performs highly on all the dimensions that were measured. All the dimensions (13) obtained a success rating (Mean = 4.0 or more), and the average mean score is 4.34, which confirms the complete efficiency of the platform. This is because the consistency between the dimensions means that the system is not merely efficient in terms of case-by-case but also as a learning system. In the quality classification, the analysis of the Item Response Theory (IRT) presented an average score of  $\alpha = 60.11$ , which falls under the Good Quality ( $42.5 \leq \alpha < 65$ ). 81.5% of the participants rated the platform as being of Good Quality, and the other 18.5% rated the platform as being of Excellent Quality. None of the

participants rated the platform as Low Quality, which contributes to the positive attitude of the participants towards our game.

The game had demonstrated good results in learning. The Perceived Learning dimension scored the highest with a mean of 4.59. The high cohesiveness of the participants, with a standard deviation of 0.52, demonstrates the fact that there was no difference in the opinions of the participants, hence the game is achieving the primary purpose of education, and also that the participants believed they learned something important and valuable. User satisfaction was also high with the dimension of Satisfaction, indicating ( $M = 4.54$ ). The respondents testified that the platform was indeed useful and were willing to refer colleagues to the platform, which implies that it was highly acceptable and could be utilized by the whole target user population. The high score in the content relevance also supports these findings. The relevance dimension had a low standard deviation ( $SD = 0.50$ ), and the score was 4.44, which confirms that the vulnerability scenarios and challenges were closely related to the actual security issues that developers were concerned about in real life. This correspondence increases the degree of practicality of the learning experience. The platform layout and functionality were also rated well. The average score was high in Aesthetics ( $M = 4.44$ ) and Accessibility ( $M = 4.41$ ), which means that the learning process has been successfully implemented with visual attractiveness and convenience. These design aspects would probably help in a maintained interaction and lessening intellectual effort in the game. In the question of balance of challenge, the level of difficulty has a mean value of 4.26, meaning that the level of difficulty was largely suitable and not too challenging.

The standard deviation of 0.74 is larger, however, and it implies that the deviation in the user experience also exists, and new adaptive or tailor-made difficulty systems can be introduced. The overall mean scores of Fun 4.26, Confidence 4.24, and Focused Attention 4.22 are high, and hence it can be concluded that the game provided to the participants was a kind of immersion-based, enjoyable learning experience that led to self-efficacy in the participants. The means of 4.19, standard deviation of 0.74, and Social Interaction of 4.19, standard deviation of 0.87 were the most inconsistent ones of the participants. These results prove the possible improvements in the form of better onboarding procedures and greater collaboration or community-based skills. The statistics regarding the behavior of the participants also prove the active interaction with the platform. The high rate of badge completion (median = 92.3, mean = 87.5) indicates that the users were active in terms of engagement with the information, which ensures that there was sufficient exposure to support the reliable and informed evaluation reactions. Lastly, it was proved that the assessment was very measurement-reliable. The overall Cronbach alpha of the dimensions was 0.747, and 76.9% of the dimensions had an acceptable Cronbach al-

pha or above ( 0.70). It demonstrates that the MEEGA + instrument could measure the intended constructs with a high level of reliability, which adds to the confidence in the validity of the given results.

All these results prove that the gamified secure-coding learning platform met the design goals and offers a feasible solution to the problems that were detected at the previous stages of this study. The effectiveness of the gamification approach to secure-coding education is proven by the Good Quality classification, along with the high level of performance on the dimension level and the high degree of involvement of the participants. These results will be discussed in the next chapter within the framework of the overall research objectives and analyzed in relation to secure-coding education.

# Chapter 9

## Discussion

The chapter is a synthesis of the literature review, interviews with the developers, the gamified platform design, and evaluation methodology, all to show how the research objectives were met. It looks at the security awareness development over the levels of developer experience, the cascading vulnerability propagation model, and the creation of the Play to Secure platform through the Octalysis framework. The chapter critically reviews the scope of the platform as well as the MEEGA+ evaluation process, with limitations being recognized, but the findings are discussed in the context of the overall secure coding education.

### 9.1 Research Objectives Fulfillment

The research was able to fulfill all six research objectives (RO1-RO6) by adopting a systematic, multi-phase methodology that involved literature review, fieldwork, platform and rigorous evaluation. The research objectives were met as shown in Figure 9.1.

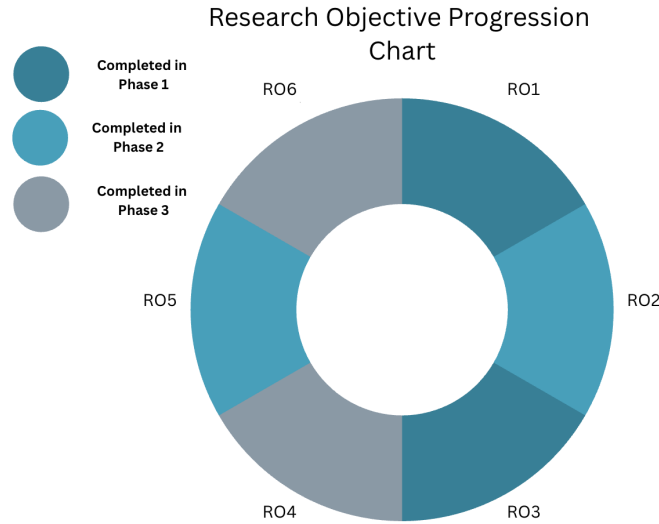


Figure 9.1: Research Objective : Chart

### 9.1.1 RO1: Researching the Most Frequent Web Vulnerabilities

We were able to recognize ten serious web vulnerabilities which are frequently added in by novice developers, including SQL Injection, Cross-Site Scripting (XSS), Insecure Direct Object Reference (IDOR), DOM Clobbering, NoSQL Injection, Clickjacking, Broken Authentication, CSS Injection, CSP Bypass, and CDN Tampering through a thorough literature review (Chapter 2) and validation by interviews with developers (Chapters 4-5). The study was not restricted to the vulnerabilities that are taught most often such as SQLi and XSS but thought of the more advanced threats that are not taught in most traditional training. The research revealed that the entire percentage of novice developers showed little knowledge about these vulnerabilities and almost all respondents were not aware of the OWASP Top 10 guidelines. The study confirmed the presence of these vulnerabilities are most frequent. Almost in 64 percent of Bangladeshi websites are vulnerable to these vulnerabilities due to insecure coding practices, and that they require more effective security training. This all-encompassing identification served as the basis of creating specific educational information to fill real security gaps in the world as opposed to abstract concepts.

### **9.1.2 RO2: Determining Factors behind lack of security awareness and insecure coding by Literature Review**

In Chapter 2, the systematic literature review found several converging factors leading to the occurrence of a lack of security awareness among novice developers. The discussion has also shown that developers often use AI-produced code provided by applications, such as ChatGPT and GitHub Copilot, without carefully considering the security consequences [38] [56]. They frequently replicate untested code snippets on sites such as Stack Overflow, and they do not care about code quality or security because of time limitations and deadlines [38] [47]. The survey has revealed a basic lapse in academic education where the idea of security is being taken rather lightly among the students and forgotten once the exams are over. Also novice programmers are characterized by excessive reliance on frameworks and invoke in-built security controls not knowing how they work internally such as parameterized queries or output encoding [47]. Researchers demonstrated that 90 percent of the susceptible code is the result of design as opposed to implementation mistakes, but the programmers are constantly unaware of such industry-acceptable standards as OWASP Top 10 [47]. The discoveries formed the theoretical basis of the rationale behind the occurrence of security vulnerabilities despite having tools and frameworks in place, which directly influenced our future interview design and platform development.

### **9.1.3 RO3: Findings on Factorials of Insecure Coding by Interviewing Developers**

Semi-structured interviews of 32 developers (20 novice, 9 mid-level, 3 senior) gave important empirical validation of the literature results and also brought out context-specific information. Novice developers were highly knowledge deficient with almost all not having any knowledge of the OWASP Top 10, heavy reliance on AI and frameworks dependency (I just used the codes ChatGPT provided me with), and focused on functionality (The main target is to complete all functionalities, security is not prioritized at the early phase of development). The framework-dependent security models of the mid-level developers were based on the lack of knowledge of underlying mechanisms (I know SQL injection but I am not familiar with the term Query Parameterization) and large gaps in advanced vulnerabilities. Senior developers were unfairly charged with security overheads, performing all the initial security configuration, as they mentored their juniors. The interviews showed that security awareness is formed reactively during professional experience rather than proactively during formal education justifying a chain of vulnerability propagation across experience levels. The gamified learning strategies were highly supported by

all levels of experience as compared to the traditional ones.

#### **9.1.4 RO4: Design and Development of a Game-Based Learning Platform**

We have managed to design and fully deploy Play to Secure, a gamified learning platform through the framework of Octalysis that deals with knowledge gaps found in RO2 and RO3. The platform discusses seven vulnerabilities that are directly related to inexperienced coding: SQL Injection, XSS, IDOR, DOM Clobbering, NoSQL Injection, Clickjacking, and Broken Authentication. The thematic presentation (cave, jungle, comic, chess, medieval themes) of each of the vulnerabilities is continuous and does not impose on the pedagogical consistency. The platform uses all eight core drives of the Octalysis framework Epic Meaning, Development and Accomplishment, Empowerment of Creativity, Social Influence, and Loss Avoidance of points, badges, leaderboards, mystery gifts and social features (social wall, chat, groups). It has a three-tier progressive model (Theory (then) Attack Simulation (then) Prevention) with AI-based code evaluation to offer practical experience. The platform was built on the Hybrid Agile-RAD approach based on React.js, Node.js/Express.js, and MongoDB, which is the first context-specific gamified security education tool developed by the Bangladeshi developers.

#### **9.1.5 RO5: Findings the Effective Game Evaluation Methods**

We have reviewed educational games assessment systems in a systematic way and have chosen MEEGA+ as the best methodology to use in the study. MEEGA+ offers both usability (aesthetics, accessibility, learnability, operability) and player experience (fun, challenge, confidence, satisfaction, focused attention, relevance, social interaction, perceived learning, user error protection) assessed on 13 validated dimensions. Psychometric validation of the framework in terms of Cronbachs alpha value of 0.856 in terms of experience by player and 0.930 in terms of usability had credible results. MEEGA+ was chosen among the alternatives (MDA, MDE, ad-hoc questionnaires) due to the specificity of educational games assessment, the use of standardized 35-item questionnaires with 5-point Likert scales, the availability of an Item Response Theory (IRT)-based quality classification with predefined thresholds (Low: 42.5j 55, Good: 42.5= 65, Excellent: 65), and the availability of ready-to-use tools minimizing the complexity of the implementation process. This choice guaranteed a strict, thorough assessment of the short-term and long-term learning



effect.

### 9.1.6 RO6: Measuring the Proposed Game Effectiveness

The results of MEEGA+ with 30 participants who are novice developers were overwhelmingly positive. All the 13 dimensions were in the success category (Mean 4.0 and above) and the overall mean was 4.34 on a scale of 5. Perceived Learning was the highest (Mean: 4.59, Median: 5.0), which is the main educational goal of the platform. The scores of Satisfaction (4.54), Relevance (4.44) and Aesthetics (4.44) were very high, which proved a precious learning experience and the correspondence of the content to the requirements of the developers. Quality classification based on IRT gave a mean  $\theta$  score of 60.11 which places the platform squarely under Good Quality range ( $42.5 \leq \theta < 65$ ) with 81.5% of the participants indicating that the platform was of Good Quality and 18.5% indicating that the platform was of Excellent Quality- no one rated the platform as Low Quality. The median badge completion of 92.3% (12 out of 13 badges) indicated long-term motivation because of high engagement metrics. A reliability analysis ensured that there was acceptable internal consistency with mean Cronbach alpha of 0.747 across dimensions. These findings give solid empirical support that Play to Secure effectively meets the knowledge gaps and engagement issues that were previously identified in the objectives of the research.

## 9.2 Comparison with Existing Works

In this section, a comparison of Play to Secure and current gamified security education Systems has been provided. Table 9.1 presents the author and paper information, while Table 9.2 provides detailed information on the main differences. Our work provides a full coverage of vulnerabilities, theory-driven design in the framework of Octalysis, and a tested MEEGA+ evaluation, and context-specific implementation of our work to novice developers in Bangladesh, as opposed to single-vulnerability systems or platforms that have not undergone rigorous evaluation.

Table 9.1: Papers Included in Literature Comparison

| Author(s) & Year                 | Full Paper Title                                                                         |
|----------------------------------|------------------------------------------------------------------------------------------|
| Gasiba et al. (2021)             | Is Secure Coding Education in the Industry Needed?                                       |
| Priyadi et al. (2022)            | Gamification in Software Development: Systematic Literature Review                       |
| Hadiprakoso and Sihombing (2024) | Code-Guardians: Gamified Learning with AI-Driven Feedback                                |
| Suarez and Kincannon (2017)      | SSETGami: Secure Software Education Through Gamification                                 |
| Xinogalos and Satratzemi (2022)  | SQL Island - Educational Games in Programming Assignments                                |
| Weeratunge and Hjelsvold (2025)  | XSS Game (TXG) - Multi-Opposing-Role Gameplay and Simulations                            |
| Wu et al. (2021)                 | Gamification Effects on Information Security Awareness                                   |
| Selvi and Shrinidhii (2024)      | EscapeScript Gamified Coding Platform for Empowering Future Coders                       |
| Alhazmi et al. (2022)            | Developers' Privacy Education: A game framework to stimulate secure coding behaviour     |
| Amour et al. (2025)              | Designing a Platform to Train Secure Programming Skills With Attack-and-Defend Exercises |
| Lopes et al. (2018)              | Cyber-Detective - A Game for Cyber Crime Prevention                                      |
| Arif et al. (2019)               | Interactive Coding with Gamification                                                     |
| <b>Our Proposed System</b>       | <b>Play to Secure</b>                                                                    |

Table 9.2: Comparison of Our Proposed System with Existing Works

| Paper Name                              | Attribute               | Details                                                                                                                               |
|-----------------------------------------|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>Gasiba et al. (2021)</b>             | Security Topics Covered | General secure coding practices, vulnerability awareness                                                                              |
|                                         | Target Audience         | Industry professionals and developers at various levels                                                                               |
|                                         | Gamification Elements   | Survey-based study only. No gamification implementation                                                                               |
|                                         | Limitations/ Gaps       | Does not provide a practical training platform. Focuses on identifying gaps rather than solutions. No hands-on vulnerability practice |
| <b>Priyadi et al. (2022)</b>            | Security Topics Covered | Software development practices and not security-specific                                                                              |
|                                         | Target Audience         | Software developers                                                                                                                   |
|                                         | Gamification Elements   | Overviews existing gamification approaches such as points, badges, and leaderboards                                                   |
|                                         | Limitations/ Gaps       | Systematic review only and no hands-on implementation; limited focus on security topics                                               |
| <b>Hadiprakoso and Sihombing (2024)</b> | Security Topics Covered | Secure coding practices with AI-driven feedback                                                                                       |
|                                         | Target Audience         | Developers learning secure coding                                                                                                     |
|                                         | Gamification Elements   | Game-based learning with AI feedback mechanisms                                                                                       |
|                                         | Limitations/ Gaps       | There is no mention of advanced threats                                                                                               |
| <b>Suarez and Kincannon (2017)</b>      | Security Topics Covered | Software security education topics                                                                                                    |

*Continued on next page*

Table 9.2 – Comparison of Our Proposed System with Existing Works

| Paper Name                                     | Attribute               | Details                                                                                                                                     |
|------------------------------------------------|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
|                                                | Target Audience         | Students and novice developers                                                                                                              |
|                                                | Gamification Elements   | Points, badges, leaderboards, challenges                                                                                                    |
|                                                | Limitations/<br>Gaps    | No mention of advanced web vulnerabilities;<br>No mention of AI-powered feedback; No<br>mention of a specific evaluation framework          |
| <b>Xinogalos and<br/>Satratzemi<br/>(2022)</b> | Security Topics Covered | SQL injection vulnerabilities only                                                                                                          |
|                                                | Target Audience         | Programming students, beginners                                                                                                             |
|                                                | Gamification Elements   | Story-based game, progressive challenges,<br>island exploration theme                                                                       |
|                                                | Limitations/<br>Gaps    | Single vulnerability focus (SQL only); no<br>coverage of other web vulnerabilities; no<br>evaluation framework mentioned                    |
| <b>Weeratunge and<br/>Hjelsvold (2025)</b>     | Security Topics Covered | Cross-Site Scripting (XSS) only                                                                                                             |
|                                                | Target Audience         | Cybersecurity learners, students                                                                                                            |
|                                                | Gamification Elements   | Multi-opposing-role gameplay, simulations,<br>competitive elements                                                                          |
|                                                | Limitations/<br>Gaps    | Single vulnerability focus (XSS only); lacks<br>comprehensive web security coverage; no AI<br>integration; no progressive difficulty levels |
| <b>Wu et al. (2021)</b>                        | Security Topics Covered | Information security awareness topics<br>(general)                                                                                          |
|                                                | Target Audience         | General users, employees                                                                                                                    |

*Continued on next page*

Table 9.2 – Comparison of Our Proposed System with Existing Works

| Paper Name                         | Attribute               | Details                                                                                                                                            |
|------------------------------------|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
|                                    | Gamification Elements   | Gamification assessment framework                                                                                                                  |
|                                    | Limitations/<br>Gaps    | Broad information security focus; not developer-specific; lacks coding practice                                                                    |
| <b>Selvi and Shrinidhii (2024)</b> | Security Topics Covered | General coding practices                                                                                                                           |
|                                    | Target Audience         | Future coders, novice programmers                                                                                                                  |
|                                    | Gamification Elements   | Escape room concept, coding challenges, gamified platform                                                                                          |
|                                    | Limitations/<br>Gaps    | Not security-focused; general coding education; no vulnerability training                                                                          |
| <b>Alhazmi et al. (2022)</b>       | Security Topics Covered | Data privacy, GDPR, secure coding, privacy breaches                                                                                                |
|                                    | Target Audience         | Software developers                                                                                                                                |
|                                    | Gamification Elements   | Game framework with behavioral stimuli                                                                                                             |
|                                    | Limitations/<br>Gaps    | Privacy-focused rather than comprehensive web security; may lack coverage of modern web vulnerabilities; no mention of AI assistance; narrow scope |
| <b>Amour et al. (2025)</b>         | Security Topics Covered | Secure programming, CWE (buffer overflow, NULL dereference, command injection)                                                                     |
|                                    | Target Audience         | CS students, developers                                                                                                                            |
|                                    | Gamification Elements   | Attack–defend exercises, Immediate feedback, Drill-and-practice tasks                                                                              |

*Continued on next page*

Table 9.2 – Comparison of Our Proposed System with Existing Works

| Paper Name                                 | Attribute                      | Details                                                                                                                                                     |
|--------------------------------------------|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                            | Limitations/<br>Gaps           | Small user study, Limited participants, Prototype only, Limited vulnerability set, No long-term evaluation, No learning outcome comparison                  |
| <b>Lopes et al. (2018)</b>                 | Security Topics Covered        | Ransomware, social network information sharing, phishing, and password security                                                                             |
|                                            | Target Audience                | Teenagers ages 14-17, educational professionals, and law enforcement investigators                                                                          |
|                                            | Gamification Elements          | Detective narrative/role play, mini-games on each security topic, decision-based learning with immediate feedback                                           |
|                                            | Limitations/<br>Gaps           | Is more about general cyber crime prevention, not secure coding or development; is aimed at end-users, not developers                                       |
| <b>Arif et al. (2019)</b>                  | Security Topics Covered        | General programming/coding practices (HTML, CSS, JavaScript)                                                                                                |
|                                            | Target Audience                | Programming learners, vocational high school students                                                                                                       |
|                                            | Gamification Elements          | Real-time coding, points rewards, immediate feedback, ADDIE model + prototyping                                                                             |
|                                            | Limitations/<br>Gaps           | Not security-focused; general programming education and no vulnerability training                                                                           |
| <b>Our Proposed System: Play to Secure</b> | <b>Security Topics Covered</b> | <b>10 web vulnerabilities: SQLi, XSS, Clickjacking, Insufficient Authentication, DOM Clobbering, CSS Injection, CDN Tampering, NoSQLi, CSP Bypass, IDOR</b> |
|                                            | <b>Target Audience</b>         | <b>Novice developers in Bangladesh</b>                                                                                                                      |

*Continued on next page*

Table 9.2 – Comparison of Our Proposed System with Existing Works

| Paper Name | Attribute                    | Details                                                                                                                                                                                                                                                                                                    |
|------------|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | <b>Gamification Elements</b> | Narrative-based learning 3 levels (Theory → Attack Simulation → Prevention), Octalysis framework-based design, points/coins, badges, leaderboards, social features (knowledge wall, chat, groups), mystery gifts, integrated AI-powered feedback system. Validated through the MEEGA+ evaluation framework |
|            | <b>Limitations/ Gaps</b>     | To incorporate more advanced vulnerabilities                                                                                                                                                                                                                                                               |

# Chapter 10

## Conclusion and Future Work

This study was conducted in a systematic manner to cover the gap between the resources of security and secure coding practices among novice developers. The research employed several approaches such as examining the existing literature, empirical interviews of 32 developers, creation of a gamified platform and undertaking a serious evaluation of 30 participants.

The study contributes to cybersecurity education and gamification in a number of ways, by giving the initial organized interviews of security awareness among the various experience levels in the Bangladeshi developer society. The research reported the significant knowledge gaps, as all novice participants were unaware of the OWASP Top 10 security guidelines. It was found that there was heavy dependency on frameworks by developers who did not have the knowledge about the security concepts behind them. The study found a vulnerability propagation model at various career stages where novices create security vulnerabilities, mid-level developers identify inconsistently and senior developers bear the organizational liability of fixing the vulnerabilities.

Play to Secure, the gamified platform created during the course of the study, covers seven vulnerabilities the most applicable to novice developers. These are SQL Injection, Cross-site scripting, Insecure Direct Object Reference, and DOM Clobbering, NoSQL injection, Clickjacking, and Broken Authentication. The research has purposely omitted the attacks related to the infrastructure like CDN Tampering, CSS Injection, and CSP Bypass. This was decided upon following the results of the interviews, where these high-level subjects are not within the normal purview of the responsibilities of novice developers. The platform used the Octalysis framework in a systematic way, which offers a repeatable framework that can be applied by other people to motivational design in security education.



The evolution method was based on the methodology that included the MEEGA+ assessment with IRT-based quality classification and determined the empirical assessment standards that surpass mere subjective feedback. Each of the 13 evaluation dimensions met the success threshold with a mean of 4.34 of the 5.0. The IRT rating of the platform was Good Quality with a rating of 60.11. On the rating scale, 81.5% of the participants considered the platform as of Good Quality and 18.5% considered it of Excellent Quality, with none of them perceiving it as Low Quality. The level of engagement was also very high with a median badge completion rate of 92.3 and a mean of 87.5.

64% of Bangladeshi websites are vulnerable and this gamified platform explores the common vulnerabilities that exist in this percentage.. As opposed to single-purpose training tools, it includes social elements of learning like social walls, chat, and group features. These additions are inspired by the interview result that developers acquire security skills specifically in the areas of peer interaction and mentorship as opposed to academic study.

## 10.1 Limitations

The platform covers seven vulnerabilities that are chosen by considering novice coding practices. This also implies that it does not involve more complex subjects such as CDN Tampering, CSS Injection, and CSP Bypass, which are better suited to senior developers and DevOps teams to deal with. The examples of the platform are based on the JavaScript technology stack with React, Node.js, and MongoDB. This option is indicative of existing industry trends but implies that the usability of the platform to other technology stacks has not been proven.

There were a number of methodological limitations that influenced the study. The study was geographically oriented on the Bangladesh software industry. Although this was insightful in terms of context-specific information, it implies that the results might not be applicable to other cultures that have a different educational system and organization culture. The sample size was 32 participants of interview (20 novice, 9 mid-level, 3 senior developers) and 30 of evaluation participants. These were large enough to be statistically valid and thematically saturated, but not large enough to provide detailed subgroup analysis. Two weeks was the evaluation period and this reflected instant perceptions but did not reflect the long term memory or behavioral change in real production settings. The research lacked a control study group who were provided with conventional security training, as a result of which the particular contribution that gamification brought to the learning outcomes cannot be isolated from traditional learning.

The method of evolution was also limited. The MEEGA+ model evaluates perceived learning and engagement but not the objective security knowledge acquisition and real improvements in code quality. Accessibility dimension was found to be low with Cronbach’s alpha of 0.258. Professional networks and snowball sampling used to recruit participants can have resulted in self-selection bias because the volunteers of the study are already interested in security more than the general population.

## 10.2 Future Research Directions

The future work can be dedicated to the further reinforcement of the security awareness among the developers in two primary directions. First, the platform can be extended to accommodate advanced vulnerability scenarios, which indicate emerging real-world threats, including CDN tampering, CSS injection, and Content Security Policy (CSP) bypass, CSRF (Cross Site Request Forgery) as well as other server-side attacks. The view of such advanced attack vectors can increase the knowledge of the developers about current security risks and encourage a more security-conscious mindset in development.

Second, it is possible to conduct long-term controlled studies to assess the long-term effect of the platform on the security awareness and coding behavior of the developers. Because the present study was conducted within a short time period, a longitudinal study comparing game-based learning to the conventional training programs would be beneficial in evaluating the sustainability of security awareness and secure code writing.

## 10.3 Conclusion

Security breaches have been happening in web applications even though tools and guidelines are available. This study proves that the issue is based on the systemic failure of education. Security is taught in theory and not reinforced practically, and is considered secondary to functionality, and is not well incorporated into training programs of developers.

A vulnerability propagation model was found in the interview study of 32 developers. The level of basic security knowledge among novice developers is very low, and almost none of them know OWASP Top 10 guidelines. Mid-level developers exhibit the security behaviors that are framework-dependent without having knowledge of the backend concepts of the framework libraries. The disproportionate burden of remediation is on senior developers. This wave is evidence of malfunction of the

educational system and not negligence of developers individually.

Play to Secure attempts to fill these gaps in four key ways. First, it targets seven vulnerabilities that can be caused by beginners, grounded on the gaps of knowledge that are documented through the interviews. Second, it uses the Octalysis system to ensure long-term engagement. Third, it employs three-level progression system that proceeds to awareness to attack simulation to a prevention coding. Fourth, it has social learning capabilities which facilitate cooperation among peers.

This method is supported by the MEEGA+ assessment using 30 subjects. The platform was rated as Good Quality with 60.11 score. The success was attained in all dimensions with an average mean of 4.34 against 5.0. The best scores were Perceived Learning with 4.59 and Satisfaction with 4.54. The participation was high with a median completion rate of 92.3.

This research has a number of important implications. To start with, the issue of scope is important. The prioritization of the seven vulnerabilities that were directly related to the novice coding behavior was more successful than attempting to address everything altogether, which would have bombarded the learners with infrastructure-related issues that were beyond the scope of their capabilities. Second, motivation is necessary and not optional. The application of the Octalysis framework proves that the motivation design is required to maintain a long-term interaction with technical security material. Third, assumptions do not work as well as evidence-based design. The effectiveness of the platform lies in the basis on empirical data through interviews, documented gaps, and real-life practices, as opposed to what the developers are supposed to know. Fourth, standards of evaluation are important. The educational security tools should be empirically evaluated according to such frameworks as MEEGA+ with IRT classification and not presumed to be effective due to engagement levels. Fifth, learning is fueled by collaboration and not solitude. The social characteristics were necessary since developers get to learn security in the process of interacting with peers.

To apply in practice, the vulnerability rate of 64% in Bangladeshi websites is the representation of systemic educational choice. The study gives proven methodology of alternative approaches. Viable security education demands the input of various stakeholders. Colleges should incorporate security into their coursework, coupled with practical implementation as opposed to making it a stand-alone subject. Business organisations must invest in active and continuous training that is not merely a compliance checklist. The developers of the tools must create security-by-default-designed tools that are easier to code securely. The policymakers should come up with standards that will raise the level of security education.

The research study provides validated research methodology, empirical evidence, and implementation specifications that can be replicated and extended by other researchers and practitioners. The platform, frameworks employed, and the assessment approaches are recorded in such a way that they can be re-use in other contexts, technologies, and organizational environments.

The change from insecure to secure code practice takes time to be sustained. This study shows that gamification can be effective and interesting in providing security education in case of empirically-based design, suitable scoping, and motivation based on the framework. The question that is left is on the speed and scale of implementation. How quickly do these proven techniques get to the developers who require them, with material being well aligned to their jobs and duties?

Security education should not be taught passively in theory but practically, and the content taught should be based on what the learners are actually doing in their coding work. This change necessitates evidence-based practice such as that revealed in this research. The techniques, results, and platform that are reported here offer a basis on which others can build and acclimatize to their unique situations.

# References

- [1] M. S. N. Chowdhury, *Best practices for securing mern stack applications: A comprehensive study of authentication, authorization and data protection*, Jan. 1970. [Online]. Available: <https://www.theseus.fi/handle/10024/876758>.
- [2] P. Beynon-Davies, C. Carne, H. Mackay, and D. Tudhope, “Rapid application development (rad): An empirical review,” *European Journal of Information Systems*, vol. 8, no. 3, pp. 211–223, 1999.
- [3] R. Moreno, R. E. Mayer, H. A. Spire, and J. C. Lester, “The case for social agency in computer-based teaching: Do students learn more deeply when they interact with animated pedagogical agents?” *Cognition and instruction*, vol. 19, no. 2, pp. 177–213, 2001.
- [4] B. Rittle-Johnson and K. R. Koedinger, “Comparing instructional strategies for integrating conceptual and procedural knowledge,” 2002.
- [5] R. Hunicke, M. Leblanc, and R. Zubek, “Mda: A formal approach to game design and game research,” *AAAI Workshop - Technical Report*, vol. 1, Jan. 2004.
- [6] C. N. Quinn, *Engaging learning: Designing e-learning simulation games*. John Wiley & Sons, 2005.
- [7] A. Repenning and C. Lewis, “Playing a game: The ecology of designing, building and testing games as educational activities,” in *EdMedia+ Innovate Learning*, Association for the Advancement of Computing in Education (AACE), 2005, pp. 4901–4905.
- [8] V. Braun and V. Clarke, “Using thematic analysis in psychology,” *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.
- [9] W. G. Halfond, J. Viegas, A. Orso, et al., “A classification of sql injection attacks and countermeasures,” in *ISSSE*, 2006.
- [10] S. Triukose, Z. Al-Qudah, and M. Rabinovich, “Content delivery networks: Protection or threat?” In *Computer Security—ESORICS 2009: 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings 14*, Springer, 2009, pp. 371–389.

- [11] M. Boberski et al., “The ten most critical web application security risks,” *Technique Report, OWASP Foundation*, 2010.
- [12] L.-S. Huang, Z. Weinberg, C. Evans, and C. Jackson, “Protecting browsers from cross-origin css attacks,” in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 619–629.
- [13] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson, “Busting frame busting: A study of clickjacking vulnerabilities at popular sites,” *IEEE Oakland Web*, vol. 2, no. 6, p. 24, 2010.
- [14] A. K. Dalai and S. K. Jena, “Evaluation of web application security risks and secure design patterns,” in *Proceedings of the 2011 International Conference on Communication, Computing & Security*, 2011, pp. 565–568.
- [15] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schechter, and C. Jackson, “Click-jacking: Attacks and defenses,” in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 413–428.
- [16] A. Moniruzzaman and D. S. A. Hossain, “Comparative study on agile software development methodologies,” *arXiv preprint arXiv:1307.3356*, 2013.
- [17] J. R. Anderson, *Rules of the mind*. Psychology Press, 2014.
- [18] K. Elshazly, Y. Fouad, M. Saleh, and A. Sewisy, “A survey of sql injection attack detection and prevention,” *Journal of Computer and Communications*, vol. 2014, 2014.
- [19] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G. N. Peri, “Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation,” in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 66–77.
- [20] N. Kaur and P. Kaur, “Mitigation of sql injection attacks using threat modeling,” *SIGSOFT Softw. Eng. Notes*, vol. 39, no. 6, pp. 1–6, Dec. 2014, ISSN: 0163-5948. DOI: 10.1145/2674632.2674638. [Online]. Available: <https://doi.org/10.1145/2674632.2674638>.
- [21] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, “When https meets cdn: A case of authentication in delegated service,” in *2014 IEEE Symposium on Security and Privacy*, IEEE, 2014, pp. 67–82.
- [22] D. Gillman, Y. Lin, B. Maggs, and R. K. Sitaraman, “Protecting websites from attack with secure delivery networks,” *Computer*, vol. 48, no. 4, pp. 26–34, 2015.

- [23] S. Gupta and B. B. Gupta, "Cross-site scripting (xss) attacks and defense mechanisms: Classification and state-of-the-art," *International Journal of Systems Assurance Engineering and Management*, vol. 8, Sep. 2015. DOI: 10.1007/s13198-015-0376-0.
- [24] K. Robson, K. Plangger, J. Kietzmann, I. McCarthy, and L. Pitt, "Is it all a game? understanding the principles of gamification," *Business Horizons*, Apr. 2015. DOI: 10.1016/j.bushor.2015.03.006.
- [25] J. Chen et al., "Forwarding-loop attacks in content delivery networks.," in *NDSS*, 2016.
- [26] I. Fianyi, "Curbing cyber-crime and enhancing e-commerce security with digital forensics," *CoRR*, vol. abs/1610.08369, 2016. arXiv: 1610.08369. [Online]. Available: <http://arxiv.org/abs/1610.08369>.
- [27] B. Hou, K. Qian, L. Li, Y. Shi, L. Tao, and J. Liu, "Mongodb nosql injection analysis and detection," in *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2016, pp. 75–78. DOI: 10.1109/CSCloud.2016.57.
- [28] A. Ron, A. Shulman-Peleg, and A. Puzanov, "Analysis and mitigation of nosql injections," *IEEE Security & Privacy*, vol. 14, no. 2, pp. 30–39, 2016. DOI: 10.1109/MSP.2016.36.
- [29] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc, "Csp is dead, long live csp! on the insecurity of whitelists and the future of content security policy," Oct. 2016, pp. 1376–1387. DOI: 10.1145/2976749.2978363.
- [30] H. Maldonado et al., "We learn better together: Enhancing elearning with emotional characters," in *Computer supported collaborative learning 2005*, Routledge, 2017, pp. 408–417.
- [31] M. Naderifar, H. Goli, F. Ghaljaie, et al., "Snowball sampling: A purposeful method of sampling in qualitative research," *Strides in development of medical education*, vol. 14, no. 3, pp. 1–6, 2017.
- [32] "Analysis of gamification models in education using mda framework," *Procedia Computer Science*, vol. 135, pp. 385–392, 2018, The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life, ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2018.08.187>.
- [33] F. A. Bachtiar, F. Pradana, B. Priyambadha, and D. I. Bastari, "Coma: Development of gamification-based e-learning," in *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2018, pp. 1–6. DOI: 10.1109/ICITEED.2018.8534875.

- [34] S. Kundu, K. A. Islam, T. T. Jui, S. Rafi, M. A. Hossain, and I. H. Chowdhury, "Cyber crime trend in bangladesh, an analysis and ways out to combat the threat," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*, IEEE, 2018, pp. 474–480.
- [35] J. Mullins and R. Sabherwal, "Beyond enjoyment: A cognitive-emotional perspective of gamification," Jan. 2018. DOI: 10.24251/HICSS.2018.152.
- [36] G. Petri, C. Gresse von Wangenheim, and A. Borgatto, *Meega+: A method for the evaluation of educational games for computing education*, Jul. 2018.
- [37] R. F. Arif, H. A. Rosyid, and U. Pujianto, "Design and implementation of interactive coding with gamification for web programming subject for vocational high school students," in *2019 International Conference on Electrical, Electronics and Information Engineering (ICEEIE)*, vol. 6, 2019, pp. 177–182. DOI: 10.1109/ICEEIE47180.2019.8981454.
- [38] W. Bai, O. Akgul, and M. L. Mazurek, "A qualitative investigation of insecure code propagation from online forums," in *2019 IEEE Cybersecurity Development (SecDev)*, IEEE, 2019, pp. 34–48.
- [39] Y.-k. Chou, *Actionable gamification: Beyond points, badges, and leaderboards*. Packt Publishing Ltd, 2019.
- [40] C. Gupta, R. Singh, and A. K. Mohapatra, "Automated code correction to mitigate sql injection vulnerabilities," Nov. 2019.
- [41] S. Ivanova and G. Georgiev, "Towards a justified choice of gamification framework when building an educational application," in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019, pp. 594–599.
- [42] M. Liu, B. Zhang, W. Chen, and X. Zhang, "A survey of exploitation and detection methods of xss vulnerabilities," *IEEE Access*, vol. 7, pp. 182004–182016, 2019. DOI: 10.1109/ACCESS.2019.2960449.
- [43] M. Moniruzzaman, F. Chowdhury, and M. S. Ferdous, "Measuring vulnerabilities of bangladeshi websites," in *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, IEEE, 2019, pp. 1–7.
- [44] M. Tahaei and K. Vaniea, "A survey on developer-centred security," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS & PW)*, 2019, pp. 129–138. DOI: 10.1109/EuroSPW.2019.00021.
- [45] C. Blundo, S. Cimato, and L. Siniscalchi, "Managing constraints in role based access control," *IEEE Access*, vol. 8, pp. 140 497–140 511, 2020. DOI: 10.1109/ACCESS.2020.3011310.



- [46] Dubizzle Labs. “Express-mongo-sanitize: Express middleware to sanitize mongodb queries.” [Online]. Available: <https://www.npmjs.com/package/express-mongo-sanitize>.
- [47] T. E. Gasiba, U. Lechner, M. Pinto-Albuquerque, and D. Mendez, “Is secure coding education in the industry needed? an investigation through a large scale survey,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, IEEE, 2021, pp. 241–252.
- [48] S. Park and S. Kim, “Leaderboard design principles to enhance learning and motivation in a gamified educational environment: Development study,” *JMIR serious games*, vol. 9, no. 2, e14746, 2021.
- [49] M. Penelova, “Access control models,” *Cybernetics and Information Technologies*, vol. 21, no. 4, pp. 77–104, 2021.
- [50] O. Priyadi, I. Ramadhan, D. Sensuse, R. Suryono, and K. Kautsarina, “Gamification in software development: Systematic literature review,” in Sep. 2022, pp. 386–398, ISBN: 978-3-031-15190-3. DOI: 10.1007/978-3-031-15191-0\_37.
- [51] O. Priyadi, I. Ramadhan, D. I. Sensuse, R. R. Suryono, and Kautsarina, “Gamification in software development: Systematic literature review,” in *International Conference on Networking, Intelligent Systems and Security*, Springer, 2022, pp. 386–398.
- [52] L. K. Shar, C. M. Poskitt, K. J. Shim, and L. Y. L. Wong, “Xss for the masses: Integrating security in a web programming course using a security scanner,” in *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, 2022, pp. 463–469.
- [53] O. Alghamdi, S. Clinch, M. Alhamadi, and C. Jay, “Novice programmers strategies for online resource use and their impact on source code,” in *2023 IEEE/ACM 16th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2023, pp. 92–104. DOI: 10.1109/CHASE58964.2023.00018.
- [54] S. Khodayari and G. Pellegrino, “It’s (dom) clobbering time: Attack techniques, prevalence, and defenses,” in *2023 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2023, pp. 1041–1058.
- [55] K. Malik. “Mitigating DOM clobbering attacks in JavaScript.” [Online]. Available: <https://snyk.io/blog/mitigating-dom-clobbering-attacks-javascript/>.
- [56] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, “Do users write more insecure code with ai assistants?” In *Proceedings of the 2023 ACM SIGSAC conference on computer and communications security*, 2023, pp. 2785–2799.

- [57] M. M. Rahman, M. Z. Rahman, M. A. Kashem, and M. M. Hasan, “An analysis of xss vulnerabilities and prevention of xss attacks in web applications,” *ResearchGate*, 2023, Accessed: 2025-06-16. [Online]. Available: [https://www.researchgate.net/publication/371724261\\_An\\_Analysis\\_of\\_XSS\\_Vulnerabilities\\_and\\_Prevention\\_of\\_XSS\\_Attacks\\_in\\_Web\\_Applications](https://www.researchgate.net/publication/371724261_An_Analysis_of_XSS_Vulnerabilities_and_Prevention_of_XSS_Attacks_in_Web_Applications).
- [58] S. Singh and M. Dandotiya, “An efficient approach for mitigating insecure direct object reference (idor) bug bounty method,” *Int J Res Appl Sci Eng Technol*, vol. 11, no. 6, 2023.
- [59] O. Abuali, *Mongodb and nosql injection and prevention*, Feb. 2024. DOI: 10.13140/RG.2.2.29434.67524.
- [60] E. Amissah and F. Bentil, “Exposing insecure direct object reference (idor) vulnerabilities in academic publication platforms: A case study),” *International Journal of Engineering Research and*, vol. 13, Sep. 2024.
- [61] C. Chaki, a. Biswas, and G. M. I. Kabir, “Inspecting the legal aspects of the cyber crimes committed against financial institutions with special reference to the bangladesh bank cyber intrusion,” vol. 03, pp. 54–63, Jun. 2024.
- [62] H. Cigdem, M. Ozturk, Y. Karabacak, N. Atik, S. Gürkan, and M. H. Aldemir, “Unlocking student engagement and achievement: The impact of leaderboard gamification in online formative assessment for engineering education,” *Education and Information Technologies*, vol. 29, no. 18, pp. 24 835–24 860, 2024.
- [63] V. Nardo, R. Fino, M. Fiore, G. Mignogna, M. Mongiello, and G. Simone, “Usage of gamification techniques in software engineering education and training: A systematic review,” *Computers*, vol. 13, p. 196, Aug. 2024. DOI: 10.3390/computers13080196.
- [64] A. Paul, V. Sharma, and O. Olukoya, “Sql injection attack: Detection, prioritization & prevention,” *Journal of Information Security and Applications*, vol. 85, p. 103 871, 2024.
- [65] O. Pylyp, “Analysis of vulnerabilities in the content security policy standard to enhance website security,” *International Journal of Engineering and Computer Science*, vol. 13, pp. 26 677–26 681, Dec. 2024. DOI: 10.18535/ijecs/v13i12.4948.
- [66] A. Siderova, M. Daneva, F. A. Bukhsh, and J. J. Arachchige, “Security approaches in model-driven engineering for web applications: The state-of-the-art in the last 10 years,” in *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*, 2024, pp. 155–163.
- [67] N.-E.-S. Ahmed, “Vulnerability and accessibility analysis of bangladesh ministry of land’s government websites,” *OALib*, vol. 12, pp. 1–21, Jan. 2025. DOI: 10.4236/oalib.1112756.

- [68] N. K. Kakolaki, “A comparative analysis of identifier schemes: Uuidv4, uuidv7, and ulid for distributed systems,” *arXiv preprint arXiv:2509.08969*, 2025.
- [69] E. Kamau, T. Myllynen, A. Collins, G. Babatunde, A. Alabi, and A. Pub, “Advances in full-stack development frameworks: A comprehensive review of security and compliance models,” *International Journal of Multidisciplinary Research and Growth Evaluation*, vol. 04, : 639–652, Jan. 2025.
- [70] OWASP Cheat Sheet Series, *Insecure direct object reference prevention cheat sheet*, OWASP Foundation, 2025. [Online]. Available: [https://cheatsheetseries.owasp.org/cheatsheets/Insecure\\_Direct\\_Object\\_Reference\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html).
- [71] Z. Liu, T. Lee, J. Yu, Z. Kang, and Y. Cao, “The domino effect: Detecting and exploiting dom clobbering gadgets via concolic execution with symbolic dom,”

# Appendix A

## Gamified System Request

### A.1 System Request

The system aims to provide a gamified platform based on the famous "OCTALYSIS Framework" by Yu-kai Chou to help novice developers learn secure coding practices. It will offer interactive challenges, real-time feedback, and a social platform for users to collaborate and improve their coding skills.

#### A.1.1 Business need

There is a growing need to educate novice developers on secure coding practices, especially in light of common vulnerabilities like XSS and SQLi. This system will engage users through challenges, tracking progress, and providing detailed feedback, addressing a crucial gap in developer education.

#### A.1.2 Business requirements

- User management and profile customization
- Learning progress tracking through challenges
- Real-time feedback and interactive problem-solving
- A leaderboard and social platform for collaboration
- Secure coding challenges based on OWASP Top 10 vulnerabilities
- A peer review system to encourage feedback and collaboration

### **A.1.3 Business value**

The system will provide a valuable resource for developers, allowing them to learn and improve their skills in a fun, interactive way. It will help organizations ensure that developers adopt secure coding practices, reducing security risks and increasing overall software quality.

### **A.1.4 Special issues or constraints**

- The platform must be secure and protect user data, especially for storing user progress and challenges.
- The system needs to be scalable to handle a growing number of users and challenges.
- Integration with existing development tools or environments may require additional work.

## **A.2 Functional Requirements**

### **A.2.1 Module 1: User Management and Social Platform**

1. The system will allow the user to register/log in using email and secure password encryption.
2. The user can update profile information, like institution and experience level, in their profile segment.
3. A personalized profile will track badges earned and points.
4. A progress path-based simulation will visualize learning milestones and weekly activity (Game Completion).
5. A Knowledge wall is created where users can post about their queries and share their knowledge.
6. The user can comment and react on other members' comments and help each other.
7. A user can use a personal chatbox to communicate with other members who have joined the platform.

8. There will be an admin login system, and the admin can add/delete/edit questions for the quiz and operate the whole software easily.
9. A global dashboard will show the active members using it at a time and a quick recap on the attacks that we'll be focusing on.

### **A.2.2 Module 2: Learning Challenges and Feedback**

1. The user will receive challenges based on OWASP Top 10 vulnerabilities (e.g., XSS, SQLi).
2. Scenario-based simulations (e.g., attack-defense tasks in developing contexts) and attack questions in Level-1.
3. After each task, users will receive an explanation of the topic, and from there, they will know what error they made. Interactive problem-solving with AI-integrated features for further knowledge.
4. Level-based feature unlock (after completing certain levels, the user will get access to new chat features).

### **A.2.3 Module 3: Social Features and Leaderboard**

1. Users can search and create group chats to share learning with each other.
2. A global leaderboard ranking the users based on XP and badge count.
3. In the group chat (topic-based chat channels), a user can create the group and can add members. Only the creator can remove members and change the group name. Any member can leave the group.

### **A.2.4 Module 4: Advanced Learning and Assessment**

1. Designing the Level-2 with multiple scenarios and segmented simulations for each attack vector. Each attack will include different types of situations, challenges, and segments like detection, mitigation, and post-attack analysis to help users understand the full attack cycle through interactive practice.
2. In the Level-3, the users will be asked to build a small project based on the attack vectors they have learned. This hands-on experience will allow them to present their knowledge and understanding of the vulnerabilities and secure coding practices.

3. An AI agent will be built to evaluate the user's submitted projects where the potential vulnerabilities and coding flaws will be detected. It will provide detailed feedback, which will help the users later on.
4. There will be a mystery gift system where users will receive surprise rewards after completing certain levels or milestones.
5. Players will lose points if they make any mistakes.

## A.3 Technology

### A.3.1 Framework and Languages

- **Language:** JavaScript
- **Framework:** React.js (Frontend), Node.js with Express.js (Backend)
- **Styling:** Tailwind CSS
- **Database:** MongoDB
- **Deployment:** Cloudflare (Frontend), Render (Backend), MongoDB Atlas (Database) , N8N (Render)