SEMARAK ILMU PUBLISHING

Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:
https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/index
ISSN: 2462-1943


# OWASP A03 Injection Vulnerability in Web Application Development

Phei-Chin Lim[1,*], Ging-Wei Andy Chieng[2], Huo-Chong Ling[3], Nurfauza Jali[1]

[1] Faculty of Computer Science and Information Technology, Universiti Malaysia Sarawak, 94300 Kota Samarahan, Sarawak, Malaysia
[2] Asia Pacific Cloud Continuous Operation & Delivery Department, Huawei Technologies Co. Ltd., 50400 Kuala Lumpur, Malaysia
[3] School of Science, Engineering & Technology, RMIT University Vietnam, 700000 Ho Chi Minh City, Vietnam

**ABSTRACT**

Web applications are crucial for businesses and individuals by providing efficient communication, collaboration, and access to services and information via browsers, boosting connectedness, productivity, and creativity in the digital era. Insecure web applications pose risks of data breaches, malware, and unauthorized access which jeopardize user privacy, trust, and organizational security. Web developers must be knowledgeable and prepared to deal with common vulnerabilities in web applications. A prototype web application (https://webriska3.tech) with lesson and editor module is developed to train web developers on the Open Web Application Security Project (OWASP) Top Ten security risks, focusing on A03 - Injection vulnerability. OWASP A03 Injection vulnerability is one of the most common vulnerabilities that is at the heart of any database-driven web applications. Evaluation on the prototype in improvement knowledge on A03 – Injection vulnerability, testers are recruited to complete two coding tasks in laboratory environment. 80% of testers mastered Output escaping/encoding defensive technique while Prepared statement/Parameterized Query defensive technique is the hardest to master. The prototype obtained average System Usability Scale (SUS) score of 57 that is below average, indicating issues with the prototype interface. This work showed promising results of increase understanding on A03 Injection vulnerability and implementation skills to protect web application against attack and exploitations.

***Keywords:***
OWASP Top 10; Web application vulnerability; Web security; SQL injection

## 1. Introduction

The World Wide Web (WWW) has evolved significantly over the years since the conceptualization by Tim Berners-Lee [1]. The concept of Web 1.0 was initially designed in the 1990s as 'static web' to display information to users, and later evolved to 'dynamic web' or Web 2.0 in the 2000s which allows users to finally participate, connect and interact. Blogs, wiki and forums that allowed collaborative content and knowledge sharing became popular. Web 2.0 is also the era of social media where applications such as Facebook, Twitter and YouTube gained millions of active users. Web 3.0 or 'semantic web' [2] is the current phase that focuses on decentralization, openness, and personalized

* *Corresponding author.*
*E-mail address: pclim@unimas.my*

https://doi.org/10.37934/araset.57.1.107116

user experience through 'read, write, execute Web'. With the leveraging of technologies such as artificial intelligence, machine learning, and Internet of Things (IoT), risks such as privacy and cybersecurity become a significant challenge as Web 3.0 relies on extensive data collection and analysis.

In the era with exponential growth of information technologies, more and more of small, medium to enterprise adopted digital transformation especially due to Covid-19 pandemic lockdown. There are increasing illegal cyberattacks targeting web applications, such as stealing personal data like Personal Identifiable Information (PII) and Personal Health Information (PHI) [3] for ransom, or selling it to other parties for financial gain [4]. To minimize the vulnerability of web applications, cybersecurity awareness education [5,6] and detection framework [7] are both crucial. An open community consisting of corporations, foundations, developers, and volunteers have initiated and supported the Open Web Application Security Project or OWASP. The OWASP Top 10 version 2021 [8] contains the top 10 critical security risks with web applications:

A01: Broken Access Control
A02: Cryptographic Failures
A03: Injection
A04: Insecure Design
A05: Security Misconfiguration
A06: Vulnerable and Outdated Components
A07: Identification and Authentication Failures
A08: Software and Data Integrity Failures
A09: Security Logging and Monitoring Failures
A10: Server-Side Request Forgery

Loose web security may be abused by hackers through injection attack. The injection attack is the third security risk in the OWASP Top Ten 2021. The paper focus on the education prototype to highlight and implement the important issues in A03 Injection during code writing. The lessons and editor module in the prototype can be used to train web developers in writing secure code [9]. The structure of this paper is as follows: defending techniques and insecure web applications are reviewed in Section 2, methodology is discussed in Section 3, while Section 4 reports testing results and discussion. Conclusion of this work is given in Section 5.

## 2. Related Work

According to Marashdeh *et al.,* [10], SQL injection refers to the type of attack towards the web application's database through injecting malicious code to gain access to confidential information without authentication. The most common injection attack in A03 injection vulnerability is SQL Injection [11,12] and Cross-Site Scripting XSS injection [13,14]. Through the injection attack like SQL Injection, the attacker can have full access to the web application's database. Therefore, it is critical for web developers or security professionals to implement OWASP Top 10 as a security guideline [15,16] to effectively patch web application vulnerabilities and minimize the security risk.

### *2.1 Defencing Techniques*

In 2021, Rai *et al.,* [17] reviewed and summarized ten counter measures on the SQL Injection, which are simpler than other OWASP Top 10 vulnerabilities as most of the defencing techniques

involves comparatively simpler, but effective handling of user provided data. These ten defencing techniques are used as a metric to evaluate existing web applications compliances with OWASP Top 10 web application security risks.

The first technique is *whitelisting and blacklisting (WB)* which are detection methods to detect illegal SQL database query in the web application [18]. To use blacklisting detection method, the user must define the illegal SQL queries pattern such as AND, OR and others in their web application. When the web application receives the illegal SQL query from the injection attack, it will ignore the blacklisted SQL queries pattern. However, using blacklisting detection is not an efficient way in preventing SQL injection attack. It will be ineffective if the web developer does not define the entire backlist list for the illegal SQL query due to insufficient knowledge in this field.

i. Prepared Statement/Parameterized Query (PSPQ): One of the methods used to prevent SQL injection attack. This technique refers to when the web application developer uses a more secure way in predefining the SQL queries where the queries have the placeholder. Parameterized queries force the web developer to define all SQL statements first and the parameter of the user input will be passed into the prepared query later. Prepared statement also has high efficiency when the web application needs to execute same or similar SQL query frequently [19].
ii. Stored Procedure (SP): A group of predefined SQL statements that are stored in database for repetitive tasks such as data insert, update, delete or query. To prevent an attack, user input is passed as parameters to Stored Procedure where user input will be validated and sanitized by the database [20].
iii. Defensive Coding Practice (DCP): emphasizes the coding skills of the web application developer in developing secure code, for example, forbidding of the uses of meta-characters or the identification on the user input field.
iv. Taint Based Approach / Taint Analysis (TBA): A method to track the data flow of sensitive data inside the web application [21]. Taint analysis is used to detect malicious data flow, code or malicious behaviour happening in the web application. By using Taint analysis, web developers can know how the data is being processed inside the web application. The first step in Taint based approach is to identify the sources of untrusted input in the program, such as user-supplied data passed through form fields or query parameters. Once the untrusted inputs are identified, they are tainted with a special marker to indicate that they are unsafe to use without proper validation. The taint is then propagated through the program as the inputs are passed to different variables and data structures.
v. Proxy Filter (PF): A method used to prevent SQL injection by setting up a proxy barrier between the client-side and MySQL database. An example of the proxy filters is MySQL proxy. MySQL proxy acts as a middleware which connects and manages all communication between client connection request and SQL database at the backend. It can monitor, analyse, or modify the communication between client side and SQL database. It contains a lot of other functions such as load balancing, fault analysis, query analysis, query filter and so on [22]. Hadabi *et al.,* [23] proposed a model of using proxy filter as a middle layer between client and database. It acts as a filter barrier on the user's input. When a client makes a request to proxy server and the client is a new user, registration is required where the hash value of username and password are checked for authentication before the process continues.

Modern programming languages provide *built-in functions (BIF)* to sanitize the user input before it is inserted to the database. PHP offers functions such as mysqli_real_escape_string() or mysqli::real_escape_string() to escape special characters in user input [24]. The mysqli_real_escape_string() function will escape special characters such as \n, \r, \, ', " which are used in the creation of illegal SQL query that leads to SQL injection. Another useful built-in function offered in PHP is filter_vars() which is used to validate and sanitize data such as email id, IP address etc. The filter_var() function can validate and sanitize various variables by using specific filter parameters such as FILTER_VALIDATE_DOMAIN, FILTER_VALIDATE_EMAIL, FILTER_VALIDATE_IP and many more [24].

*Instruction Set Randomization (ISR)* prevents SQL injection by randomizing the SQLs keyword such as SELECT, FROM or WHERE by appending a random integer that forms a completely different instruction sets [18]. The randomized SQL statements are processed and checked by proxy for malicious code. Implementing ISR can be very complicated, as it requires setting up proxy filter against web application and SQL database [25].

All database management system has a user account management mechanism to control who and what to access, which is also known as privileges. According to Wu *et al.,* [26], web developers should always enforce the principle of *low privileges (LP)* where the user's permissions given are necessary and sufficient to complete the task. This helps to minimize the chance of database being attacked and prevents data exploitation. An attacker may not have privileges restricted by database that can perform dangerous and irreversible attacks such as altering or dropping a table.

Lastly, *output escaping/encoding (OE)* is the process of converting some predefined characters or untrusted values to HTML entities or JavaScript code so that it is interpreted as content only [14]. PHP htmlspecialchars() function converts some predefined characters to HTML entities [24]. For example, sensitive characters in SQL such as single quote (') will be encoded into ' which provides totally different meanings when it is processed by the SQL database. This technique can prevent cross-site scripting attack by escaping malicious command.

### *2.2 Insecure Web Applications*

This section reviewed four insecure web applications created with OWASP standard for training and to raise awareness among interested web developers in testing web application vulnerabilities.

OWASP WebGoat is an open-source vulnerable Java-based web application led by Mayhew *et al.*, [27]. WebGoat is extremely vulnerable to attacks, which makes WebGoat a great platform for the interested web application developers to learn about web application security and penetration testing technique. The main goal of WebGoat is to create an interactive teaching platform on the web application vulnerability by providing three learning steps. The first step is to explain the fundamental concepts in security vulnerabilities, followed by hands-on assignments to learn how it works, and lastly possible mitigations for actual web applications. OWASP WebGoat adapted PSPQ, SP, DCP, BIF, LP and OE defencing techniques.

Damn Vulnerable Web Application or DVWA [28] is a PHP and MySQL web application that is designed for web developers to test their hacking skills in a legal way as well as let them know on the flows to securing the web application that they developed. It was first developed in 2008 and it further received features updates until 2015. DVWA aims to provide practice to some of the most common web vulnerabilities. This means that DVWA only complies with the OWASP Top 10 version before year 2015 and only adapted PSPQ, SP, DCP and LP defensive techniques.

OWASP Juice Shop [29] is a web application mimicking an online shopping platform for purchasing juice products. It is written in Node.Js, Express and Angular. It is created by Björn Kimminich and is

being developed, maintained, and translated by a team of volunteers starting from year 2014 until today. Unlike WebGoat and DVWA, Juice Shop does not provide any lesson to the user on vulnerability, which means that the user that wishes to exploit this website must have certain level of understanding and basic in attacking on the vulnerability. OWASP Juice Shop is more like a hacking challenging website that contains scoreboard that records the vulnerability exploited by the user. Users will have to explore the website, use their knowledge in attacking vulnerabilities to find the underlying vulnerabilities to score the challenges. Juice Shop adapted PSPQ, SP and DCP defencing techniques.

Hacksplaining.com [30] is a web application that encourages learning on how to protect web application through attacking it. As an alternative to educate web vulnerabilities, hacksplaining.com provides visualized lesson which involves interactive between the hacksplaining.com and users in the learning process. The defencing techniques adapted are PSPQ, SP, DCP, BIF, LP and OE.

## 3. Methodology

A web application was developed where all 10 defencing techniques summarized by Rai *et al.,* [17] are implemented to comply with OWASP A03 Injection. The mapping of weaknesses from CWE [31] in OWASP A03 Injection to the 10 defencing techniques (refer Table 1) is the outcome of requirement analysis.

UML modelling is adapted in the design phase where use case, class and sequence diagrams are created to guarantee that the prototype outcome will match the requirements. For the implementation phase, installation, and software configuration for Visual Studio Code, Laravel, Node.js and MySQL is completed. The developed prototype (https://webriska3.tech) aims to teach and guide web developers to create secure code through the lesson and editor module. Ten lessons that correspond to the ten defencing techniques in Table 1 are created in the lesson module.

**Table 1**
CWE weakness mapped to the 10 defensive techniques

| Defensive Technique | CWE Weaknesses ID in OWASP Top Ten 2021 Category A03 - Injection |
|---|---|
| Whitelisting and blacklisting (WB) | 20, 74, 75, 77, 78, 79, 88, 89, 90, 91, 94, 95, 96, 98, 99, 113, 138, 184, 470, 564, 643 |
| Prepared Statement/Parameterized Queries (PSPQ) | 89, 564, 652 |
| Stored Procedure (SP) | 89, 564 |
| Defensive Coding Practice (DCP) | 20, 74, 75, 89, 95, 917 |
| Taint Based Approach (TBA) | 89<br>89<br>89<br>89<br>89 |
| Proxy Filters (PF) | |
| Using built in function (BIF) | |
| Instruction Set Randomization (ISR) | |
| Low Privileges (LP) | 89, 564 |
| Output Escaping (OE) | 78, 79, 80, 83, 87, 96, 98, 113, 116, 138, 644 |

Example of the lesson page is given in Figure 1.

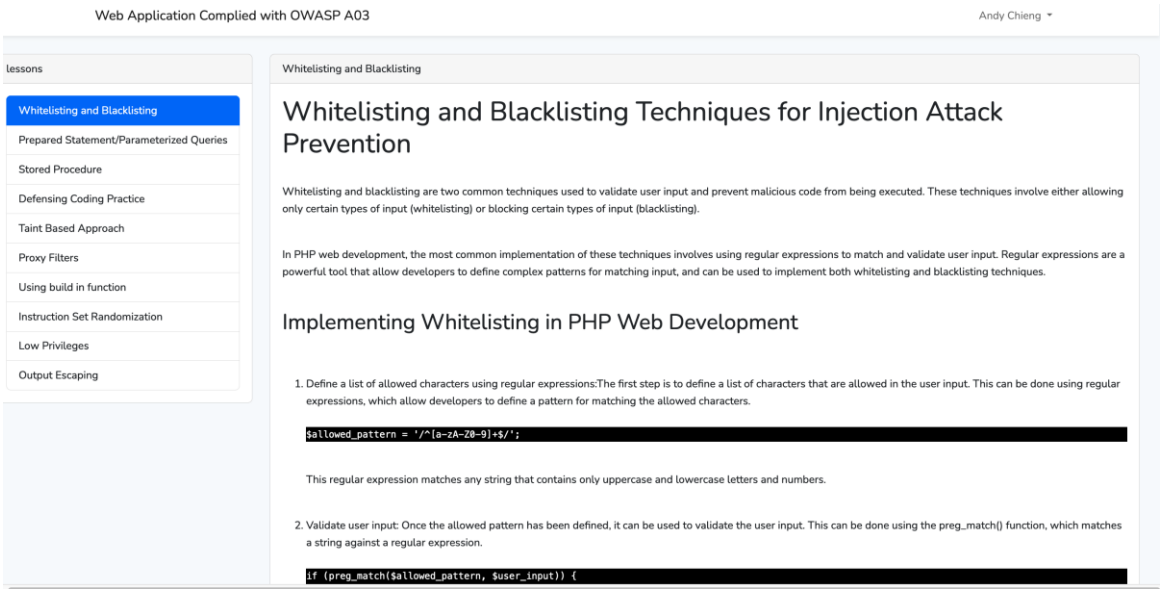


**Fig. 1.** Lesson module

For web developers to practice the defencing techniques, editor module is created as shown in Figure 2.

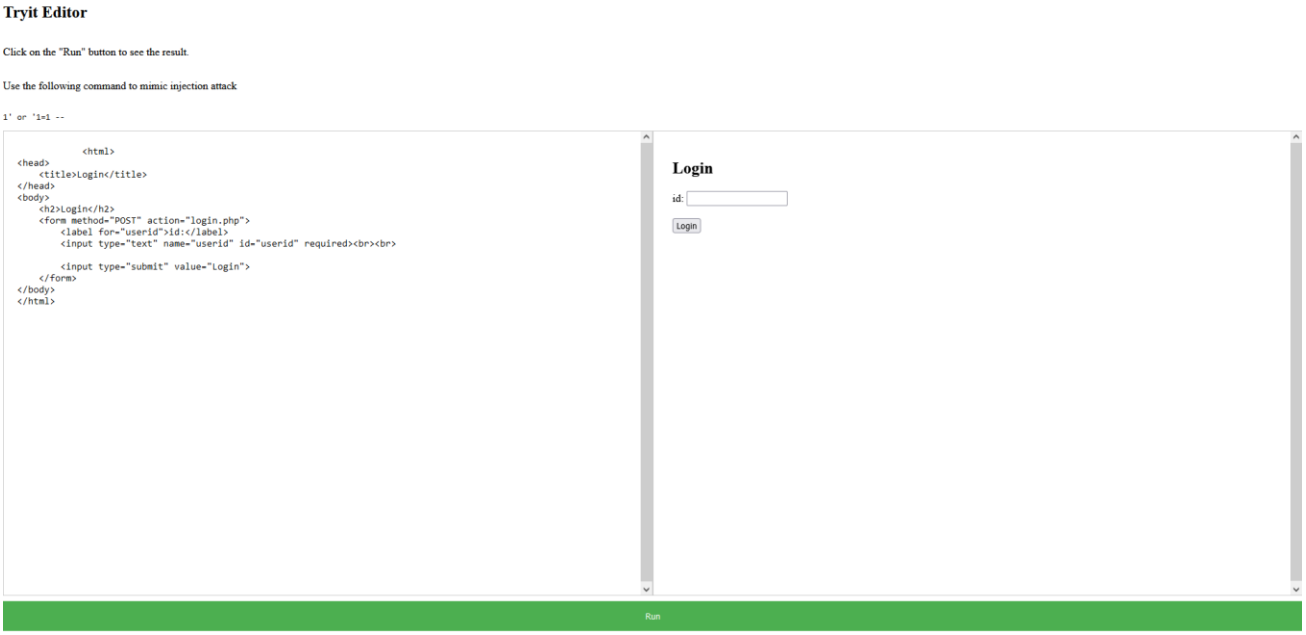


**Fig. 2.** Editor module

## 4. Results and Discussion

In the testing phase, laboratory testing and usability testing using system usability scale (SUS) [32] are conducted. Laboratory testing is conducted to test the effectiveness of the prototype developed in improving knowledge on OWASP A03 injection vulnerability, while user evaluate the usability of the prototype using SUS. 15 student testers are recruited from Software Engineering program (year 1 to year 4), Faculty of Computer Science and Information Technology, Universiti Malaysia Sarawak. 66% of the testers have no prior knowledge of OWASP A03 Injection.

In laboratory testing, testers interacted with the prototype under the authors' observation to complete two scenarios. In Scenario 1, testers are being instructed to create a simple login system. They are required to write a PHP script to handle user registration, validate user input from an HTML form, and store the user information in a database. Additionally, they need to create an HTML login form with fields for username and password for the login functionality. Lastly, they should write a PHP script to validate the login credentials entered by the user. Once the testers have completed their code, it will be evaluated using test criteria in Table 2 to assess the quality and adherence to the given requirements. In Scenario 2, the testers are instructed to take the lessons in the prototype which covers secure coding practices, security vulnerabilities, and mitigation techniques related to injection vulnerability before repeating the same tasks from Scenario 1 by applying the knowledge gained from the lessons to write a more secure code. The new set of code is evaluated using the same test criteria to determine any improvements or changes compared to code written in Scenario 1.

**Table 2**
Test criteria to evaluate testers' written code

| Defensive technique | Criteria |
|---|---|
| Whitelisting and Blacklisting (WB) | Code incorporates either whitelisting or blacklisting techniques to validate and filter user input. |
| Prepared Statement/Parameterized Queries (PSPQ) | Code utilizes prepared statements or parameterized queries when interacting with the database. |
| Stored Procedure (SP) | Code includes the use of stored procedures as a defensive technique. |
| Defensive Coding Practices (DCP) | Code demonstrates the implementation of defensive coding practices. |
| Using Built-In Function (BIF) | Code utilizes built-in functions or libraries that offer security features or protections. |
| Output Escaping (OE) | Code implements output escaping techniques. |

The evaluation results on each tester in both Scenario 1 and Scenario 2 are reported in Table 3. This comparison helps measure the effectiveness of the lessons in enhancing the testers' abilities to create secure login systems and identify potential security vulnerabilities. OE technique showed the highest improvement of 80%, followed by 73% of testers showed improvement in SP and BIF technique. Both WB and DCP techniques achieved 67% improvement while PSPQ technique achieved lowest improvement of 47%.

For subjective usability testing, the same testers are required to fill a 10 questions SUS questionnaire immediately after the laboratory testing to evaluate the perceived ease-of-use of the prototype. Firstly, the internal consistency of this SUS questionnaire with Cronbach's Alpha 0.85 indicated good reliability of the questions even with small sample size [33]. Approximately 87% of testers score SUS that falls in the 'ok/marginally acceptable' range while 2 testers (~13%) with individual SUS score in the 'poor/unacceptable' range. The average SUS score for the prototype is 57, which is below average and indicated that there are issues with the prototype interface. One possibility may be the scenario design which requires testers to perform coding tasks using other interfaces than the prototype. However, testers are required to average their worst and best experience in the SUS rating on the prototype interface. Another possibility is biasness of tester subjective assessment, where testers are unable to complete the coding task but still rate the usability questions highly. Referring to Table 3, T5 only showed ability to implement 1 out of 6 defensive techniques yet the individual SUS score is nearly the same as T1, who showed ability to implement all 6 defensive techniques. Even though T7 managed to show ability to implement 4 defensive techniques, however, SUS score for T7 is similarly as poor as T8.

**Table 3**
Evaluation Results of Testers' written code

| Tester | WB | | PSPQ | | SP | | DCP | | BIF | | OE | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Scenario 1 | Scenario 2 | Scenario 1 | Scenario 2 | Scenario 1 | Scenario 2 | Scenario 1 | Scenario 2 | Scenario 1 | Scenario 2 | Scenario 1 | Scenario 2 |
| T1 | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |
| T2 | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| T3 | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| T4 | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |
| T5 | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| T5 | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |
| T6 | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ |
| T7 | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ |
| T8 | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| T9 | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |
| T10 | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| T11 | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| T12 | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |
| T13 | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| T14 | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| T15 | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |

## 5. Conclusions

This paper presents a prototype web application designed and developed for educational purposes to train web developers to write secure code that complied with OWASP A03 injection vulnerability. The prototype consists of 10 lessons corresponding to the 10 defensive techniques. The lessons aim to deliver knowledge on how each vulnerability happens with examples. The editor module at the end of the lessons let web developer practice how the defensive technique work and how to defend against each vulnerability. The effectiveness of the prototype in improving knowledge on OWASP A03 Injection vulnerability is reflected by the result gained from laboratory testing where five defensive techniques (WB, SP, DCP, BIF and OE) obtained 67% and above improvements. However, the below average SUS score of 57 indicated that significant interface redesign and updates are needed, which shall be the focus of further investigation.

## Acknowledgement
This research was not funded by any grant. Authors wish to thank the Faculty of Computer Science and Information Technology, Universiti Malaysia Sarawak for the support.

## References
[1] Choudhury, Nupur. "World wide web and its journey from web 1.0 to web 4.0." *International Journal of Computer Science and Information Technologies* 5, no. 6 (2014): 8096-8100.
[2] Gan, Wensheng, Zhenqiang Ye, Shicheng Wan, and Philip S. Yu. "Web 3.0: The future of internet." In *Companion Proceedings of the ACM Web Conference 2023*, pp. 1266-1275. 2023. https://doi.org/10.1145/3543873.3587583
[3] Singh, Himanshi, and Mohit Dua. "Website attacks: Challenges and preventive methodologies." In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 381-387. IEEE, 2018. https://doi.org/10.1109/ICIRCA.2018.8597259
[4] Zakaria, Wira Zanoramy A., Nur Mohammad Kamil Mohammad Alta, Mohd Faizal Abdollah, Othman Abdollah, and SM Warusia Mohamed SMM Yassin. "Early Detection of Windows Cryptographic Ransomware Based on Pre-Attack API Calls Features and Machine Learning." *Journal of Advanced Research in Applied Sciences and Engineering Technology* 39, no. 2 (2024): 110-131. https://doi.org/10.37934/araset.39.2.110131

[5] Jalil, Masita, Noraida Hj Ali, Farizah Yunus, Fakhrul Adli Mohd Zaki, Lee Hwee Hsiung, and Mohammed Amin Almaayah. "Cybersecurity Awareness among Secondary School Students Post Covid-19 Pandemic." *Journal of Advanced Research in Applied Sciences and Engineering Technology* 37, no. 1 (2024): 115-127. https://doi.org/10.37934/araset.37.1.115127

[6] Asmawi, Aziah, Ezzah Mawadah Saifulbahri, and Noor Afiza Mohd Ariffin. "Development of BlockScholar as an Educational Mobile Application on Blockchain Technology." *Journal of Advanced Research in Applied Sciences and Engineering Technology* 34, no. 1 (2024): 15-23. https://doi.org/10.37934/araset.34.1.1523

[7] Ismail, Nuur Ezaini Akmar, Noraida Haji Ali, Masita Abdul Jalil, Farizah Yunus, and Ahmad Dahari Jarno. "A Proposed Framework of Vulnerability Assessment and Penetration Testing (VAPT) in Cloud Computing Environments from Penetration Tester Perspective." *Journal of Advanced Research in Applied Sciences and Engineering Technology* 39, no. 1 (2024): 1-14. https://doi.org/10.37934/araset.39.1.114

[8] OWASP, "OWASP Top 10:2021," *owasp.org*, (2021). https://owasp.org/Top10

[9] Gaurav, Devottam, Yash Kaushik, Santhoshi Supraja, Abhi Khandelwal, Karsheet Negi, Manmohan Prasad Gupta, and Manmohan Chaturvedi. "Cybersecurity training for web applications through serious games." In *2021 IEEE International Conference on Engineering, Technology & Education (TALE)*, pp. 390-398. IEEE, 2021. https://doi.org/10.1109/TALE52509.2021.9678531

[10] Marashdeh, Zain, Khaled Suwais, and Mohammad Alia. "A survey on sql injection attack: Detection and challenges." In *2021 International Conference on Information Technology (ICIT)*, pp. 957-962. IEEE, 2021. https://doi.org/10.1109/ICIT52682.2021.9491117

[11] Tanakas, Petros, Aristidis Ilias, and Nineta Polemi. "A novel system for detecting and preventing SQL injection and cross-site-script." In *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pp. 1-6. IEEE, 2021. https://doi.org/10.1109/ICECET52533.2021.9698688

[12] Singh, Nikhil Kumar, Prasoon Gupta, Vaibhav Singh, and Raju Ranjan. "Attacks on Vulnerable Web Applications." In *2021 International Conference on Intelligent Technologies (CONIT)*, pp. 1-5. IEEE, 2021. https://doi.org/10.1109/CONIT51480.2021.9498396

[13] Srivastava, Mayank, Animesh Raghuvanshi, and Dhruv Khandelwal. "Security and Scalability of E-Commerce Website by OWASP threats." In *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*, pp. 1-8. IEEE, 2023. https://doi.org/10.1109/ISCON57294.2023.10111955

[14] Shanmugasundaram, G., S. Ravivarman, and P. Thangavellu. "A study on removal techniques of Cross-Site Scripting from web applications." In *2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)*, pp. 0436-0442. IEEE, 2015. https://doi.org/10.1109/ICCPEIC.2015.7259498

[15] Petranović, Teodora, and Nikola Žarić. "Effectiveness of Using OWASP TOP 10 as AppSec Standard." In *2023 27th International Conference on Information Technology (IT)*, pp. 1-4. IEEE, 2023. https://doi.org/10.1109/IT57431.2023.10078626

[16] Lala, Shubham Kumar, Akshat Kumar, and T. Subbulakshmi. "Secure web development using owasp guidelines." In *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 323-332. IEEE, 2021. https://doi.org/10.1109/ICICCS51141.2021.9432179

[17] Rai, Aditya, MD Mazharul Islam Miraz, Deshbandhu Das, and Harpreet Kaur. "Sql injection: Classification and prevention." In *2021 2nd International conference on Intelligent Engineering and Management (ICIEM)*, pp. 367-372. IEEE, 2021. https://doi.org/10.1109/ICIEM51511.2021.9445347

[18] Nomura, Komei, Kenji Rikitake, and Ryosuke Matsumoto. "Automatic whitelist generation for sql queries using web application tests." In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 465-470. IEEE, 2019. https://doi.org/10.1109/COMPSAC.2019.10250

[19] Castillo, Reynaldo E., Jasmin A. Caliwag, Roxanne A. Pagaduan, and Aira Camille Nagua. "Prevention of SQL injection attacks to login page of a website application using prepared statement technique." In *Proceedings of the 2nd International Conference on Information Science and Systems*, pp. 171-175. 2019. https://doi.org/10.1145/3322645.3322704

[20] Mavromoustakos, Stephanos, Aakash Patel, Kinjal Chaudhary, Parth Chokshi, and Shaili Patel. "Causes and prevention of SQL injection attacks in web applications." In *Proceedings of the 4th International Conference on Information and Network Security*, pp. 55-59. 2016. https://doi.org/10.1145/3026724.3026742

[21] Dovgalyuk, Pavel, Maria Klimushenkova, Natalia Fursova, Ivan Vasiliev, and Vladislav Stepanov. "Natch: Detecting Attack Surface for Multi-Service Systems with Hybrid Introspection." In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, pp. 176-185. IEEE, 2023. https://doi.org/10.1109/QRS-C60940.2023.00023

[22] Ping, Chen, Wang Jinshuang, Pan Lin, and Yu Han. "Research and implementation of SQL injection prevention method based on ISR." In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 1153-1156. IEEE, 2016.

[23] Hadabi, Abdalla, Eltyeb Elsamani, Ali Abdallah, and Rashad Elhabob. "An efficient model to detect and prevent SQL injection attack." *Journal of Karary University for Engineering and Science* (2022). https://doi.org/10.54388/jkues.v1i2.141
[24] Achour, Medhi *et al.,* "PHP Manual". *php.net*, (2023). https://www.php.net/manual/en/index.php
[25] Christou, George, Giorgos Vasiliadis, Vassilis Papaefstathiou, Antonis Papadogiannakis, and Sotiris Ioannidis. "On architectural support for instruction set randomization." *ACM Transactions on Architecture and Code Optimization (TACO)* 17, no. 4 (2020): 1-26. https://doi.org/10.1145/3419841
[26] Wu, Haoqi, Zhengxuan Yu, Dapeng Huang, Haodong Zhang, and Weili Han. "Automated enforcement of the principle of least privilege over data source access." In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 510-517. IEEE, 2020. https://doi.org/10.1109/TrustCom50675.2020.00075
[27] Mayhew, B., Baars, N., White, J. and Zubčević, "OWASP WebGoat," *Owasp.org,* (2020). https://owasp.org/www-project-webgoat
[28] RandomStorm and R.Dewhurst, "Damn Vulnerable Web Application", *DVMA*, (2015). https://github.com/digininja/DVWA
[29] Kimminich, B. "OWASP Juice Shop", *Owasp.org*, (2022). https://github.com/juice-shop/juice-shop
[30] "Hacksplaining: Web Security for Developers," *Hacksplaining*. https://www.hacksplaining.com
[31] Common Weakness Enumeration, "CWE VIEW: Weakness in OWASP Top Ten 2021", *cwe.mitre.org* (2023). https://cwe.mitre.org/data/definitions/1344.html
[32] Bangor, Aaron, Philip Kortum, and James Miller. "Determining what individual SUS scores mean: Adding an adjective rating scale." *Journal of usability studies* 4, no. 3 (2009): 114-123.
[33] Kortum, Philip, and Claudia Ziegler Acemyan. "How low can you go? Is the system usability scale range restricted?." *Journal of Usability Studies* 9, no. 1 (2013).